MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025

Table of contents

1 MIMXRT1180-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 . . . 58
  - 1.3.1 Getting Started with MCUXpresso SDK Repository . . . 58
- 1.4 Release Notes . . . 71
  - 1.4.1 MCUXpresso SDK Release Notes . . . 71
- 1.5 ChangeLog . . . 78
  - 1.5.1 MCUXpresso SDK Changelog . . . 78
- 1.6 Driver API Reference Manual . . . 209
- 1.7 Middleware Documentation . . . 209
  - 1.7.1 Multicore . . . 209
  - 1.7.2 MCU Boot . . . 209
  - 1.7.3 eIQ . . . 209
  - 1.7.4 FreeMASTER . . . 209
  - 1.7.5 AWS IoT . . . 209
  - 1.7.6 NXP Wi-Fi . . . 209
  - 1.7.7 FreeRTOS . . . 209
  - 1.7.8 Wireless EdgeFast Bluetooth PAL . . . 209
  - 1.7.9 lwIP . . . 210
  - 1.7.10 File systemFatfs . . . 210

2 MIMXRT1186 **211**
- 2.1 ACMP: Analog Comparator Driver . . . 211
- 2.2 AOI: Crossbar AND/OR/INVERT Driver . . . 217
- 2.3 ASRC: Asynchronous sample rate converter . . . 220
- 2.4 ASRC Driver . . . 220
- 2.5 ASRC EDMA Driver . . . 236
- 2.6 Battery-Backed Non-Secure Module . . . 240
- 2.7 CACHE: ARMV7-M7 CACHE Memory Controller . . . 242
- 2.8 DAC12: 12-bit Digital-to-Analog Converter Driver . . . 245
- 2.9 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 251
- 2.10 eDMA core Driver . . . 282
- 2.11 eDMA soc Driver . . . 289
- 2.12 EQDC: Enhanced Quadrature Encoder/Decoder . . . 289
- 2.13 EWM: External Watchdog Monitor Driver . . . 307
- 2.14 FGPIO Driver . . . 310
- 2.15 FlexCAN: Flex Controller Area Network Driver . . . 310
- 2.16 FlexCAN Driver . . . 310
- 2.17 FlexCAN eDMA Driver . . . 358
- 2.18 FlexIO: FlexIO Driver . . . 361
- 2.19 FlexIO Driver . . . 361
- 2.20 FlexIO eDMA I2S Driver . . . 378
- 2.21 FlexIO eDMA SPI Driver . . . 382
- 2.22 FlexIO eDMA UART Driver . . . 385

2.23 FlexIO I2C Master Driver . . . 388
2.24 FlexIO I2S Driver . . . 396
2.25 FlexIO SPI Driver . . . 407
2.26 FlexIO UART Driver . . . 420
2.27 FLEXSPI: Flexible Serial Peripheral Interface Driver . . . 430
2.28 FLEXSPI eDMA Driver . . . 447
2.29 FLEXSPI_FLR: Flexible Serial Peripheral Interface Follower Driver . . . 449
2.30 GPT: General Purpose Timer . . . 457
2.31 I3C: I3C Driver . . . 464
2.32 I3C Common Driver . . . 466
2.33 I3C Master Driver . . . 468
2.34 I3C Master DMA Driver . . . 494
2.35 I3C Slave Driver . . . 497
2.36 I3C Slave DMA Driver . . . 510
2.37 IEE: Inline Encryption Engine . . . 512
2.38 Ieer . . . 516
2.39 KPP: KeyPad Port Driver . . . 518
2.40 Common Driver . . . 520
2.41 LPADC: 12-bit SAR Analog-to-Digital Converter Driver . . . 532
2.42 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 553
2.43 LPI2C Master Driver . . . 554
2.44 LPI2C Master DMA Driver . . . 569
2.45 LPI2C Slave Driver . . . 571
2.46 LPIT: Low-Power Interrupt Timer . . . 581
2.47 LPSPI: Low Power Serial Peripheral Interface . . . 588
2.48 LPSPI Peripheral driver . . . 588
2.49 LPSPI eDMA Driver . . . 610
2.50 LPTMR: Low-Power Timer . . . 617
2.51 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 622
2.52 LPUART Driver . . . 622
2.53 LPUART eDMA Driver . . . 641
2.54 MECC: internal error correction code . . . 644
2.55 MSGINTR: Message Unit . . . 651
2.56 MSGINTR Driver . . . 651
2.57 MU: Messaging Unit . . . 651
2.58 NETC driver . . . 664
2.59 Abbreviation in NETC driver . . . 669
2.60 API layer . . . 669
2.61 NETC Endpoint (EP) Driver . . . 669
2.62 Endpoint (EP) Generic Configuration . . . 669
2.63 Endpoint (EP) data path . . . 674
2.64 Endpoint (EP) Interrupt Module . . . 675
2.65 Endpoint (EP) Table Management Module . . . 676
2.66 Endpoint (EP) PSI/VSI . . . 676
2.67 Endpoint (EP) Ingress data path configuration . . . 679
2.68 Endpoint (EP) Statistic Module . . . 691
2.69 Endpoint (EP) Egress data path configuration . . . 692
2.70 Endpoint (EP) Transmit/Receive . . . 694
2.71 Hardware layer . . . 699
2.72 Hardware Common Functions . . . 708
2.73 Hardware ENETC . . . 708
2.74 Hardware Port . . . 714
2.75 Hardware Port MAC . . . 726
2.76 Hardware Port Rx . . . 733
2.77 Hardware Port Tx . . . 734
2.78 Hardware Station Interface(SI) . . . 735
2.79 Hardware Switch . . . 752
2.80 Hardware Table Access Functions . . . 758

2.81 NETC MDIO Driver . . . 838
2.82 MDIO initialization module . . . 838
2.83 MDIO PHY status module . . . 839
2.84 MDIO write/read module . . . 840
2.85 NETC SWITCH (SWT) Driver . . . 841
2.86 Switch (SWT) data path . . . 841
2.87 Switch (SWT) Generic Configuration . . . 841
2.88 Switch (SWT) Interrupt Module . . . 846
2.89 Switch (SWT) management . . . 846
2.90 Switch (SWT) Table Management Module . . . 858
2.91 Switch (SWT) Ingress data path configuration . . . 859
2.92 Switch (SWT) Statistic Module . . . 872
2.93 Switch (SWT) Egress data path configuration . . . 876
2.94 Switch (SWT) Transmit/Receive . . . 881
2.95 NETC Timer Driver . . . 885
2.96 Timer adjustment module . . . 885
2.97 Timer initialization module . . . 886
2.98 Local time synchronization module . . . 888
2.99 OTFAD: On The Fly AES-128 Decryption Driver . . . 892
2.100 PDM: Microphone Interface . . . 895
2.101 PDM Driver . . . 895
2.102 PDM EDMA Driver . . . 913
2.103 PWM: Pulse Width Modulator . . . 917
2.104 QTMR: Quad Timer Driver . . . 941
2.105 RGPIO: Rapid General-Purpose Input/Output Driver . . . 952
2.106 RGPIO Driver . . . 954
2.107 RTWDOG: 32-bit Watchdog Timer . . . 958
2.108 S3MU: Sentinel . . . 964
2.109 SAI: Serial Audio Interface . . . 966
2.110 SAI Driver . . . 966
2.111 SAI EDMA Driver . . . 992
2.112 SEMA42: Hardware Semaphores Driver . . . 999
2.113 SEMC: Smart External DRAM Controller Driver . . . 1002
2.114 SINC: SINC Filter . . . 1027
2.115 SPDIF: Sony/Philips Digital Interface . . . 1058
2.116 SPDIF eDMA Driver . . . 1072
2.117 SRAMC: SRAMC Driver . . . 1075
2.118 Sysctr . . . 1078
2.119 SYSCTR: System Counter Driver . . . 1083
2.120 SYSPM: System Performance Monitor . . . 1083
2.121 TEMPSENSOR: Temperature Sensor Module . . . 1086
2.122 TPM: Timer PWM Module . . . 1089
2.123 TRDC: Trusted Resource Domain Controller . . . 1105
2.124 TSTMR: Timestamp Timer Driver . . . 1126
2.125 USDHC: Ultra Secured Digital Host Controller Driver . . . 1126
2.126 VREF: Voltage Reference Driver . . . 1156
2.127 Xbar . . . 1161
2.128 CACHE: CACHE Memory Controller . . . 1163

3 Middleware 1167
3.1 Boot . . . 1167
3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource . . . 1167
3.1.2 MCUboot . . . 1168
3.2 eIQ . . . 1169
3.2.1 eIQ . . . 1169
3.3 Motor Control . . . 1197
3.3.1 FreeMASTER . . . 1198
3.4 Wireless . . . 1235

3.4.1 NXP Wireless Framework and Stacks . . . 1235
3.4.2 EdgeFast Bluetooth . . . 1235

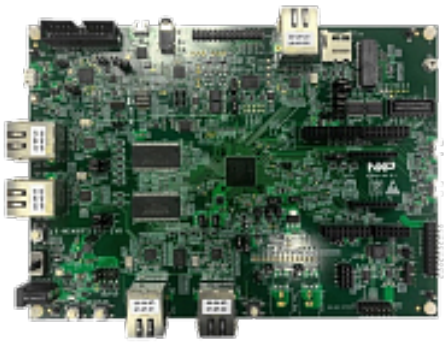
4 RTOS **1311**
4.1 FreeRTOS . . . 1311
4.1.1 FreeRTOS kernel . . . 1311
4.1.2 FreeRTOS drivers . . . 1311
4.1.3 backoffalgorithm . . . 1311
4.1.4 corehttp . . . 1311
4.1.5 corejson . . . 1311
4.1.6 coremqtt . . . 1312
4.1.7 corepkcs11 . . . 1312
4.1.8 freertos-plus-tcp . . . 1312

This documentation contains information specific to the evkmimxrt1180 board.

Chapter 1

MIMXRT1180-EVK

1.1 Overview



MCU device and part on board is shown below:

- Device: MIMXRT1189
- PartNumber: MIMXRT1189CVM8B

1.2 Getting Started with MCUXpresso SDK Package

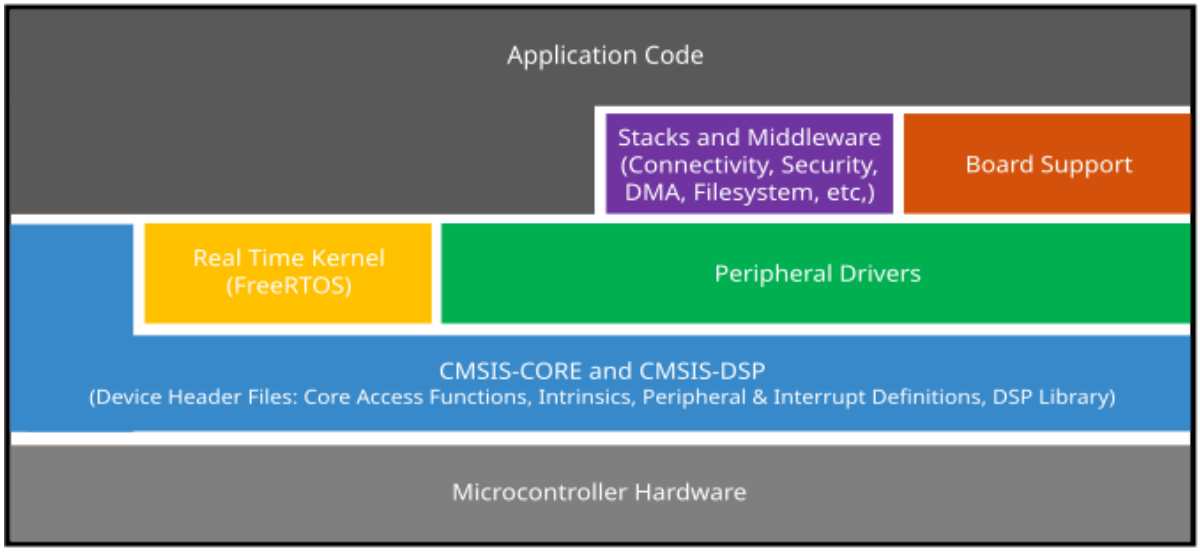
1.2.1 Getting Started with Package

Overview

The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for NXP Microcontrollers. 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 FreeRTOS and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for MIMXRT1180-EVK* (document MCUXSDKMIMXRT118XKRN).

For more details about MCUXpresso SDK, see MCUXpresso Software Development Kit (SDK).



MCUXpresso SDK board support package folders

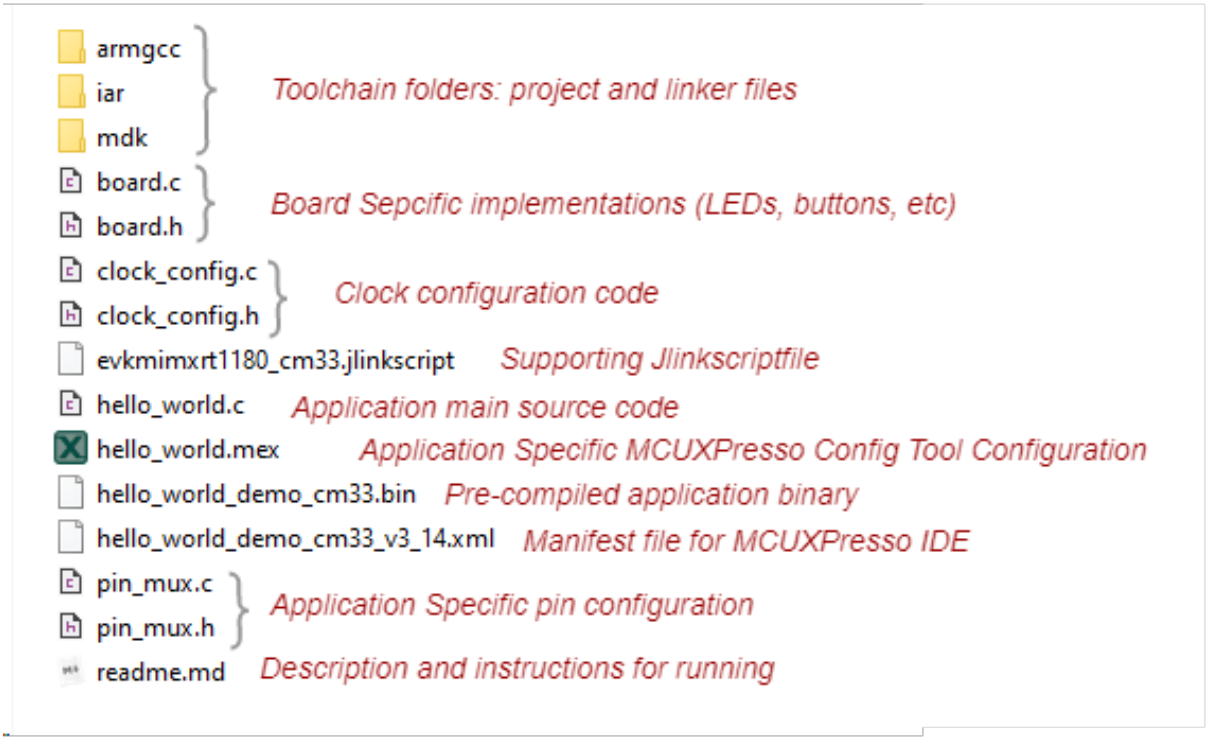
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores. Board support packages are found inside the top level boards folder and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each <board_name> folder, there are various sub-folders to classify the type of examples it contains. These types include (but are not limited to):

- cmsis_driver_examples: Simple applications intended to show how to use CMSIS drivers.
- demo_apps: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications that show how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- rtos_examples: Basic FreeRTOS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers.
- Other_examples: See detail in package *boards/evkmimxrt1180*.

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.

The *hello_world* application folder contains 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 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 CMSIS header file of the device, MCUXpresso SDK feature file and a few other files
- devices/<device_name>/drivers: All 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_template 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 MIMXRT1180-EVK hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

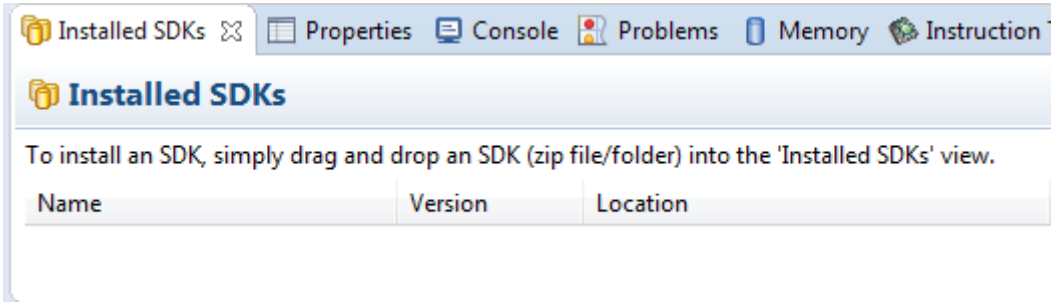
Both CMSIS-DAP and J-Link debugging interface is supported for MCUX IDE. When using CMSIS-DAP debugging interface to debug CM33, the SW5[1..4] should be put to 0100 for SPI boot mode.

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, follow these steps.

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



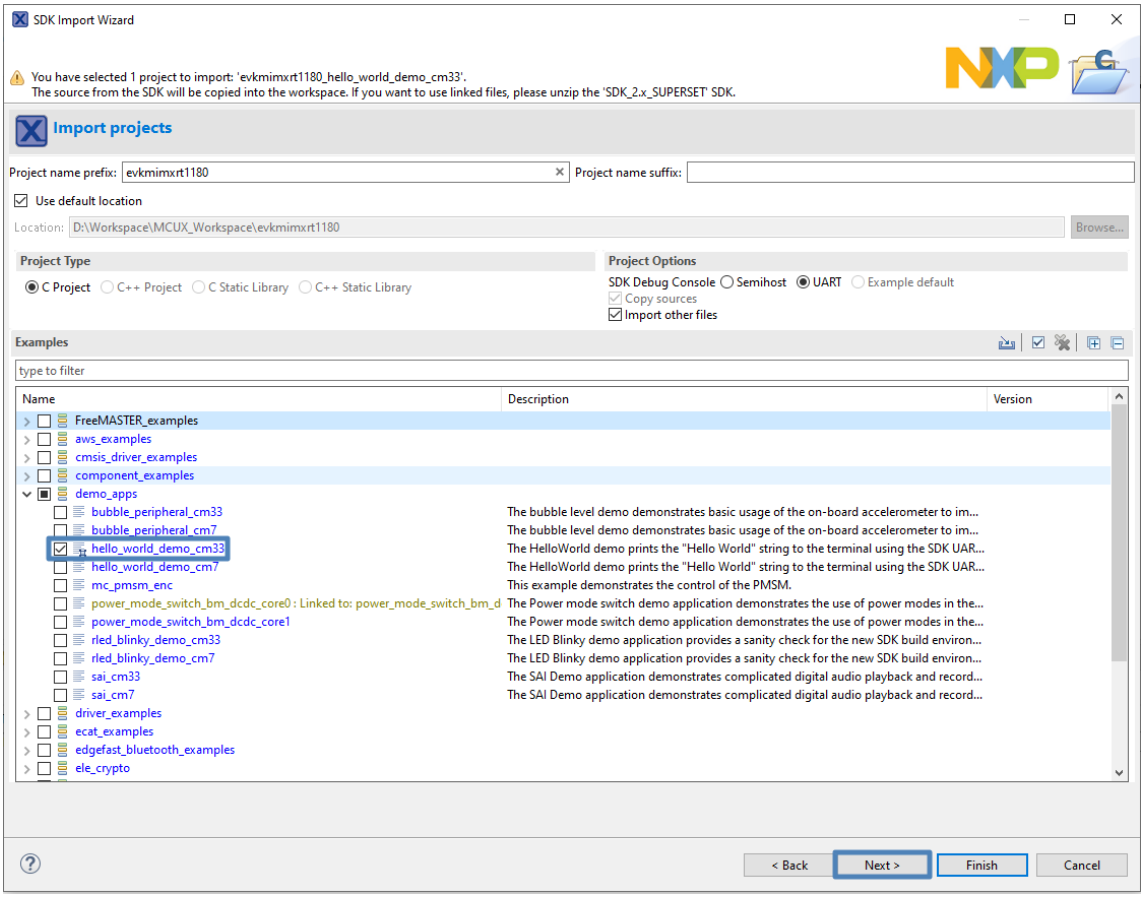
2. On the **Quickstart Panel**, click **Import SDK example(s)...**.



3. In the window that appears, select **evkmimxrt1180** or **kits** and click **Next**.



4. Expand the *demo_apps* folder and select hello_world_demo_cm33. Then, click **Next**.



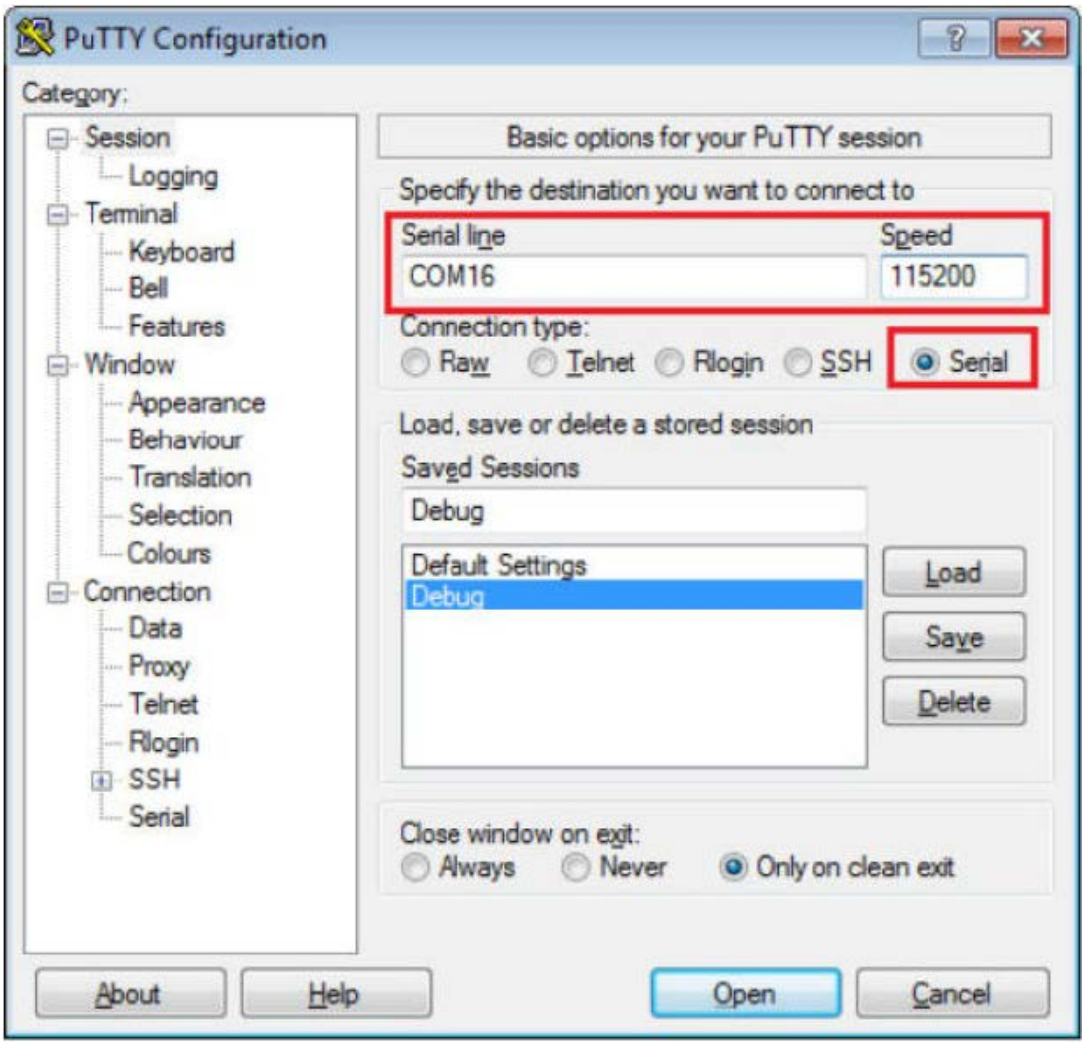
5. Click **Finish**.

Parent topic:*Run a demo using MCUXpresso IDE*

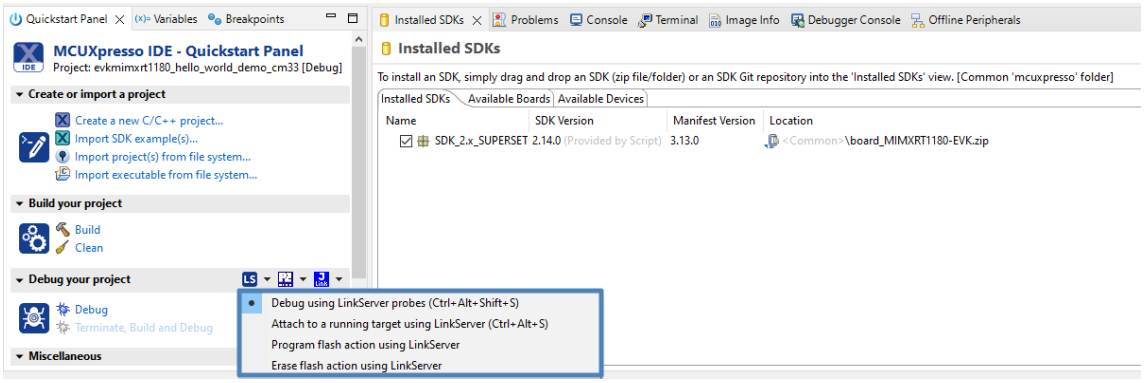
Run an example application To download and run the application, perform the following steps:

1. See *Table 1* to determine the debug interface that comes loaded on your specific hardware platform.
   - For EVKMIMXRT1180, LPC55S69 is used.
   - J53 is used as the debugging port which provides both debugging functionality and console out.
2. Connect USB cable between J53 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:
   1. 115200 baud rate 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

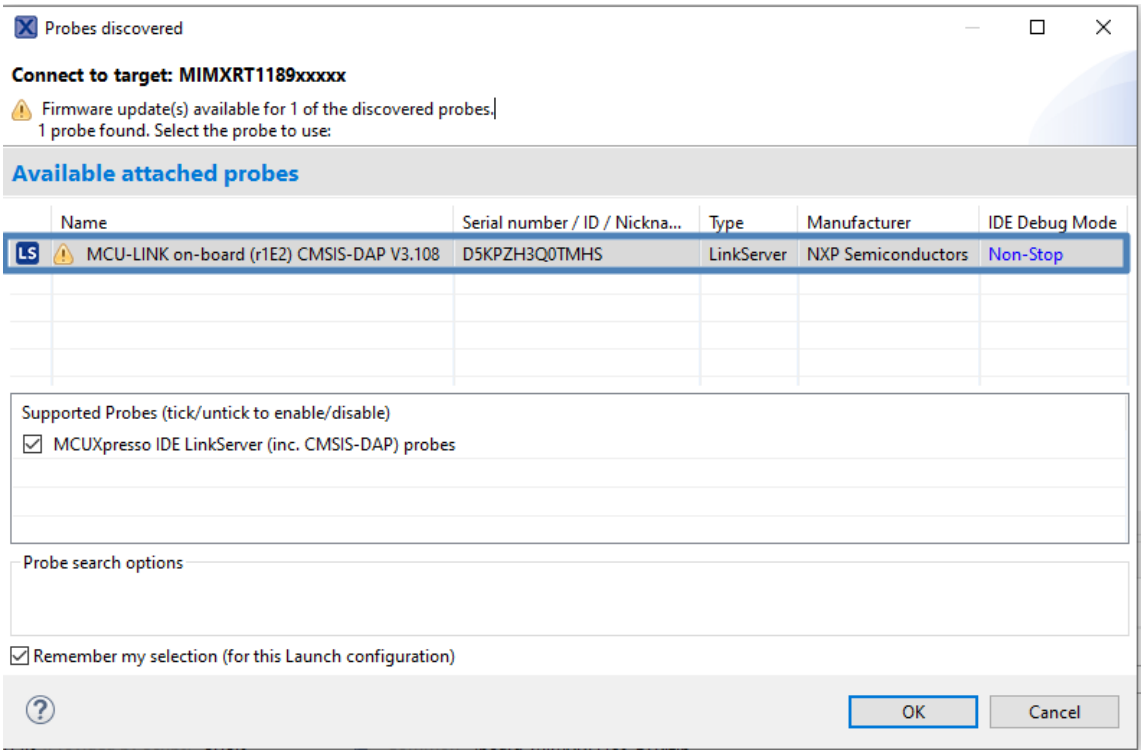


4. On the **Quickstart Panel**, click **Debug**.

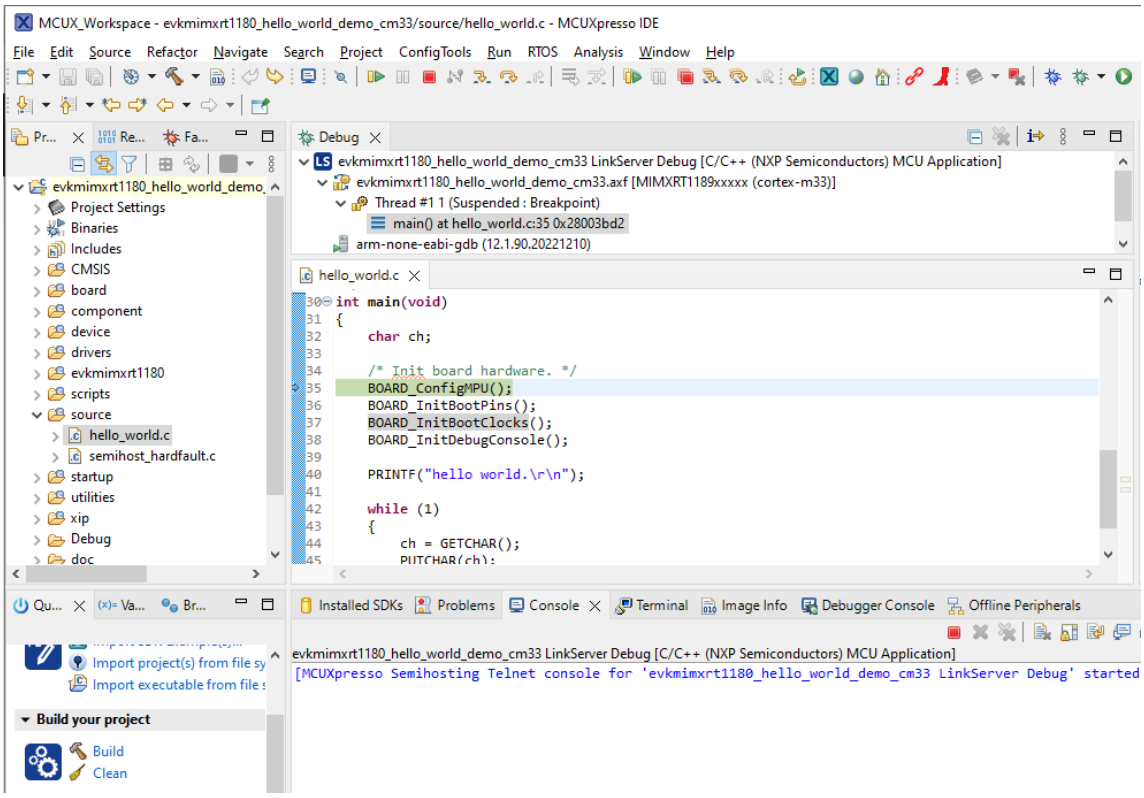


Note: Erase the flash memory before running the MCUX IDE projects for the first time. To do so, use the **Erase Flash action using LinkServer** in *Figure 2* with SW5[1..4] on 0100. In the worst case when MCUX IDE cannot erase the flash, use the secure provisioning tool. For more information, see Use Secure Provisiong Tool to erase flash.

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**. Here we assume that you are using a CMSIS-DAP debug interface. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)



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

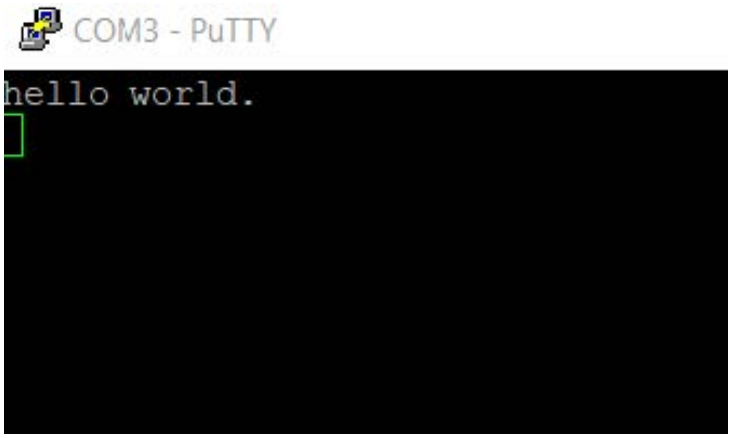


7. Start the application by clicking **Resume**.



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

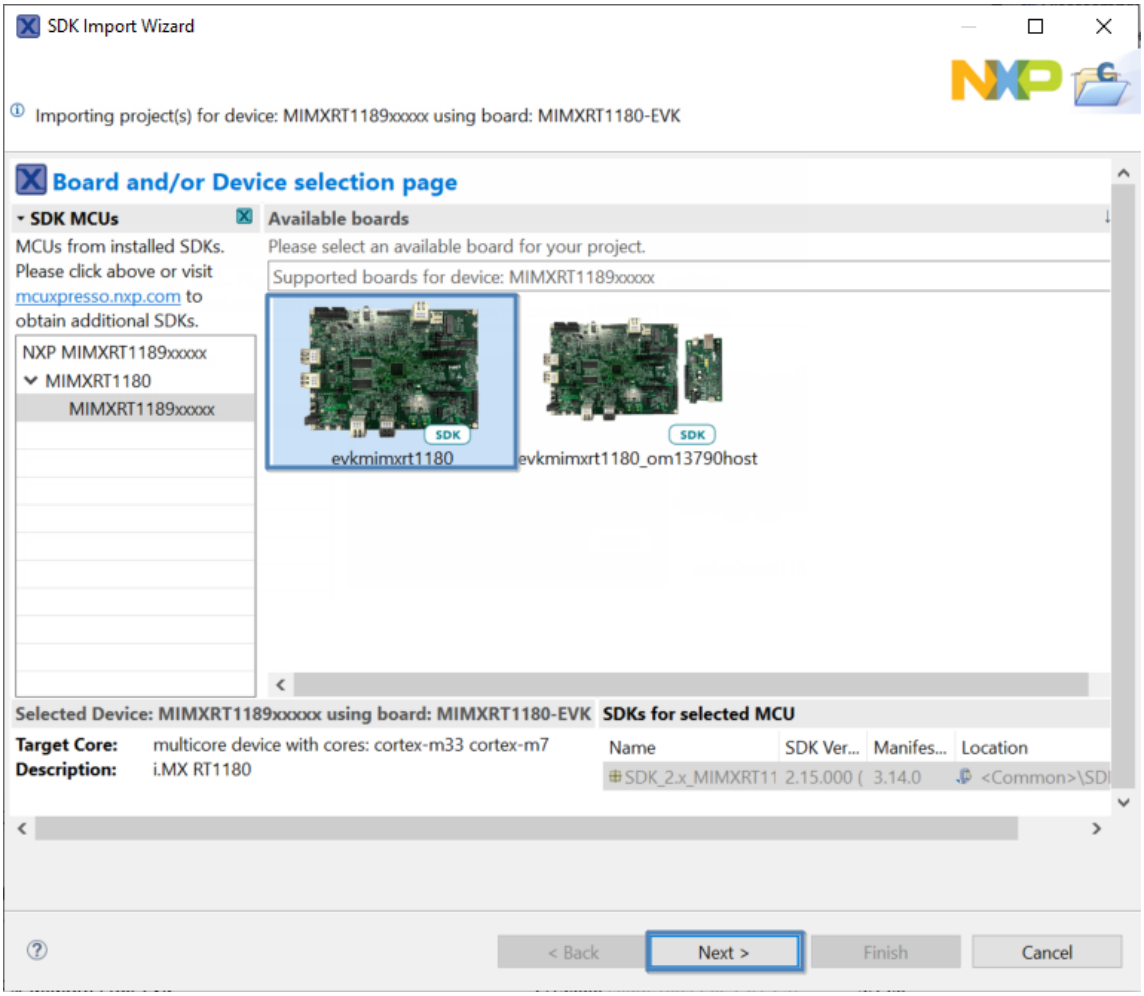
check your terminal settings and connections.



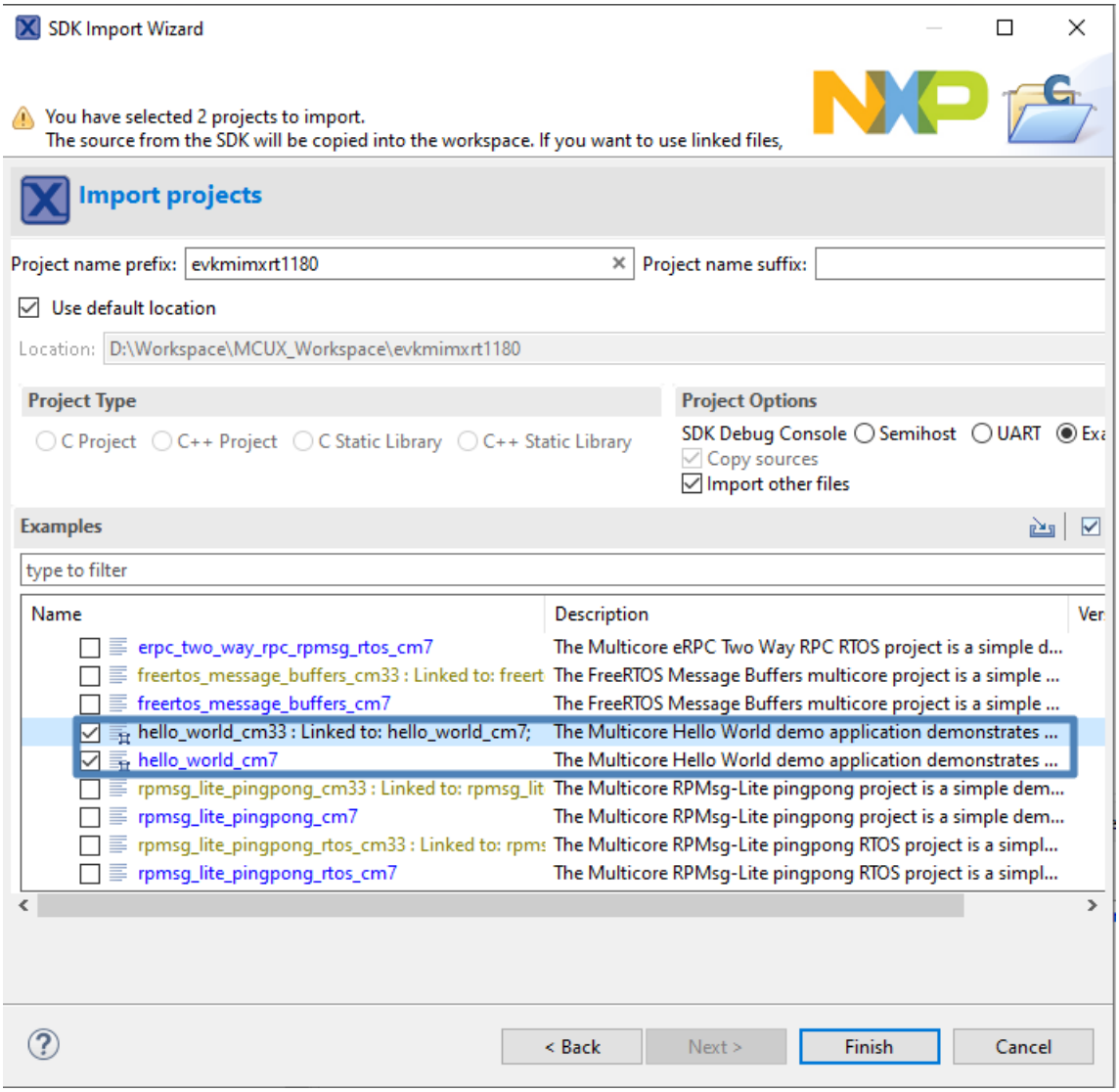
Parent topic:*Run a demo using MCUXpresso IDE*

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 **evkmimxrt1180** hardware platform is used as an example.

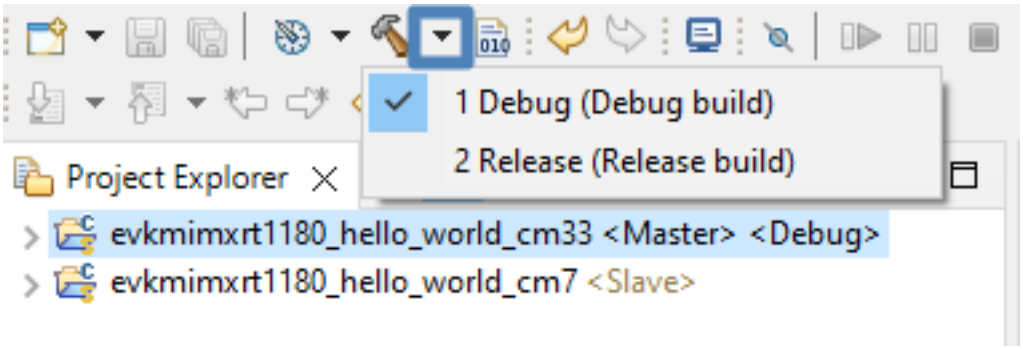
1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in Build an example application. When the SDK zip package for **evkmimxrt1180** is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, select **evkmimxrt1180** and click **Next**.



2. Expand the multicore_examples folder and select **hello_world_cm33**. The hello_world_cm7 counterpart project is automatically imported with the cm33 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 multi-core application, highlight the hello_world_cm33 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 3*. For this example, select **Debug**.

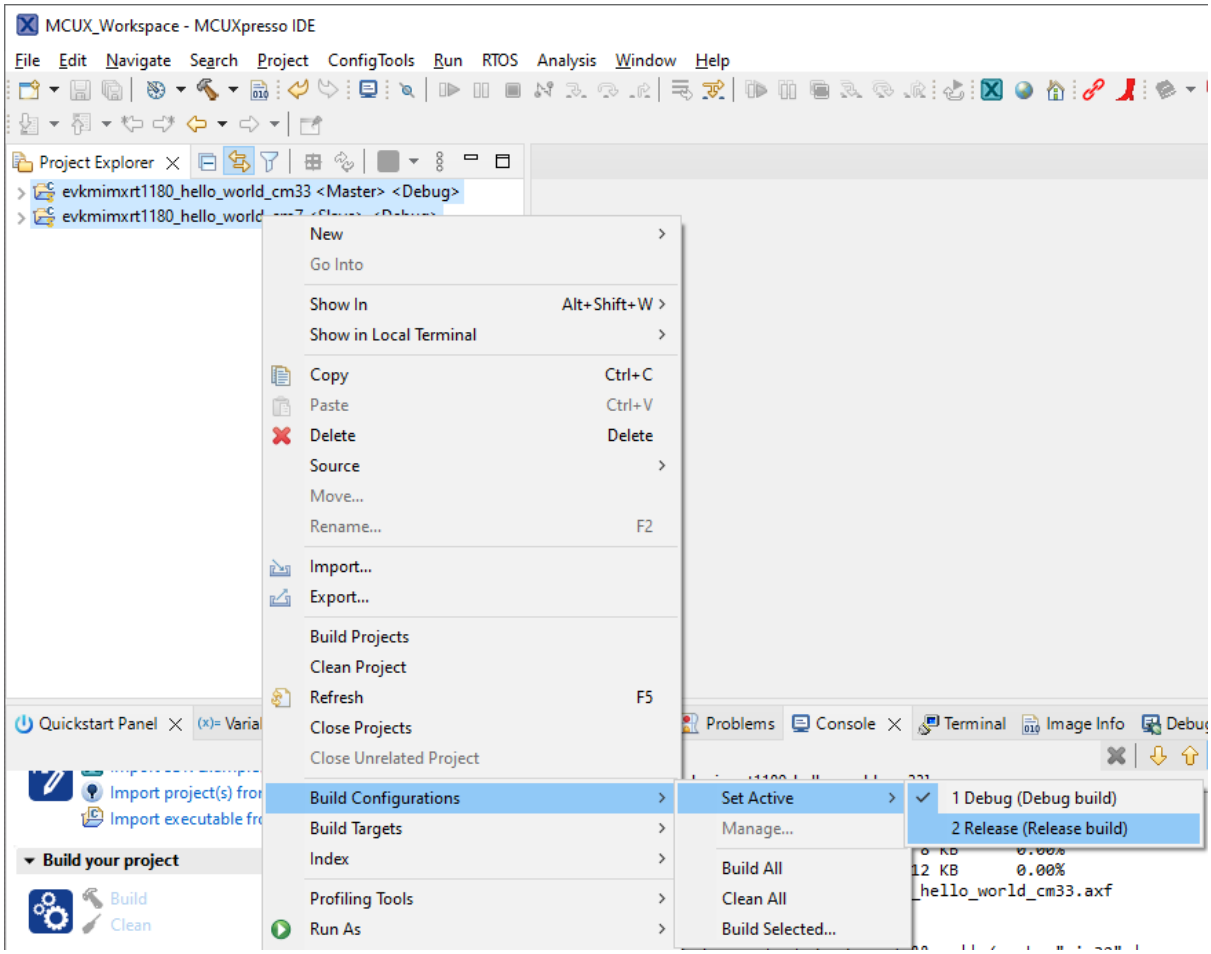


Press the **Build** button to start the multi-core project build.. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm33) also makes the referenced auxiliary core application (cm7) to build.

Note:

When the **Release** build is requested, it is necessary to change the build configuration of both

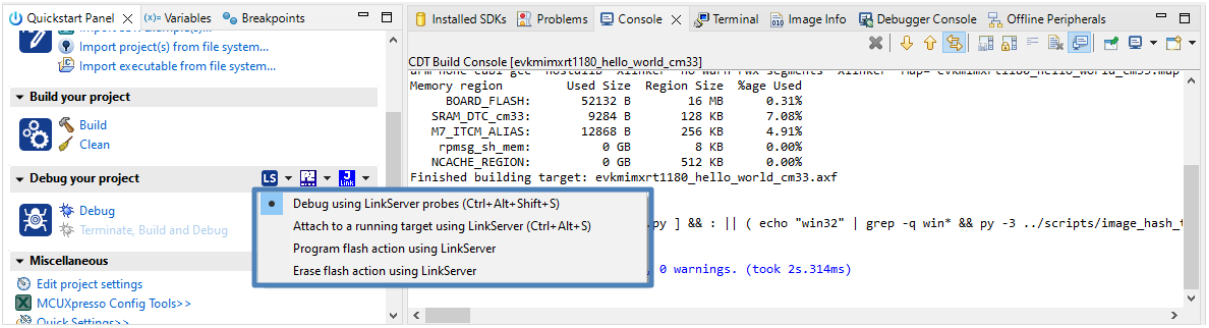
the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build Configurations** -> **Set Active** -> **Release**. This alternate navigation using the menu item is **Project** -> **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) build.

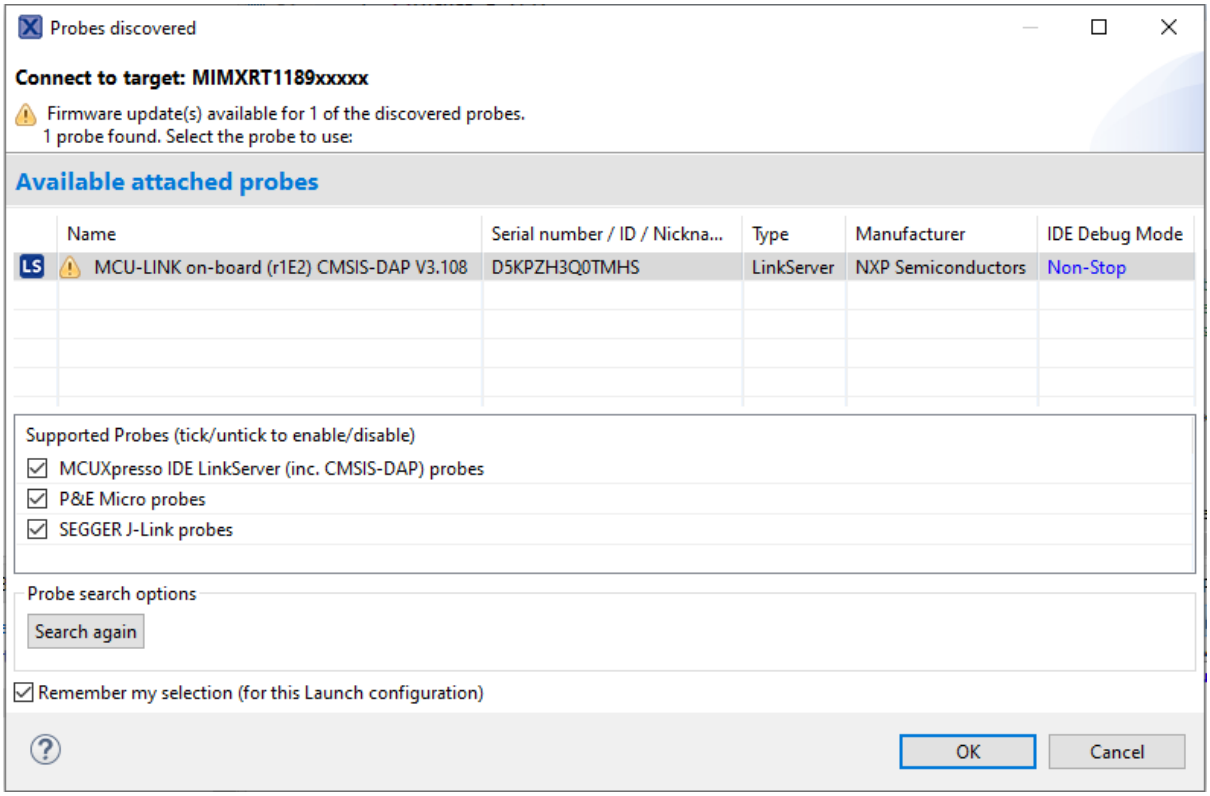


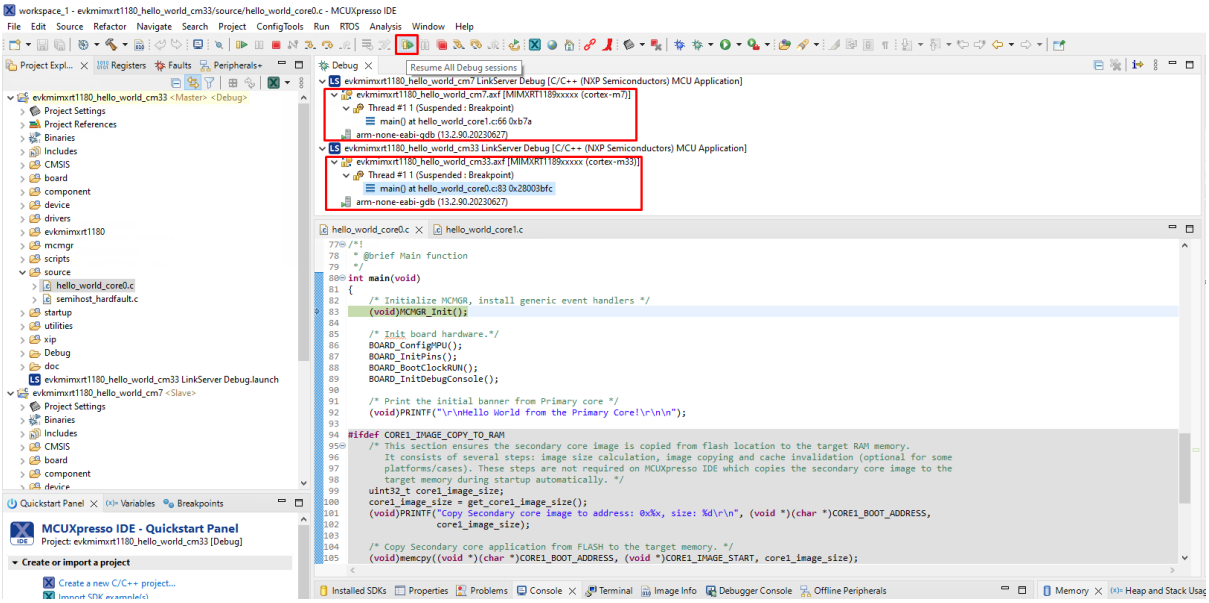
Parent topic:*Run a demo using MCUXpresso IDE*

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 cm7 project and start debugging the CM7 project. Then, select the cm33 project and start debugging the CM33 project.

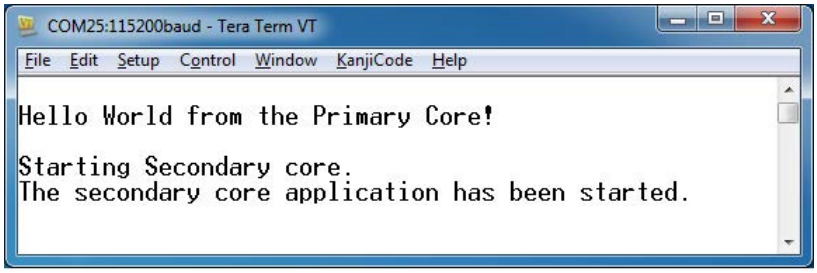
See *Figure 1* to *Figure 3* for reference.







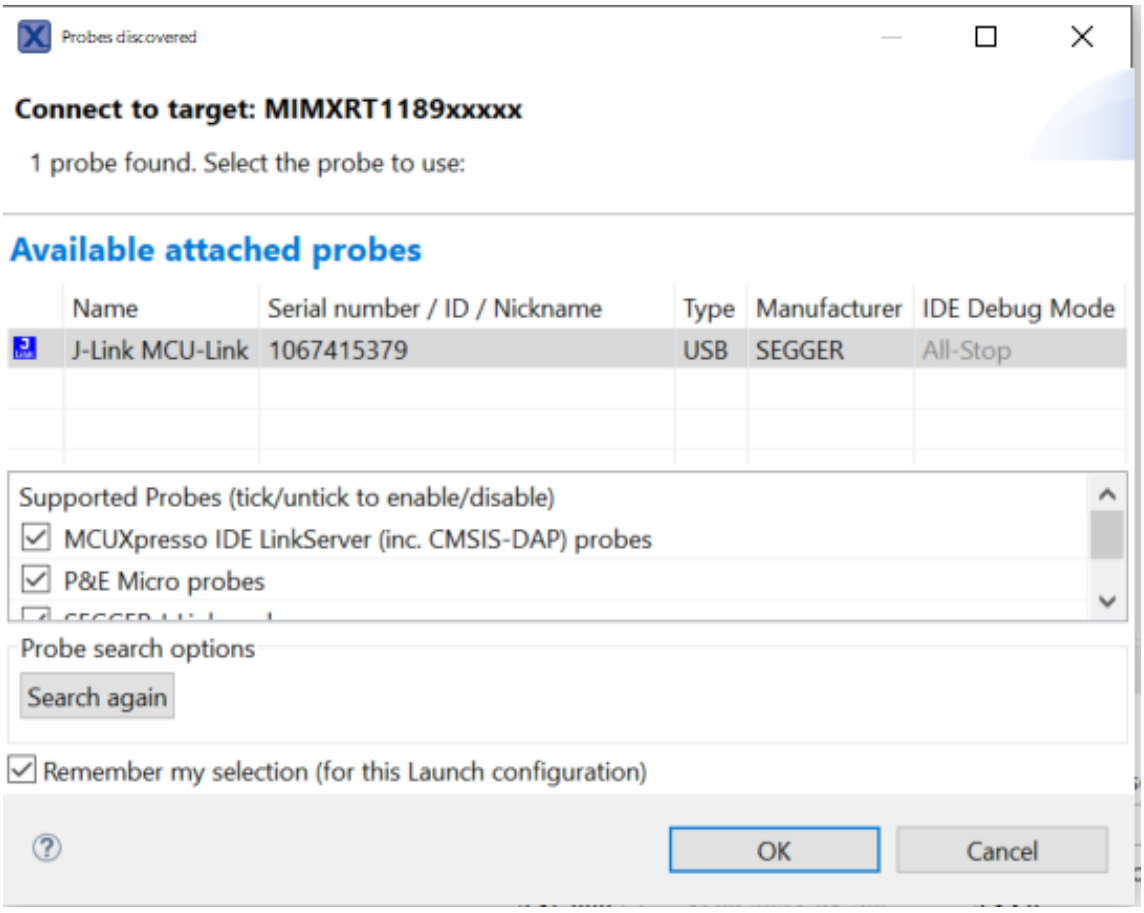
After clicking **Resume All Debug** sessions, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



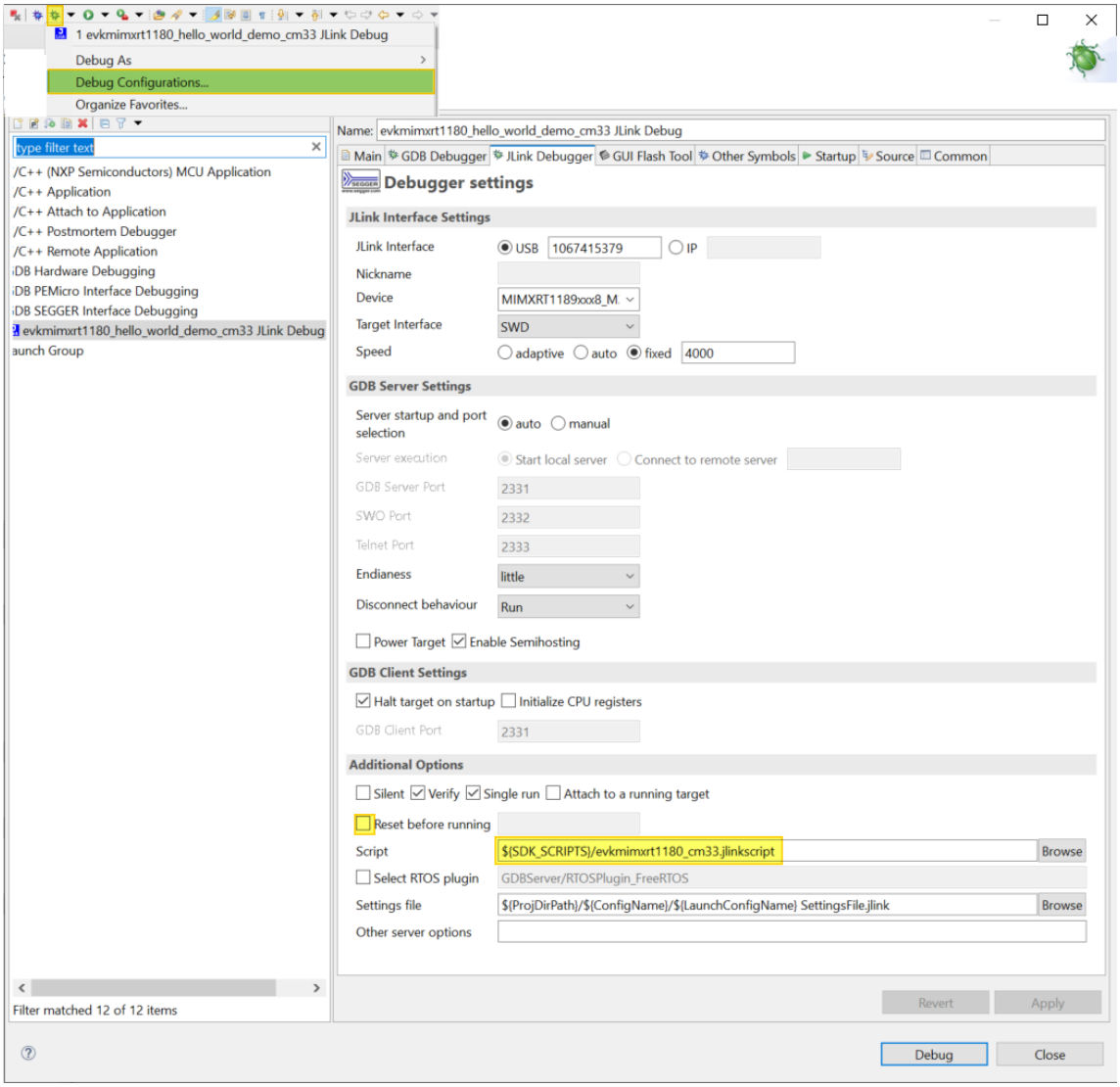
Parent topic:*Run a demo using MCUXpresso IDE*

Run applications via JLink Debug Interface The previous sections assume that you are using the CMSIS-DAP debugging interface for MCUXpresso IDE to do the debugging. If you are using J-Link probe as a debugging interface, the configurations are slightly different.

1. After you click the **Debug** button, the IDE shows the J-Link probe identified. Click **OK** to continue the debugging.



2. If you investigate on debug configurations, you will notice two settings, which are different from CMSIS-DAP debugging. The **Reset before running** is unchecked, and a jlinkscript is specified. This is because of the difference in CMSIS-DAP and JLink debugging implementation. The differences are pre-configured by MCUX default project settings.



3. The screenshot here shows a CM33 demo. For CM7 demo, situations are similar.

Parent topic:*Run a demo using MCUXpresso IDE*

Run a demo using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. This document uses hello_world demo application targeted for the MIMXRT1180-EVK as an example. These steps can be applied to any example application in the MCUXpresso SDK.

Both CMSIS-DAP and J-Link debugging interfaces are supported for IAR IDE. Thoug not mandatory, it is recommended to reset board for each download/debug.

Changes on IAR debugging support after 25.03.00 Starting from 25.03.00, we make enhancement to debugging experience to make it more robust and user friendly

These changes involves debugging code update on both SDK side and IAR toolchain side. It is expected to be integrated to IAR Embedded Workbench for Arm 9.60.4.

Update to latest IAR version You can check IAR Product Updates to know when IAR Embedded Workbench for Arm 9.60.4 will be rolled out. Befor that, the following manual change is needed

inside you IAR installation directory.

Necessary Change in your IAR installation (<= 9.60.3)

1. Navigate to <iar installation directory>/arm/config/devices/NXP/i.MX/i.MXRT/, change MIMXRT1189xxx8_M33.i79 and MIMXRT1189xxx8_M7.i79, commenting out the following lines

```
// JLinkScriptFile=$TOOLKIT_DIR$/config/debugger/NXP/mimxrt1180_cm33.jlinkscript
// JLinkScriptFile=$TOOLKIT_DIR$/config/debugger/NXP/mimxrt1180_cm7.jlinkscript
```

2. Replace the following files under <iar installation directory>/arm/config/debugger/NXP with the files in <SDK root>/boards/evkmimxrt1180/dmac/iMXRT_1180.dmac

```
iMXRT_1180.dmac
iMXRT_1180_cm33.dmac
iMXRT_1180_cm7.dmac
```

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

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>/<core_type>/iar

   Using the MIMXRT1180-EVK hardware platform as an example, the hello_world workspace is located in:

   <install_dir>/boards/evkmimxrt1180/demo_apps/hello_world/cm33/iar/hello_world_demo_cm33.eww

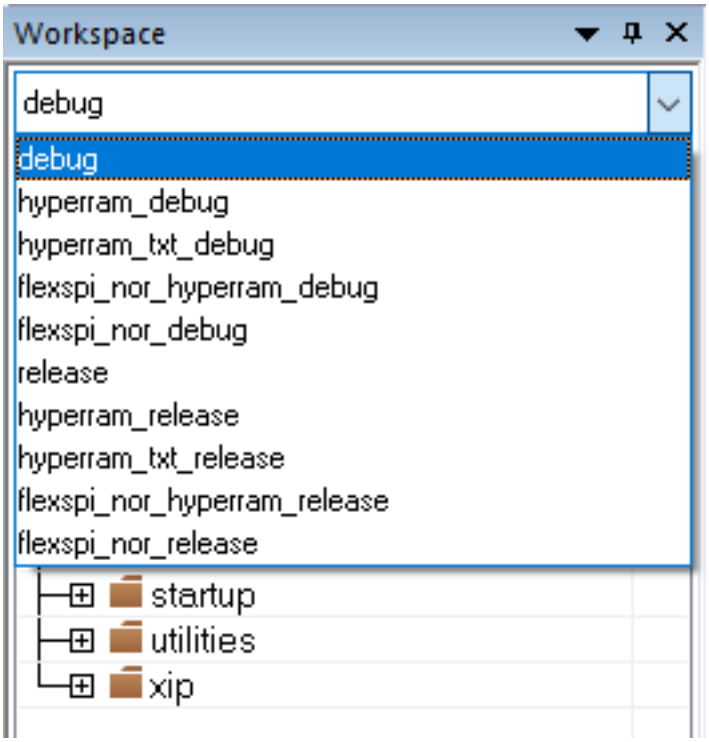
   Other example applications may have additional folders in their path.

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

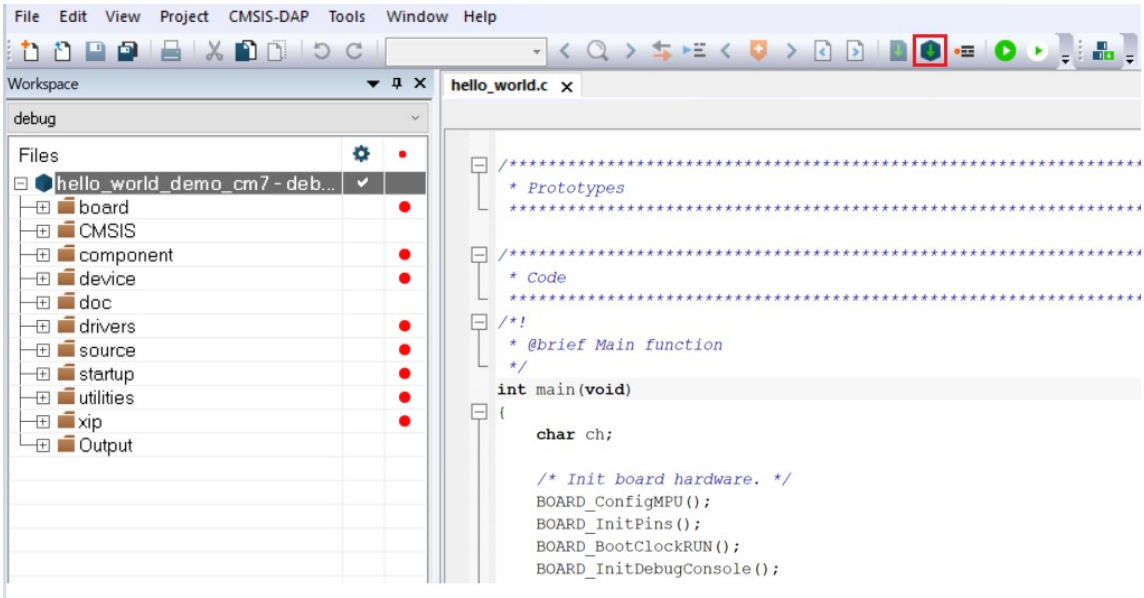
   The below lists project configurations (build targets) supported across MCUXpresso SDK projects:

   - Debug – Compiler optimization is set to low, and debug information is generated for the executable. The linker file is RAMlinker, where text and data section is put in internal TCM.
   - Release – Compiler optimization is set to high, and debug information is not generated. The linker file is RAM linker, where text and data section is put in internal TCM.
   - flexspi_nor_debug – Project configuration is same as the debug target. The linker file is flexspi_nor linker, where text is put in flash and data put in TCM.
   - flexspi_nor_release – Project configuration is same as the release target. The linker file is flexspi_nor linker, where text is put in flash and data put in TCM.
   - hyperram_debug – Project configuration is same as the debug target. The linker file is HYPERRAM linker, where text is put in internal TCM and data put in HYPERRAM.
   - hyperram_release – Project configuration is same as the release target. The linker file is HYPERRAM linker, where text is put in internal TCM and data put in HYPERRAM.
   - hyperram_txt_debug – Project configuration is same as the debug target. The linker file is HYPERRAM_txt linker, where text and data section is put in HYPERRAM.
   - hyperram_txt_release – Project configuration is same as the release target. The linker file is HYPERRAM_txt linker, where text and data section is put in HYPERRAM.

- flexspi_nor_hyperram_release - Project configuration is same as the release target. The linker file is flexspi_nor_hyperram linker, where text is put in flash and data put in HYPERRAM.
- flexspi_nor_hyperram_debug – Project configuration is same as the debug target. The linker file is flexspi_nor_hyperram linker, where text is put in flash and data put in HYPERRAM. For this example, select **hello_world_demo– debug**.



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



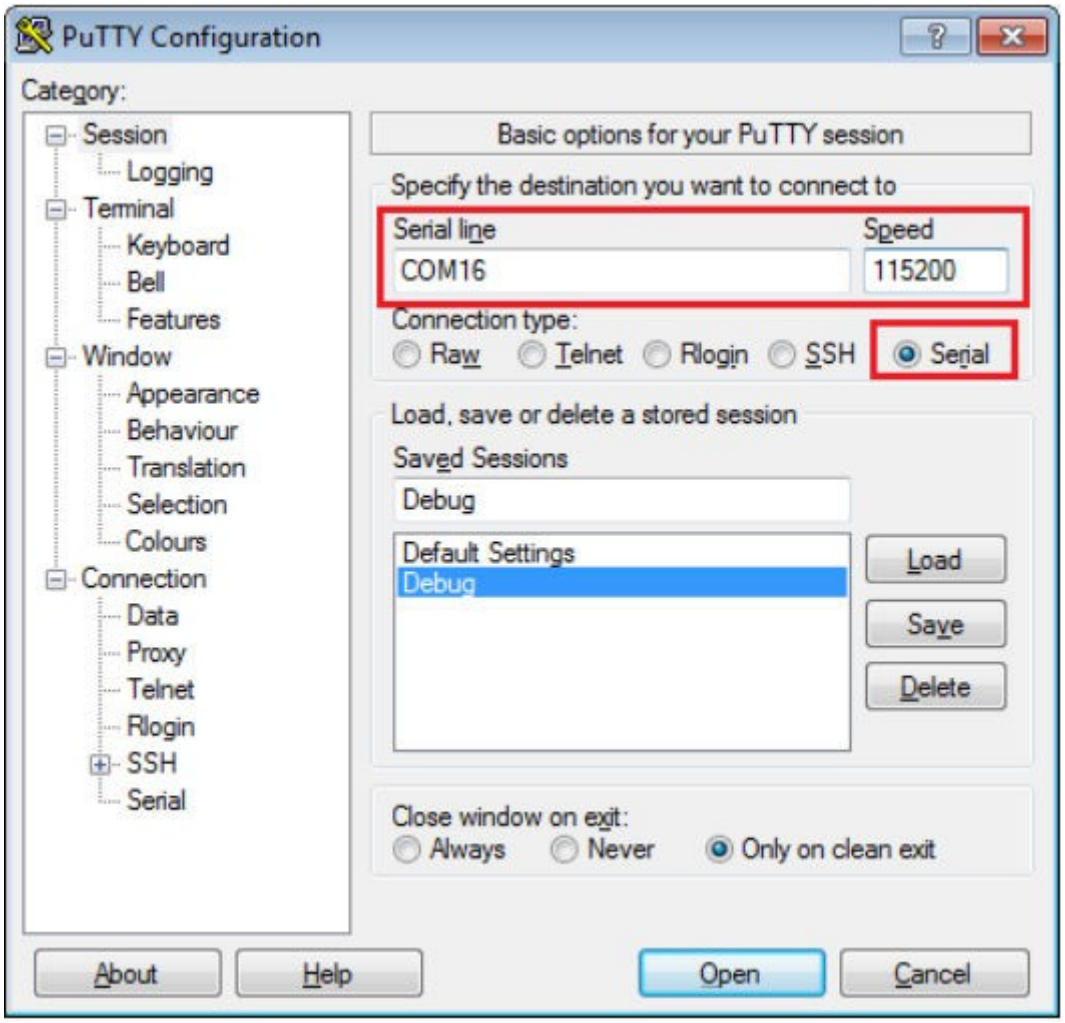
4. The build completes without errors.

Parent topic:*Run a demo using IAR*

Run an example application The default IAR project setting assumes that CMSIS-DAP debug interface is used. When such a debug interface is used, perform the following steps to do the

debugging:

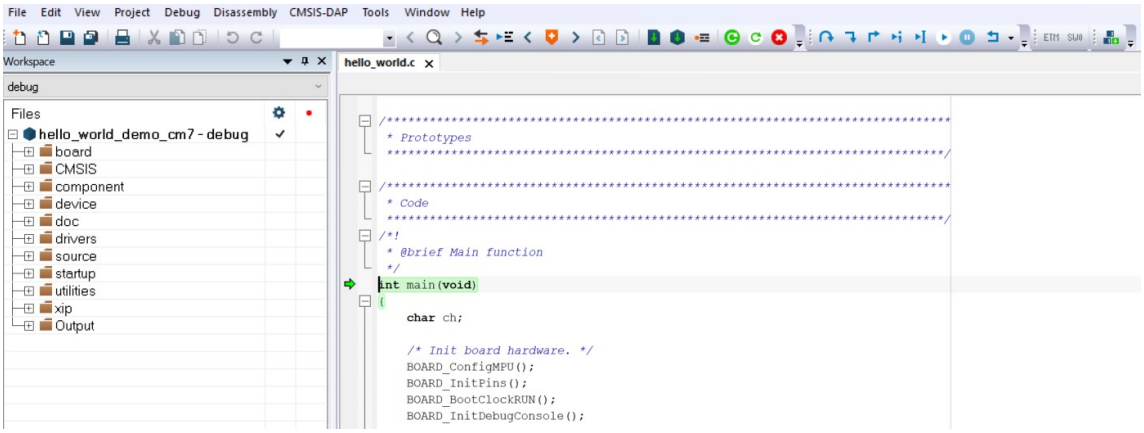
1. Connect the development platform to your PC via USB cable. Connect the USB cable to **J53** for serial output.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
    1. 115200 baud 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. In IAR, click the **Download and Debug** button to download the application to the target.



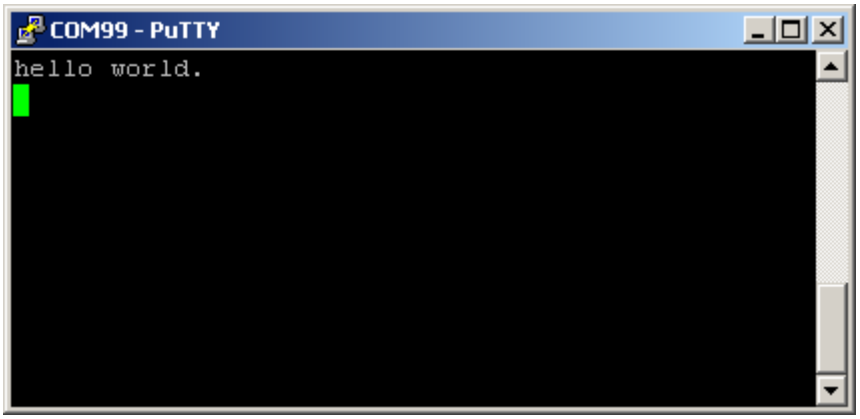
4. The application is then downloaded to the target and automatically runs to the main() function.



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



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



Parent topic:*Run a demo using IAR*

Build and run a multicore example application This section describes the steps to build and run a dual-core application. The demo applications workspace files are available in the folder: <install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/iar

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

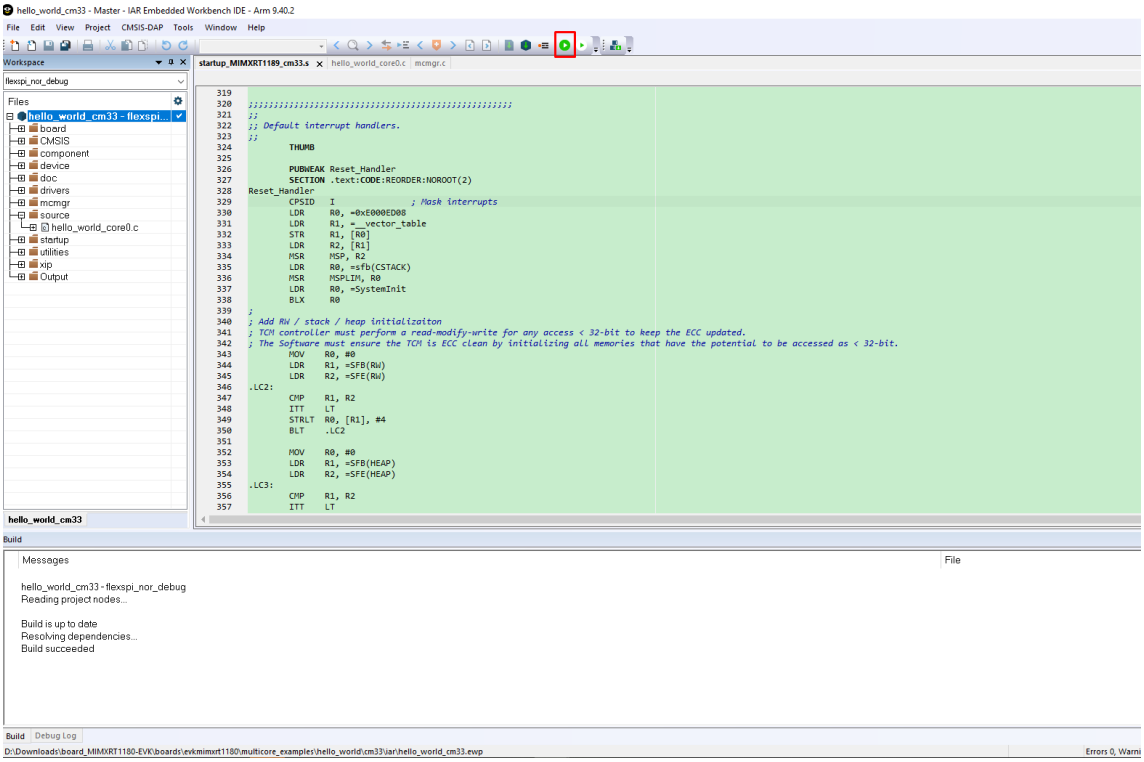
<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm7/iar/hello_world_cm7.eww

<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm33/iar/hello_world_cm33.eww

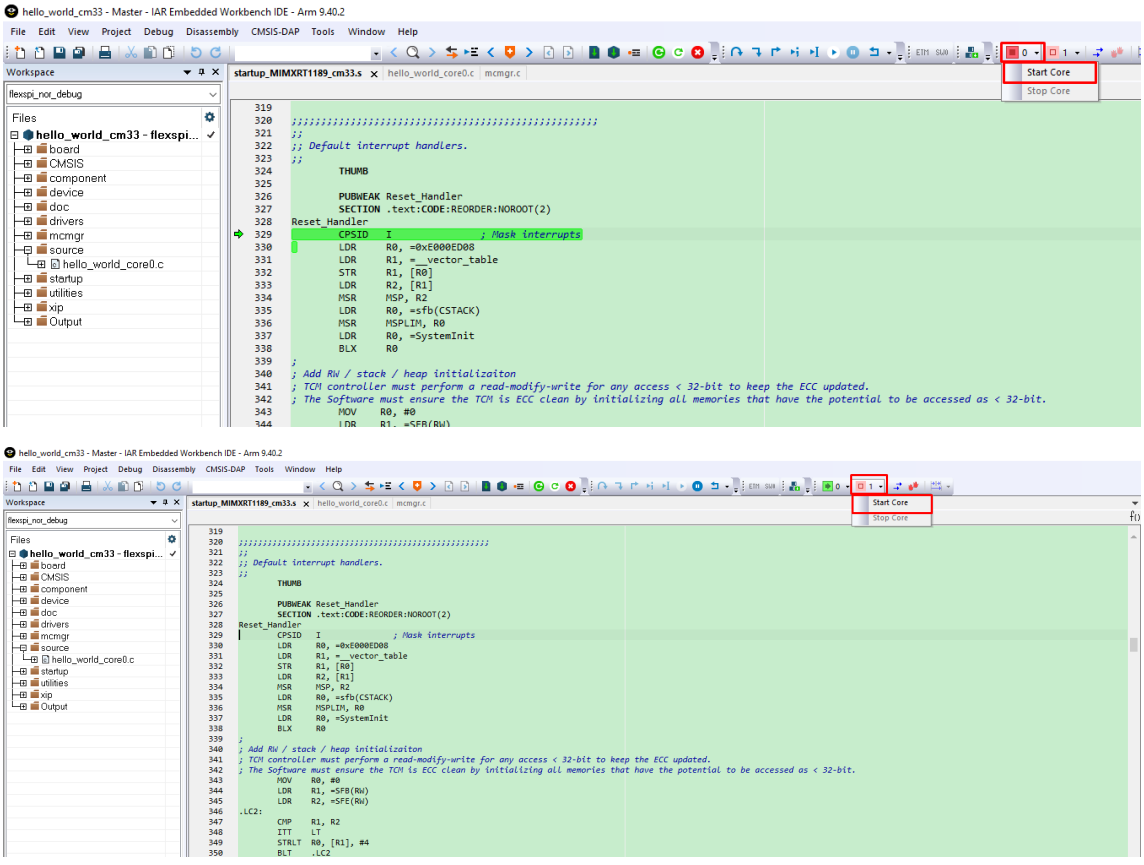
Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm7) first, because the primary core application project (cm33) must know the auxiliary core application binary when running the linker. When the auxiliary core application binary is not ready, it is impossible to finish the primary core linker.

By default, the primary core flexspi_nor_debug target links the auxiliary core debug target, and the primary core flexspi_nor_release target links the auxiliary core release target. During the primary core execution, the auxiliary core image is copied from flash into the CM7 RAM and executed.

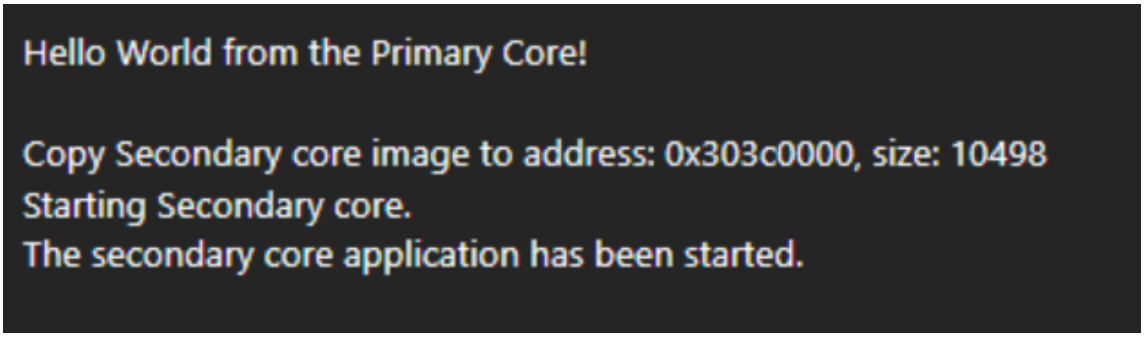
1. Build the CM7 and CM33 projects respectively.
2. Only click the Download and Debug button on the CM33 project, IAR could help start to debug a multicore project. It is user-friendly to debug multicore examples with CMSIS-DAP on IAR (Multicore Project is set on Debugger->Multicore window).



3. Start core0 and then start core1 on the CM33 project.



4. Hello_World multicore demos are now running. A banner appears on the terminal and the LED D6 blinks. If this is not true, check your terminal settings and connections.

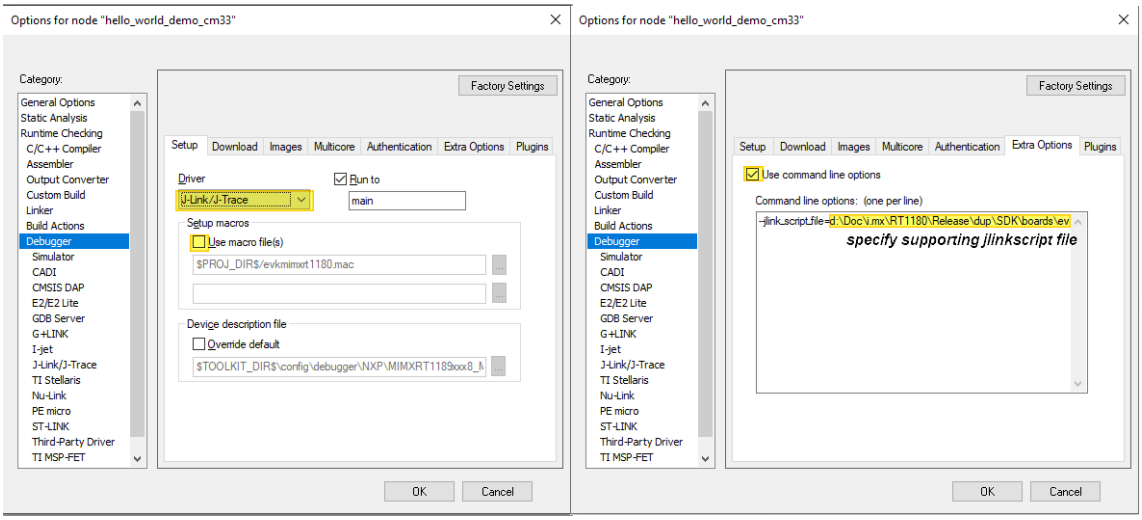


Parent topic:*Run a demo using IAR*

Run applications via JLink debug interface When JLink Debug Interface is used, manually revise **Debug Configuration** under **Project** > **Options** > **Debugging**.

1. Change the debug driver to J-Link/J-Trace.
2. Uncheck **Use macro file(s)**.
3. In the **Extra Options**, specify a jlinkscript. The jlinkscript files can be found in example folder in SDK package.

   evkmimxrt1180_cm33.jlinkscript is for CM33 projects and evkmimxrt1180_cm7.jlinkscript is for CM7 projects. Use --jlink_script_file=<full patch> to specify it.



After these settings, debugging can be performed via the J-Link debugging interface.

Note: If IAR version is higher than or equal to 9.60.1, perform just *Step 1*.

Parent topic:*Run a demo 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.

Both CMSIS-DAP and J-Link debugging interfaces are supported for Keil IDE. It is recommended to reset board for each download/debug.

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 MIMXRT118x CMSIS pack.

1. Download the MIMXRT1181, MIMXRT1182, MIMXRT1187, and MIMXRT1189 packs.
2. After downloading the DFP, double click to install it.

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

Build an example application

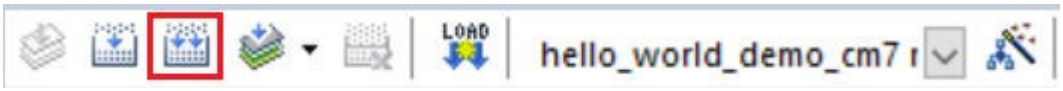
1. Open the desired example application workspace in:

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

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

   <install_dir>/boards/evkmimxrt1180/demo_apps/hello_world/cm33/mdk/hello_world_demo_cm33.uvmpw

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

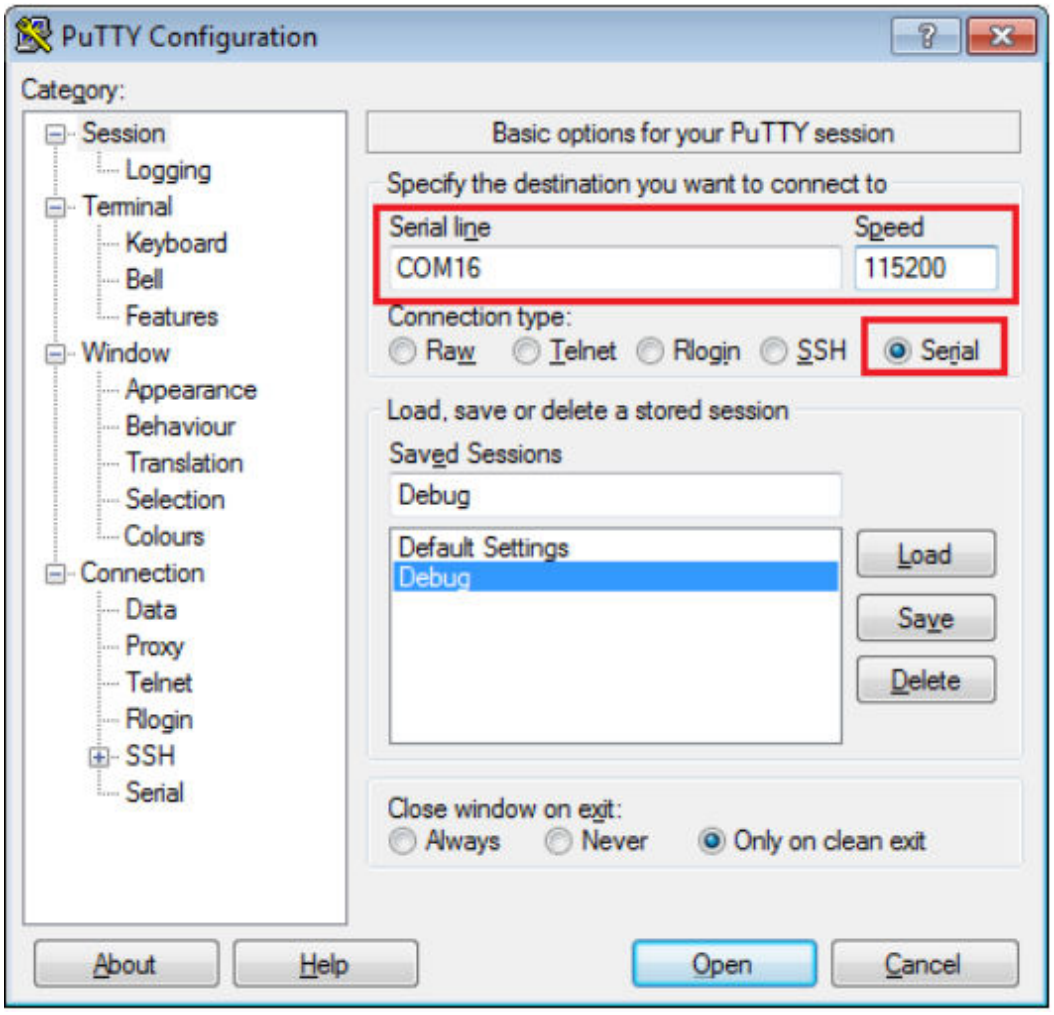


3. The build completes without errors.

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

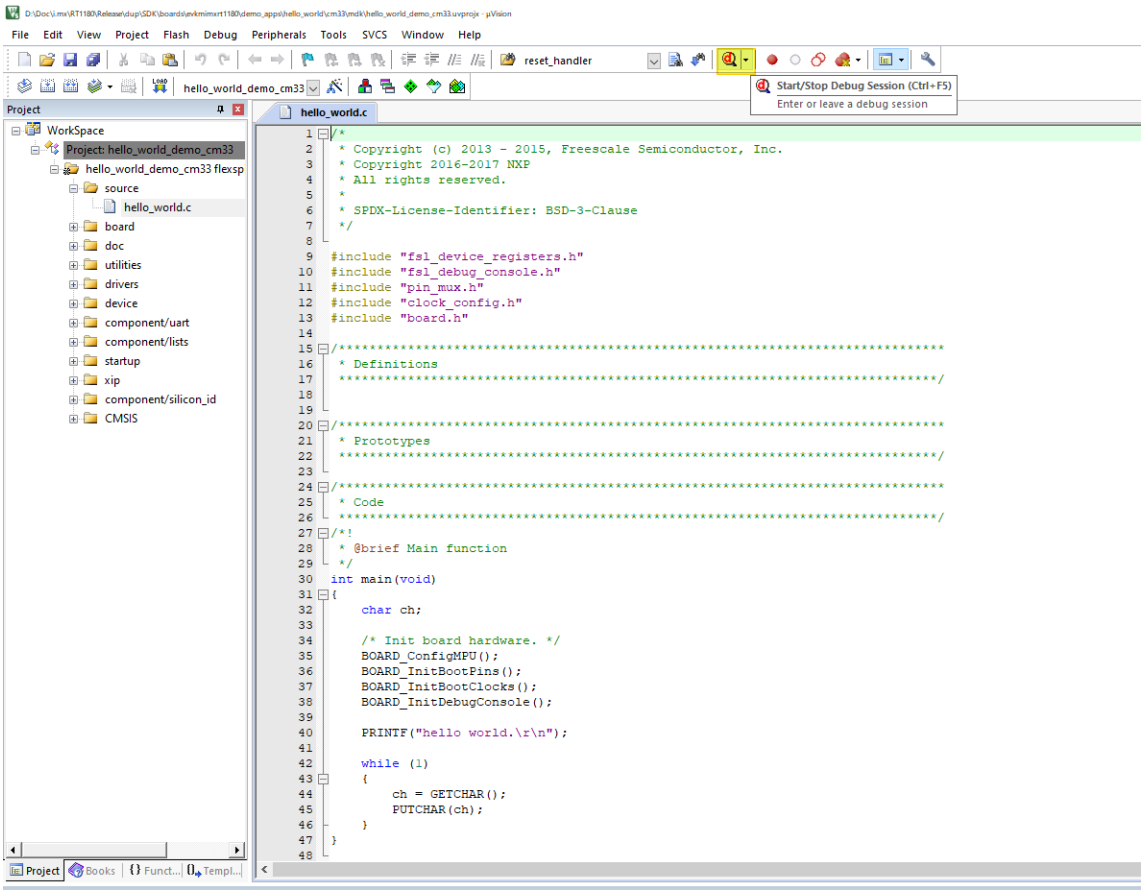
Run an example application The default MDK project settings assumes CMSIS-DAP debug interface is used. If you are using a CMSIS-DAP debugging interface, perform the following steps:

1. Connect the development platform to your PC via USB cable.
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_BAUDRATEvariable in the *board.h*file)
   2. No parity
   3. 8 data bits

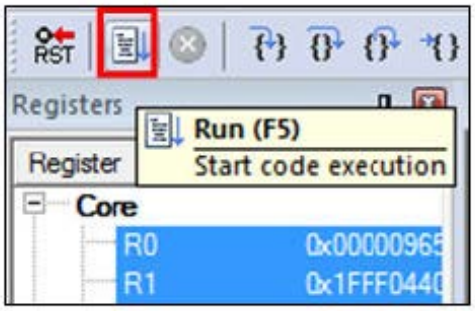


4. 1 stop bit

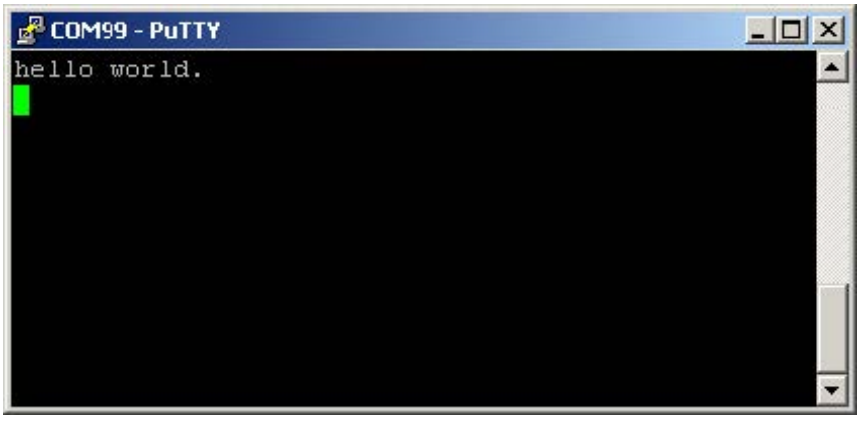
3. To debug the application, click the **Start/Stop Debug Session** button or **Ctrl** + **F5**, highlighted in *Figure 2*.



4. Run the code by clicking **Run** to start the application, as shown in *Figure 3*.



The hello_world application is now running and a banner is displayed on the terminal, as shown in *Figure 4*. If this is not true, 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 dual-core application. The demo applications workspace files are located in this folder:

<install_dir>/boards/evkmimxrt1180/multicore_examples/<application_name>/<core_type>/mdk

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

<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm7/mdk/hello_world_cm7.uvmpw

<install_dir>/boards//evkmimxrt1180/multicore_examples/hello_world/cm33/mdk/hello_world_cm33.uvmpw

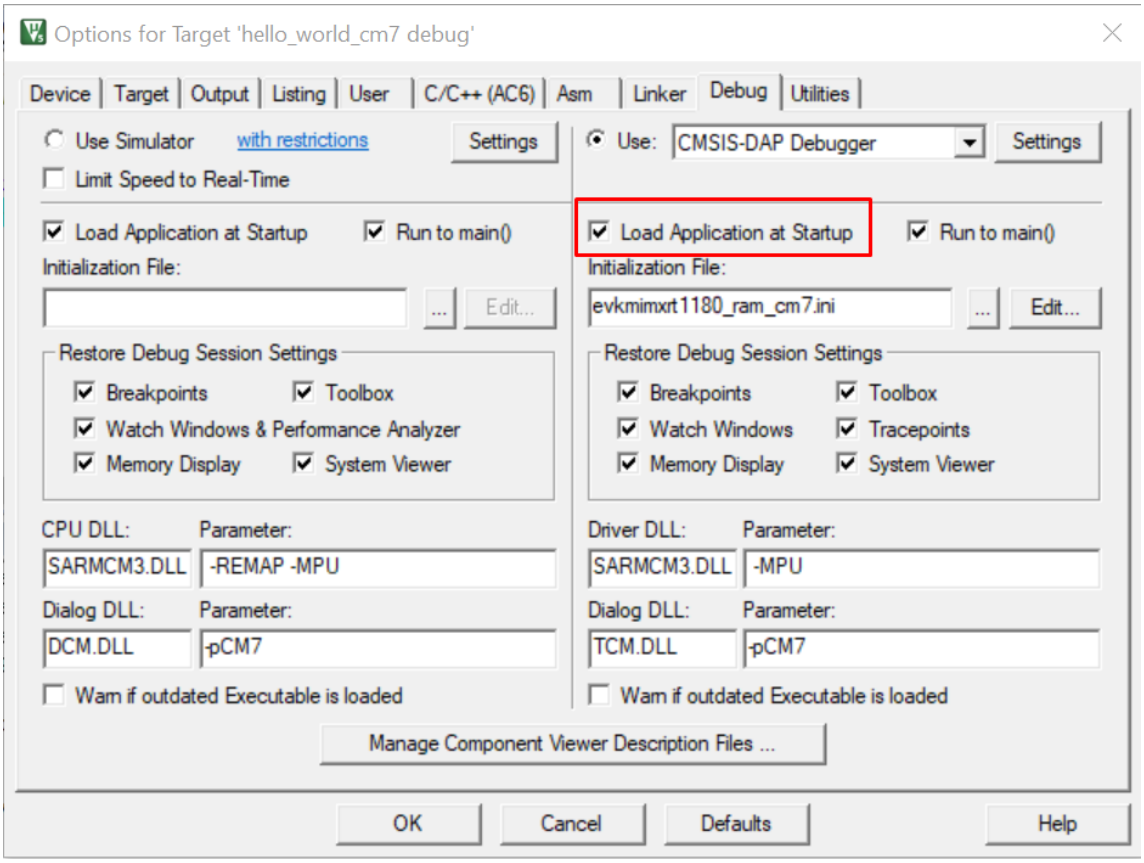
Build both applications separately by clicking the **Rebuild** button. Build the application for the auxiliary core (cm7) first, because the primary core application project (cm33) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

Because the auxiliary core runs always from RAM, debug and release RAM targets are present in the project only. When building the primary core project, it is possible to select flexspi_nor_debug/flexspi_nor_release Flash targets. When choosing Flash targets the auxiliary core binary is linked with the primary core image and stored in the external SPI Flash memory. During the primary core execution the auxiliary core image is copied from flash into the CM7 RAM and executed.

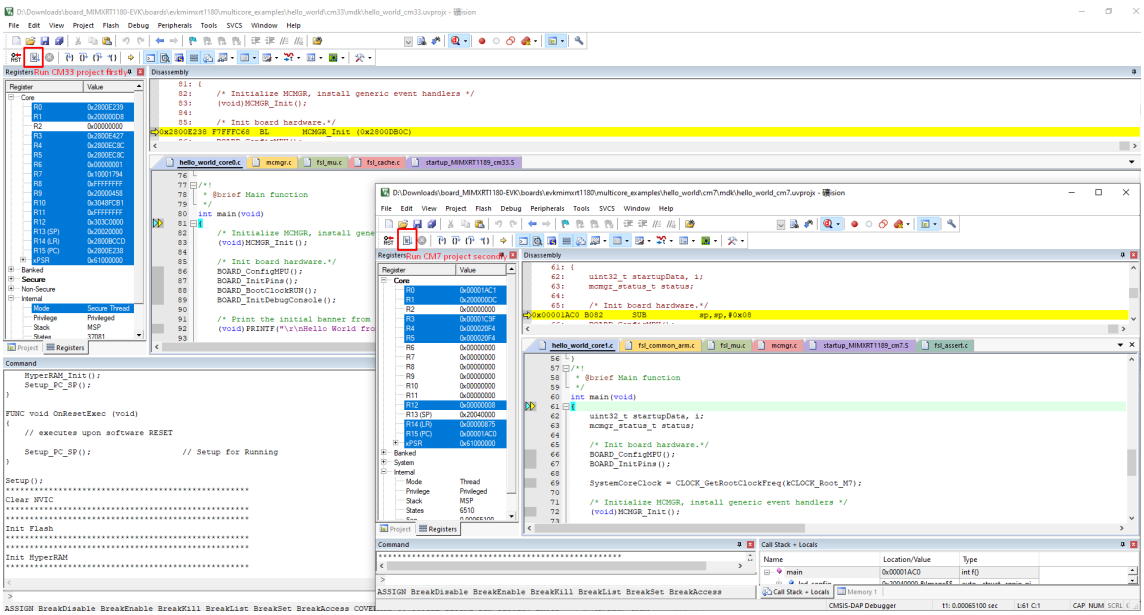
Parent topic:*Run a demo using Keil MDK/µVision*

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.

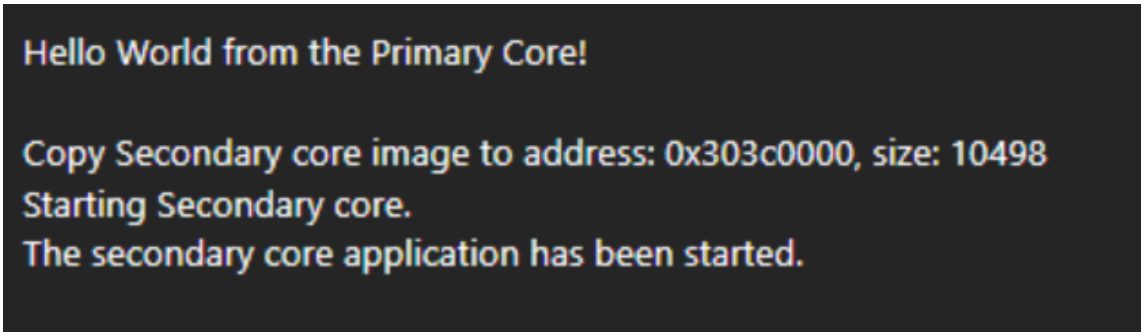
1. To download and run the multicore application, switch to the primary core application project and perform Step 1 – Step 3 as described in Build a multicore example application.
2. For the secondary core project, select the Load Application at Startup button as shown in *Figure 1*. These steps are common for both single-core and dual-core applications in µVision.



3. Run the primary core project and then run the secondary core project.



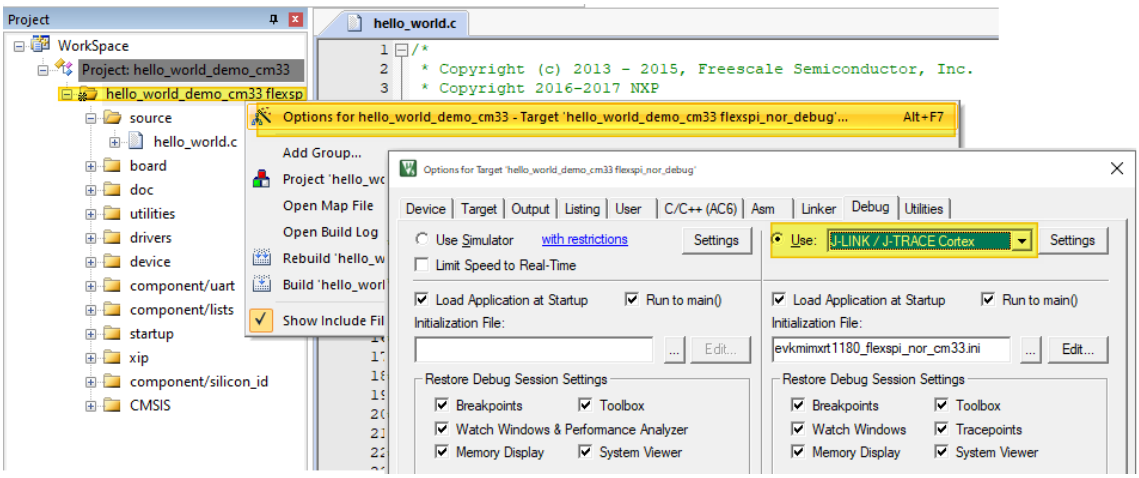
4. Hello_World multicore demos are now running. A banner appears on the terminal and the LED D6 blinks. If this is not true, check your terminal settings and connections.



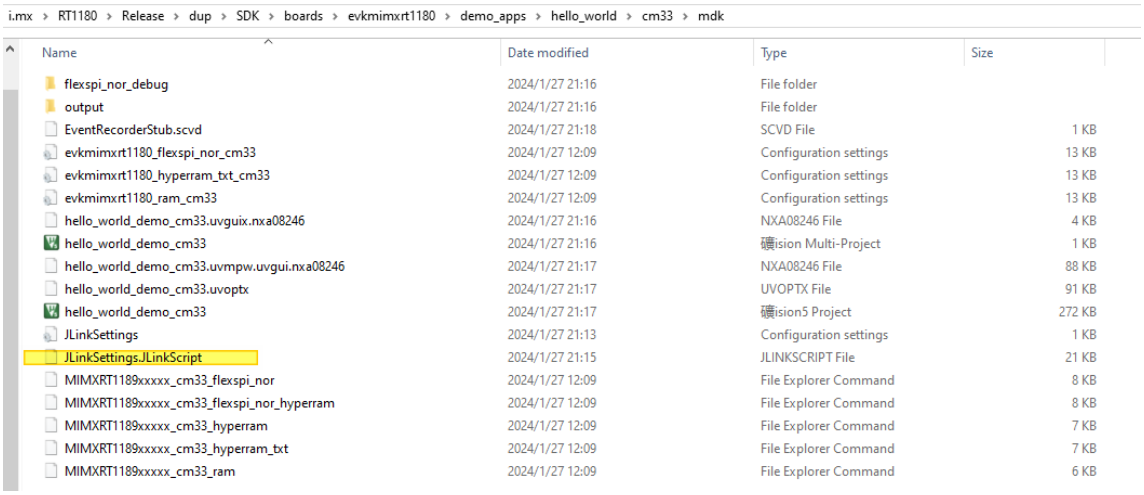
Parent topic:*Run a demo using Keil MDK/µVision*

Run applications via JLink debug interface When JLink Debug Interface is used, manually revise **Debug Configuration**.

1. Under **Project** > **Options** > **Debug**, change the debug interface to J-LINK / J-TRACE Cortex.



2. Copy the supporting JLinkscript file from example folder to project root mdk. For CM33 projects, use evkmimxrt1180_cm33.jlinkscript and for CM7 projects, use evkmimxrt1180_cm7.jlinkscript. Rename them to JLinkSettings.JLinkScript.



After these settings, debugging can be performed via the J-Link debugging interface.

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_worlddemo application is targeted which is used as an example.

Only J-Link debugging interface is supported for JLinkGDBServer and GDB.

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.

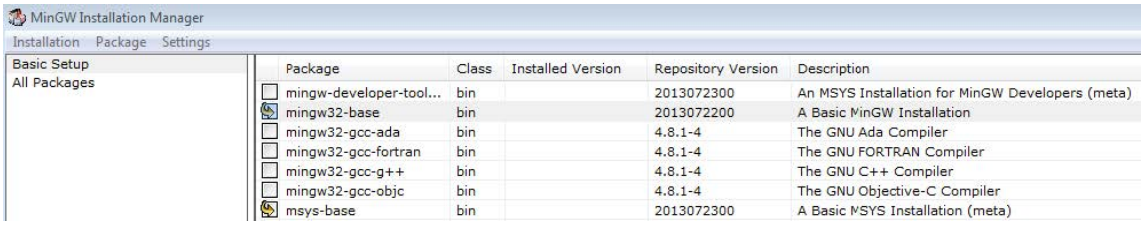
Install GCC Arm Embedded tool chain Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, etc.). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes for MIMXRT1180-EVK* (document MCUXSDKMIMXRT118XKRN).

Parent topic: Set up toolchain

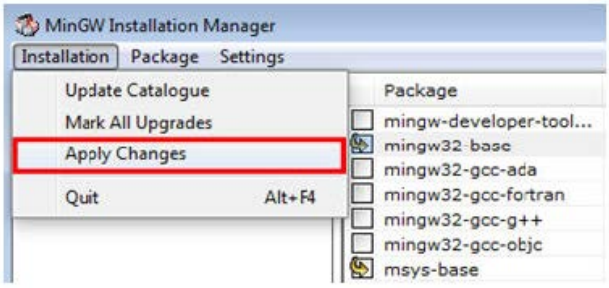
Install MinGW (only required on Windows OS) The Minimalist GNU for Windows (MinGW) development tools provide a set of tools that are not dependent on third-party C-Runtime DLLs (such as Cygwin). The build environment used by the MCUXpresso SDK does not use the MinGW build tools, but does leverage the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

1. Download the latest MinGW mingw-get-setup installer from SOURCEFORGE.
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.

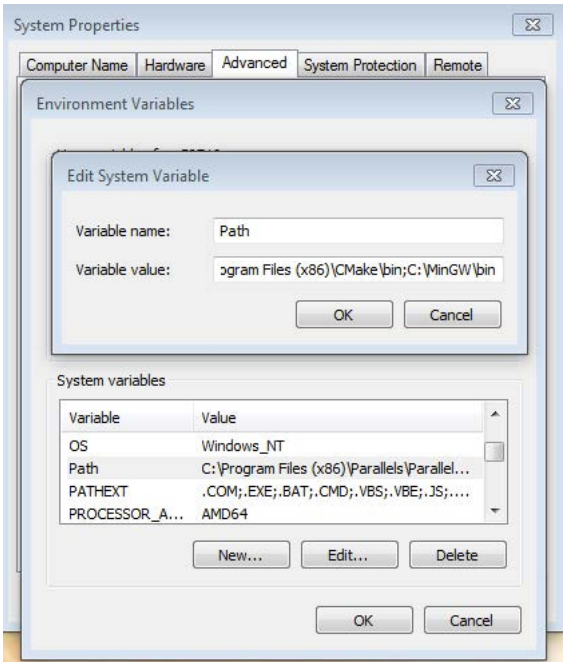


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel**->**System and Security**->**System**->**Advanced System Settings**in the **Environment Variables...**section. The path is:

<mingw_install_dir>\bin

Assuming the default installation path is *C:\MinGW*, an example is as shown in *Figure 3*. 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 system environment variable for ARMGCC_DIR Create a system environment variable and name it as ARMGCC_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

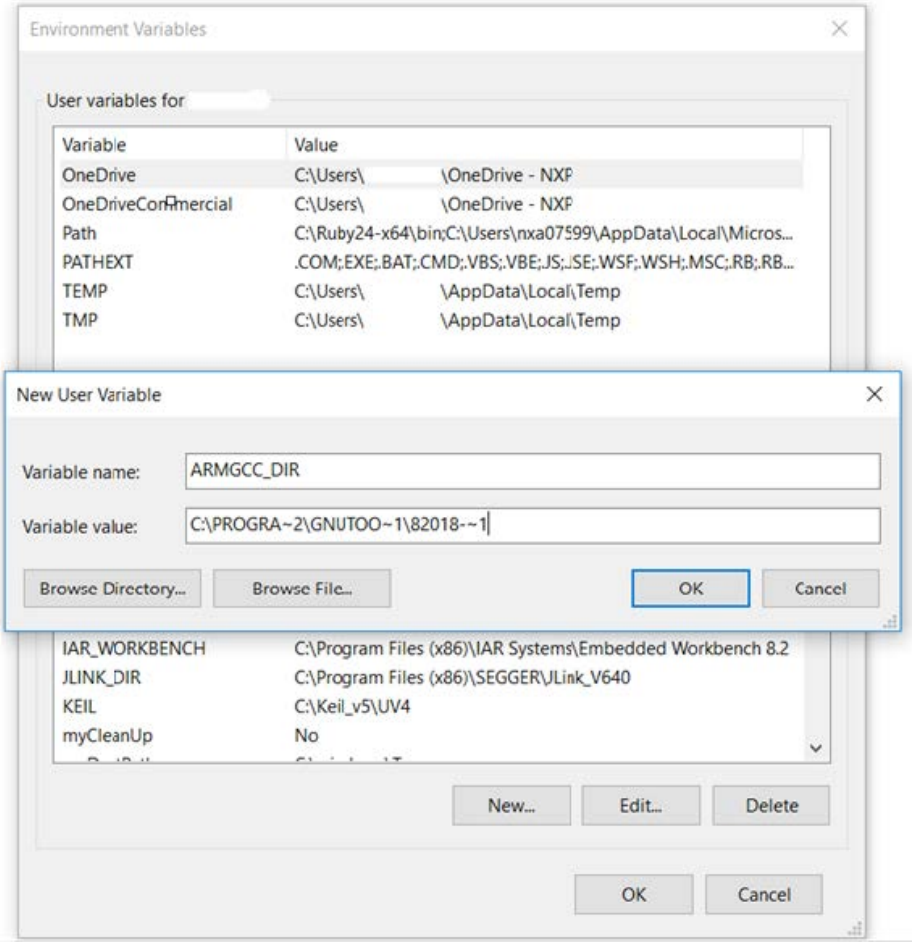
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major

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

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

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

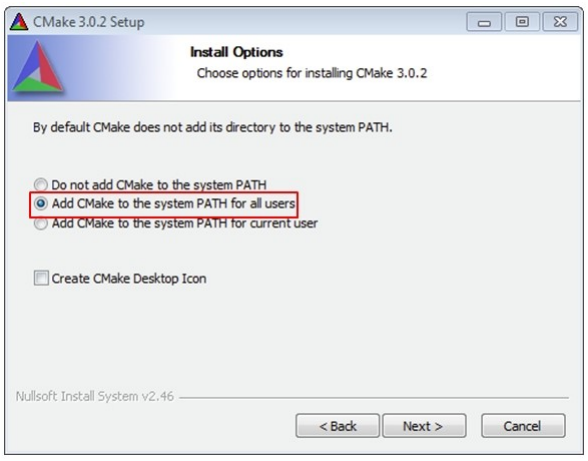
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018-~1
C:\PROGRA~2\GNUTOO~1\82018-~1
```



Parent topic: Set up toolchain

Install CMake

1. Download CMake 3.0.x from CMAKE.
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.

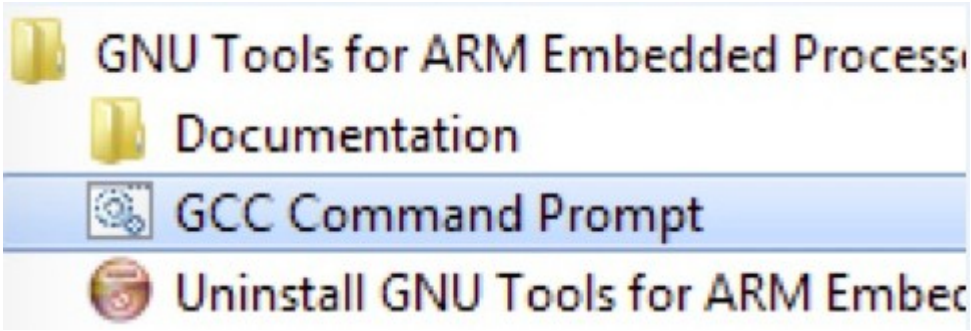
5. Make sure sh.exe is not in the Environment Variable PATH. This is a limitation of mingw32-make.

Parent topic: Set up toolchain

Parent topic:*Run a demo using Arm GCC*

Build an example application To build an example application, perform the following 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>/<core_type>/<application_name>/armgcc

   For this example, the exact path is:

   <install_dir>/examples/evkmimxrt1180/demo_apps/hello_world/cm33/armgcc

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*.

```
[ 95%] Building C object CMakeFiles/hello_world_demo_cm33.elf.dir/C_/repo1/mcu-sdk-2.0/platfo
rm/utilities/misc_utilities/fsl_sbrk.c.obj
[100%] Linking C executable debug\hello_world_demo_cm33.elf
Memory region         Used Size  Region Size  %age Used
    m_interrupts:          1 KB         1 KB    100.00%
          m_text:       20432 B       127 KB     15.71%
          m_data:        1232 B       127 KB      0.95%
        m_ncache:          0 GB       256 KB      0.00%
         m_data2:          0 GB       256 KB      0.00%
    rpmsg_sh_mem:          0 GB         0 GB
          m_heap:          1 KB         1 KB    100.00%
[100%] Built target hello_world_demo_cm33.elf
```

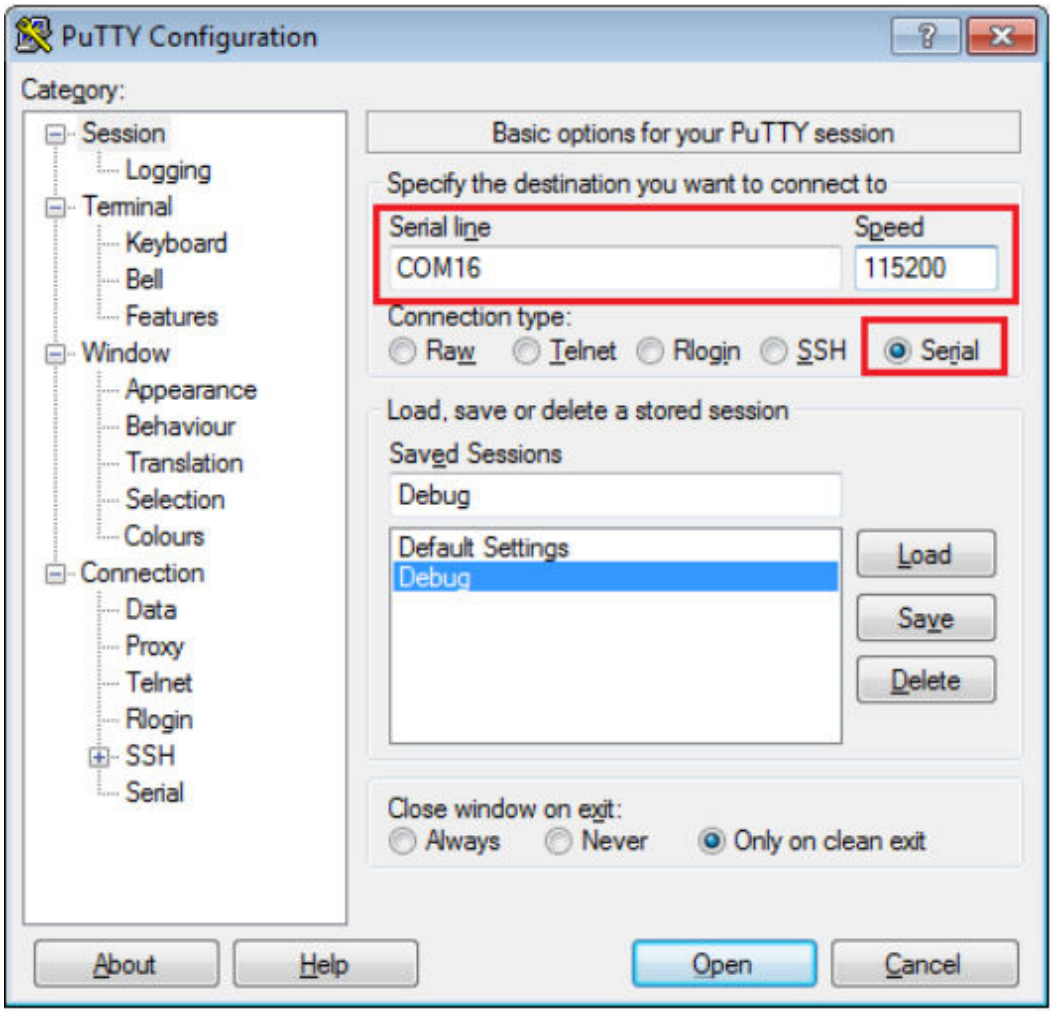
Parent topic:*Run a demo using Arm GCC*

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 a standalone J-Link pod is connected to the debug interface of your board.

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

1. This board supports the J-Link debug probe. Before using it, install SEGGER software, which can be downloaded from SEGGER.
2. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector.

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



4. Open the J-Link GDB Server application. Go to the SEGGER install folder. For example, *C:\Program Files(x86)\SEGGER\JLink_Vxxx*. Open the command windows. To debug CM33, run the following command:

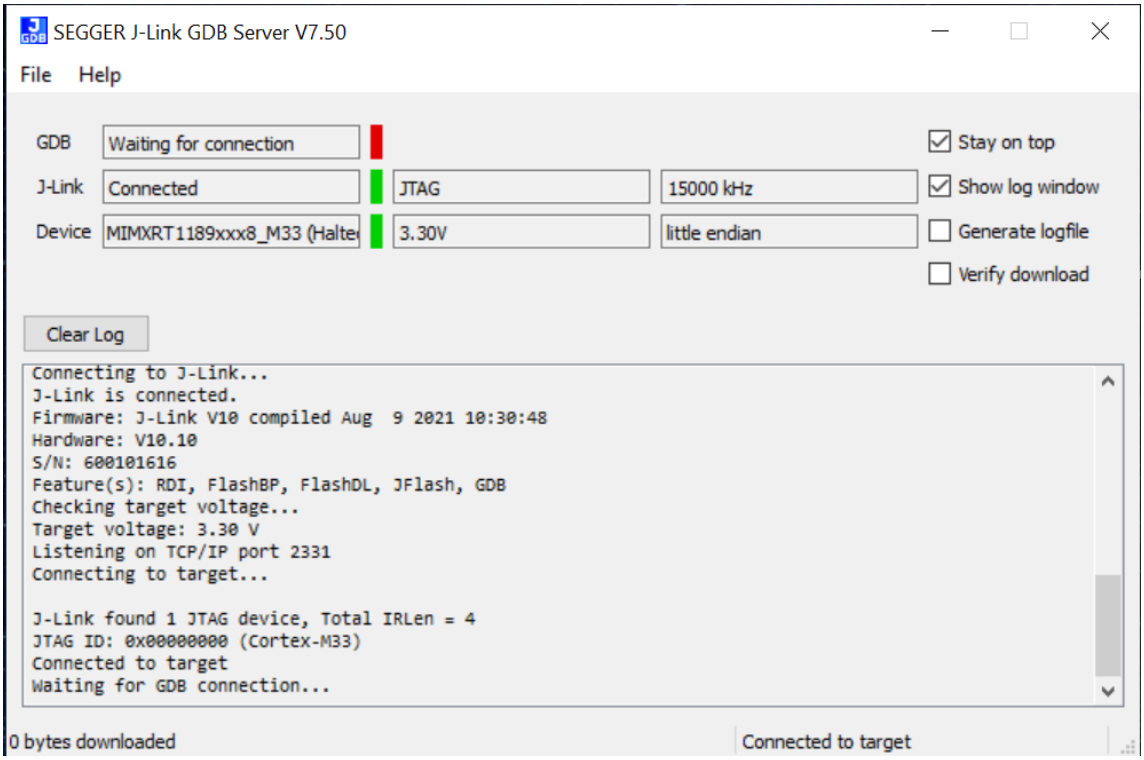
```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M33 -if SWD -speed 4000 -jlinkscriptfile
↪<evkmimxrt1180_cm33.jlinkscript> -stayontop -ir
```

To debug CM7, run the following command:

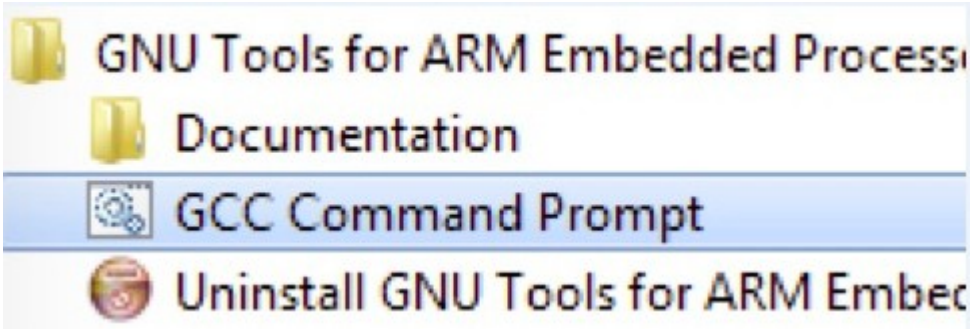
```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M7 -if SWD -speed 4000 -jlinkscriptfile
↪<evkmimxrt1180_cm7.jlinkscript> -stayontop -ir
```

Note: The supporting jlinkscript file can be found in *boards/evkmimxrt1180/jlinkscript*.

5. After it is connected, the screen should resemble *Figure 2*.



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



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

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/<core>/armgcc/<target>

   For this example, the path is:

   <install_dir>/boards/evkmimxrt1180/demo_apps/hello_world/<core>/armgcc/debug

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

```
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin>arm-none-eabi-gdb.exe C:\repo1\
mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug\hello_world_demo_cm33.elf
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin\arm-none-eabi-gdb.exe: warning:
 Couldn't determine a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 9-2020-q2-update) 8.3.1.20191211-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

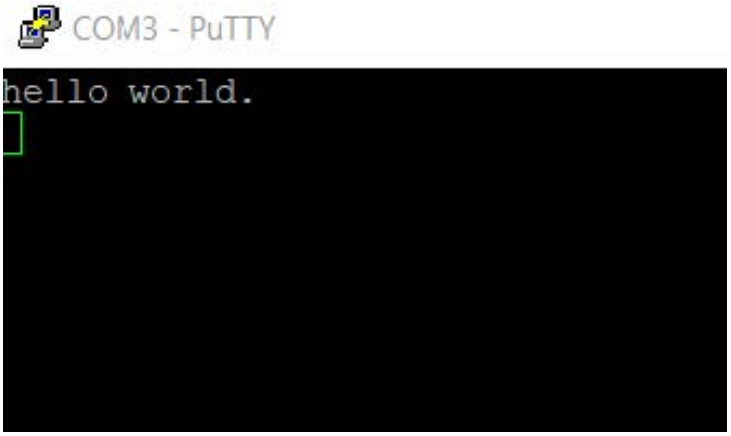
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from C:\repo1\mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug
\hello_world_demo_cm33.elf...
(gdb)
```

9. Run these commands:

```
target remote localhost:2331
monitor reset
monitor halt
load
```

10. The application is now downloaded and halted at the reset vector. Execute the `continue` or `monitor go` command to start the demo application.

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



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 dual-core application. The demo application build scripts locate in this folder:

<install_dir>/boards/evkmimxrt1180/multicore_examples/<application_name>/<core_type>/armgcc

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

<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm7/armgcc/build_debug.bat

<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm33/armgcc/build_flexspi_nor_debug.bat

Build the application for the auxiliary core (cm7) first, because the primary core application project (cm33) 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 **flexspi_nor_debug** target links the auxiliary core **debug** target, and the primary core **flexspi_nor_release** target links the auxiliary core **release** target. During the primary core execution, the auxiliary core image is copied from flash into CM7 RAM and executed.

Parent topic:*Run a demo using Arm GCC*

Bootable image generation

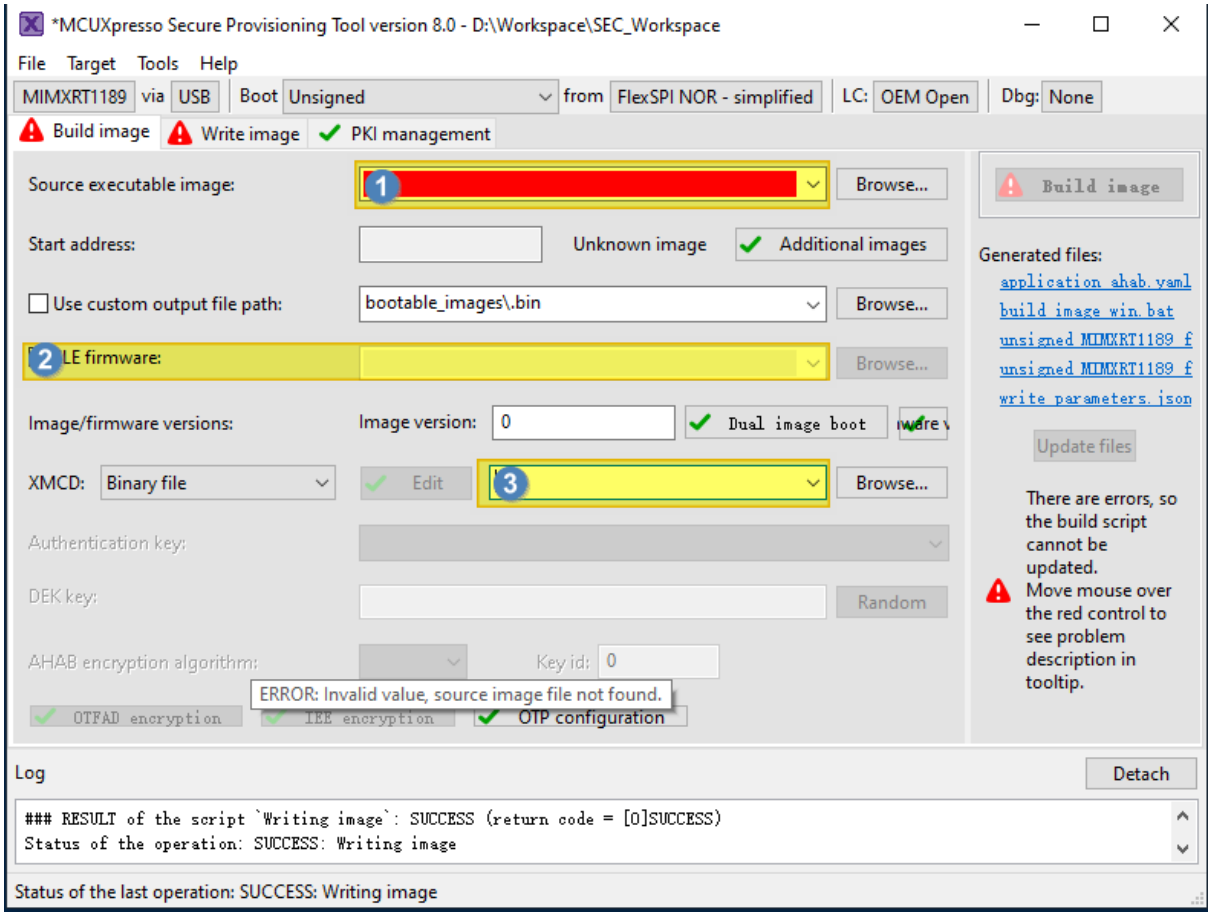
RT1180 can only boot from CM33. For CM33 application image to boot, a sophisticated boot header structure is needed. To ease the customer from complex settings, the **Secure Provisioning Tool** is provided. Among other rich features, it provides an easy way to use GUI for customer to generate bootable image from raw application image. In addition, in order for users to run the CM7 image conveniently, SDK provides the multicore_trigger demo for users to kick off CM7 image, combined with the SPT tool, we can run CM7 image by from POR.

In this section, two kinds of images are defined

Terminology	Description
RAW image	Demo image without boot header
POR image	Demo image with boot header

Most examples that we provided in SDK package create RAW images from project settings. Only some OOBE examples can create POR image. POR image can provide good out-of-box experience, but the cost is losing some flexibility. While RAW images, with the usage of Secure Provisioning Tool, have the most widely usage like image signning, multiple image combination, and so on. For demos that by default generate a POR boot image, see RAW/POR image switch.

Use SPT tool to boot cm33 image Run the Secure Provisioning Tool, and the following GUI shows.



Select the following items:

1. Toolchain generated images

   All SDK generated images are supported no matter they boot from TCM, FlexSPI Nor, or HyperRAM.

Toolchain	Suffix
ARMGCC	.elf
IAR	.out
MDK	.out/.hex
MCUX	.axf

   Note:

   - Use the RAW image for CM33. Some SDK project targets are POR image by default, it must to be switched to RAW image for SPT usage. For more information, see RAW/POR image switch.
   - It is recommended to use elf/out files, thus SPT can parse the **Start address** automatically.

2. XMCD file

   This is on-board SDRAM/HyperRAM initialization file. For the RT1180 EVK board, select *boards/evkmimxrt1180/xmcd/hyperram_xmc_auto_detect.bin*.

   Note:

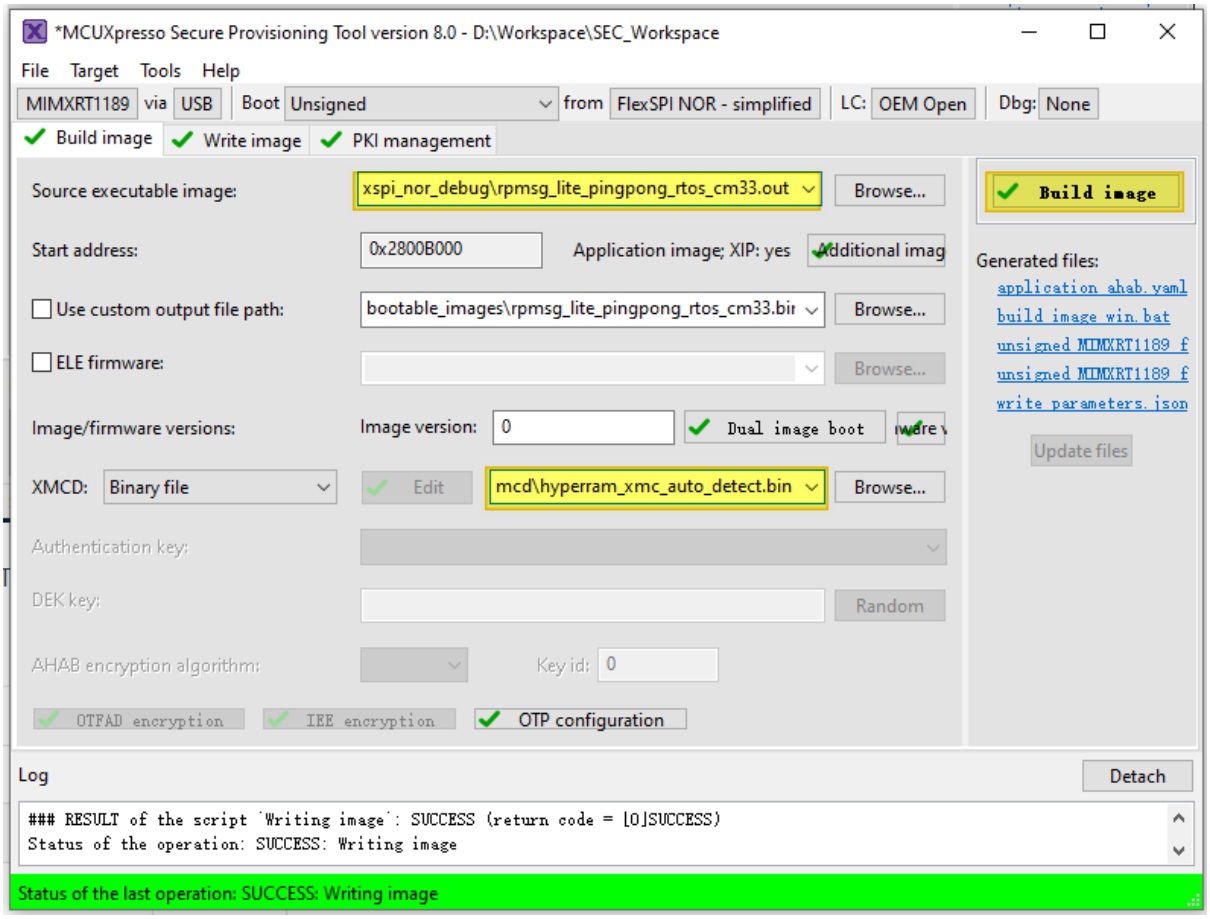
   - If the application image uses hyperram (target *hyperram_debug/release*, *flexspi_hyperram_debug/release*, *hyperram_txt_debug/release*), the XMCD is necessary.

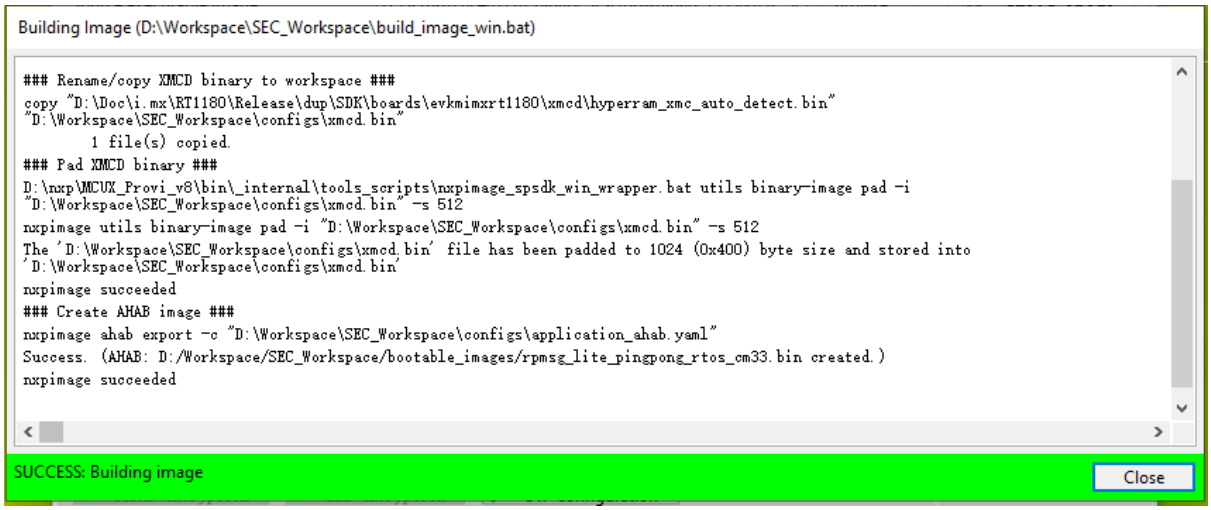
- *hyperram_xmc_auto_detect.bin* is board specific.

3. (Optional) ELE firmware

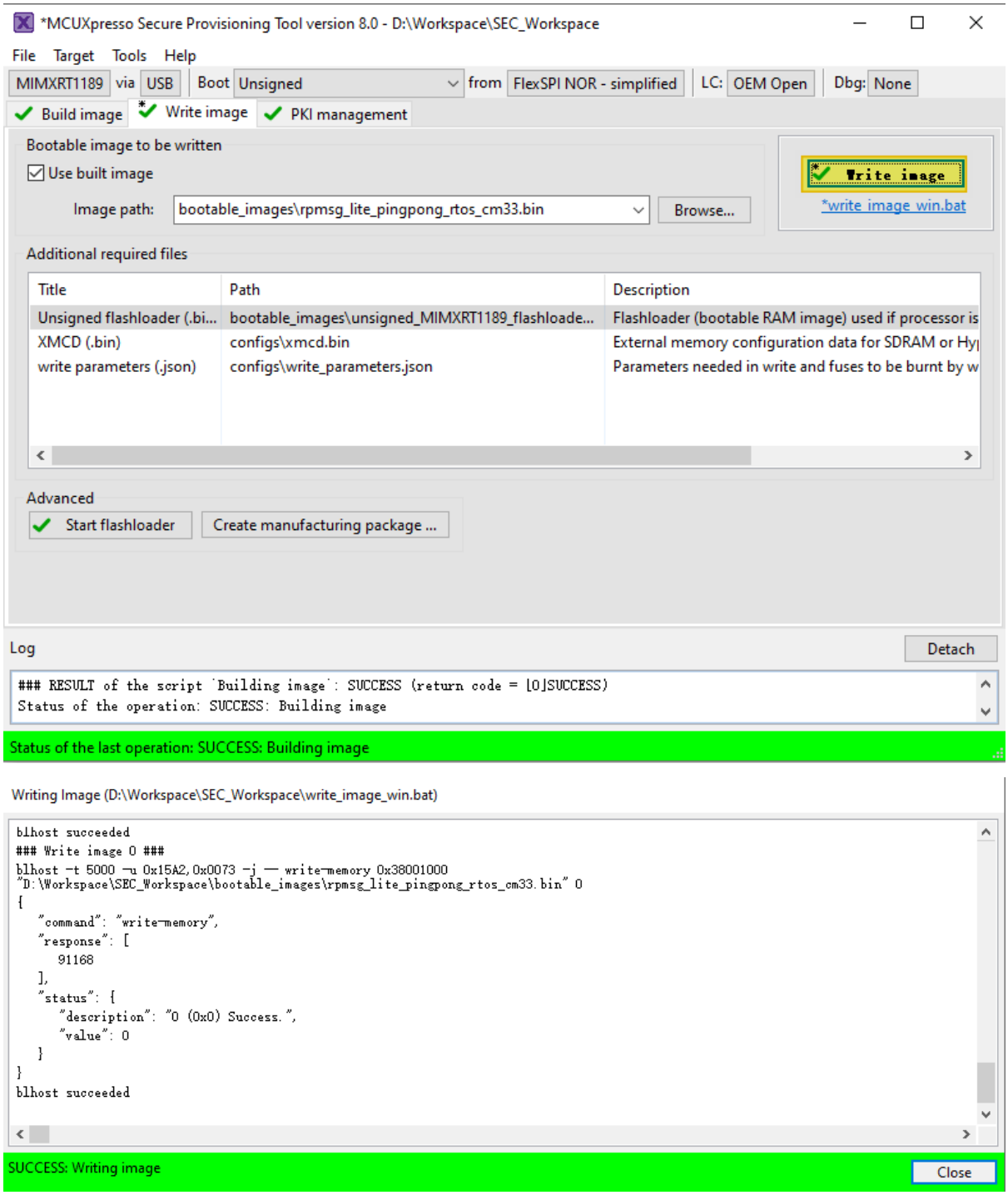
   This is needed for certain application images which need special ELE FW service. For most SDK demos, simply leave it empty.

After the selection, click Build Image to generate the bootable image (POR image).





Now switch to the **Write Image** tab. All information is automatically generated from the previous page. Connect both USB OTG1 port (J33) and MCU-Link port (J53) to the computer and power up the board (or reset via SW3). Make sure that the EVK board boot configuration switch SW5[1..4] are on 0001 SDP mode and click **Write Image**.

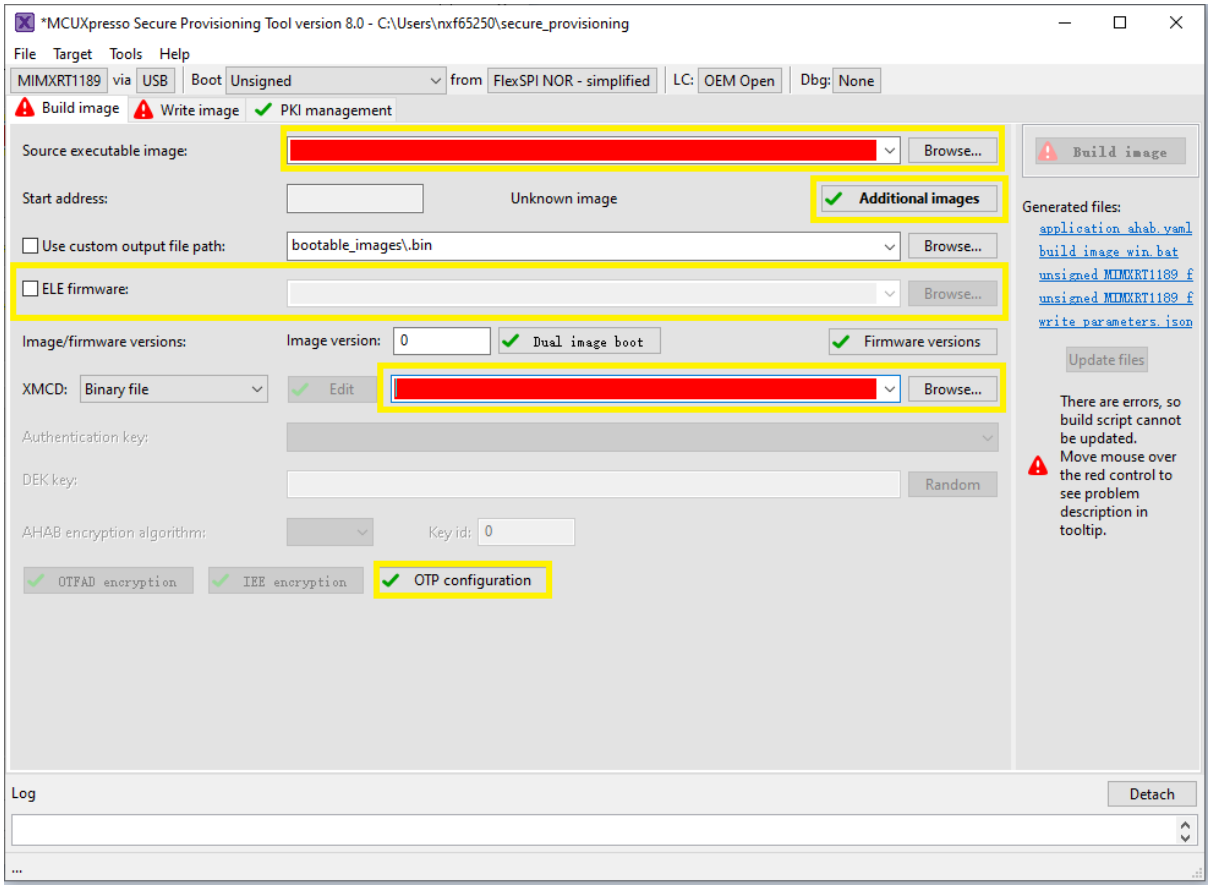


Change the SW5[1..4] to QSPI boot mode 0100 and reset the board, and then you can observe that the image is running from POR boot.

Parent topic:*Bootable image generation*

Use SPT tool and multicore_trigger image to kick off cm7 binary image This section introduces how to create a CM7 POR boot image with the aid of a CM33 kicker application. The CM33 kicker application is located in demo_apps/multicore_trigger.

For a secure provisioning tool to generate such a bootable CM7 image, specify the CM33 multicore_trigger image in Source executable image. The CM7 application image to be kicked off must be specified with additional images.



CM7 XIP image runs from external FLASH, and multicore_trigger CM33 image runs from internal RAM Perform the following steps:

1. Prepare the multicore_trigger_demo_cm33 ram image (debug or release targets) and specify it in Source executable image.
2. Specify the XMCD file if necessary, refer to Use SPT tool to boot cm33 image.
3. Use Additional images to specify CM7 application image running from flash memory. See *Table 1* for how to fill in the necessary information for this additional image.

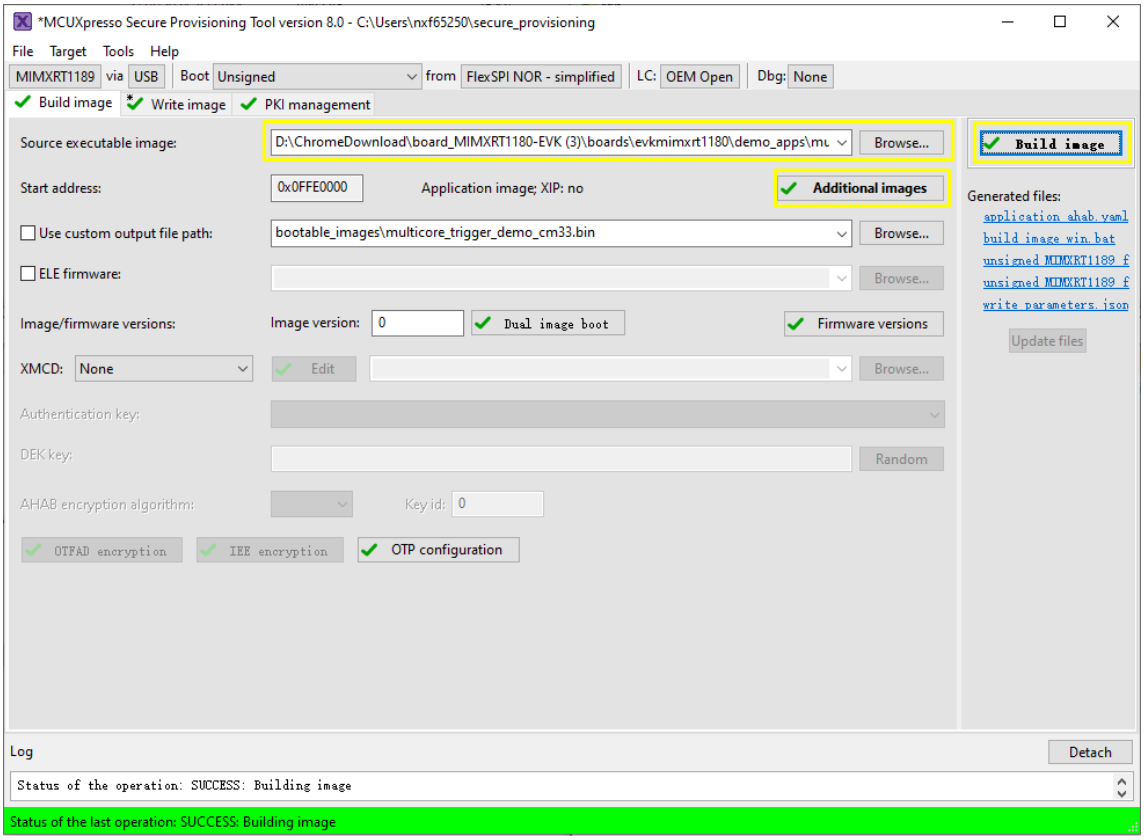
Parameter	Value	Explanation
Image offset	0x007	The offset in bytes from the beginning of the current container header to beginning of the image. In our case, the CM7 image is flashed to 0x2880_0000. The container header is put to 0x2800_1000 and the yield offset = 0x7F_F000
Load address	0x288	The destintion address of the CM7 image. ROM copies the image to load address, if it is not within flash address space. In our case, copy does not happen
Entry point	0x288	The start address of CM7 image vector table from the CM7 core address space
Core ID	cortex m7	Specify the core ID
Image type	Executab	Specify the image type
Encrypt	No	Specify the encryption



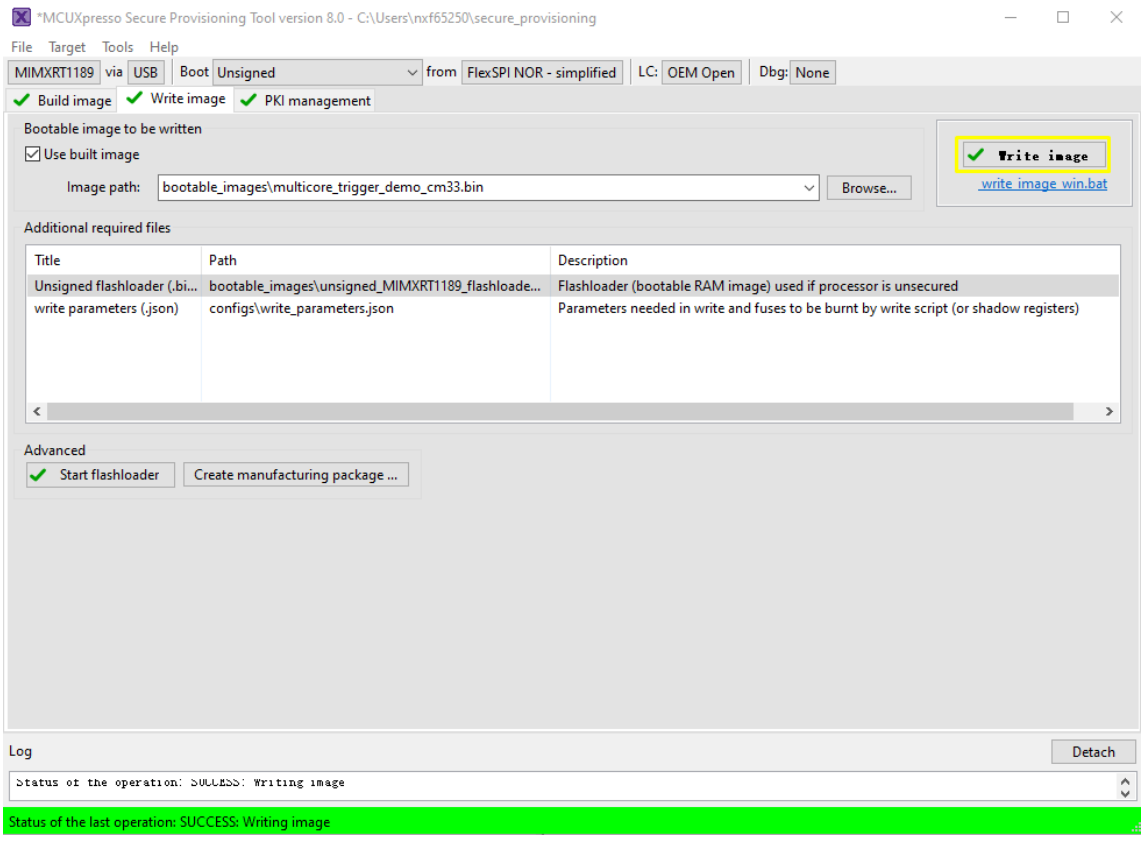
4. (Optional) ELE firmware.

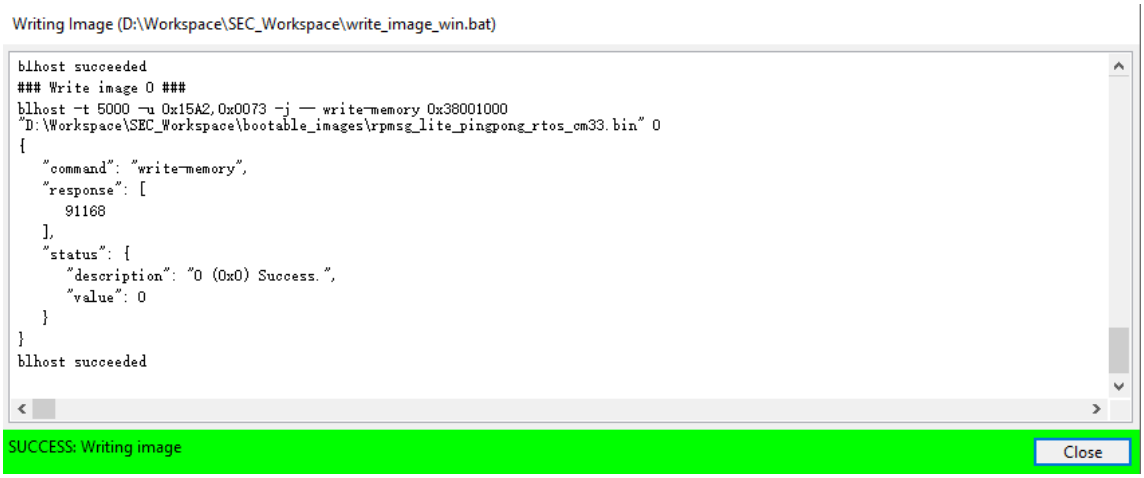
   It is required for certain application images which need special ELE FW service. For most SDK demos, leaves it empty.

5. Use Build image to build the combined bootable image. SW5[1..4] needs to be set to 0001 to let the SOC in serial download mode here.



6. Use Write image to flash the image.





After a successful write, change the SW5[1..4] to QSPI boot mode 0100 and reset the board. You can observe that the image is running from POR boot.

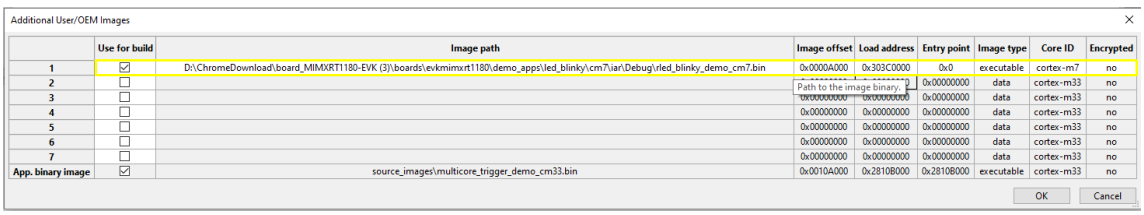
Parent topic: Use SPT tool and multicore_trigger image to kick off cm7 binary image

CM7 binary image runs from internal TCM, and multicore_trigger CM33 image runs from external FLASH

Note: Be very careful that this scenario requires fuse manipulation, which is not revertible.

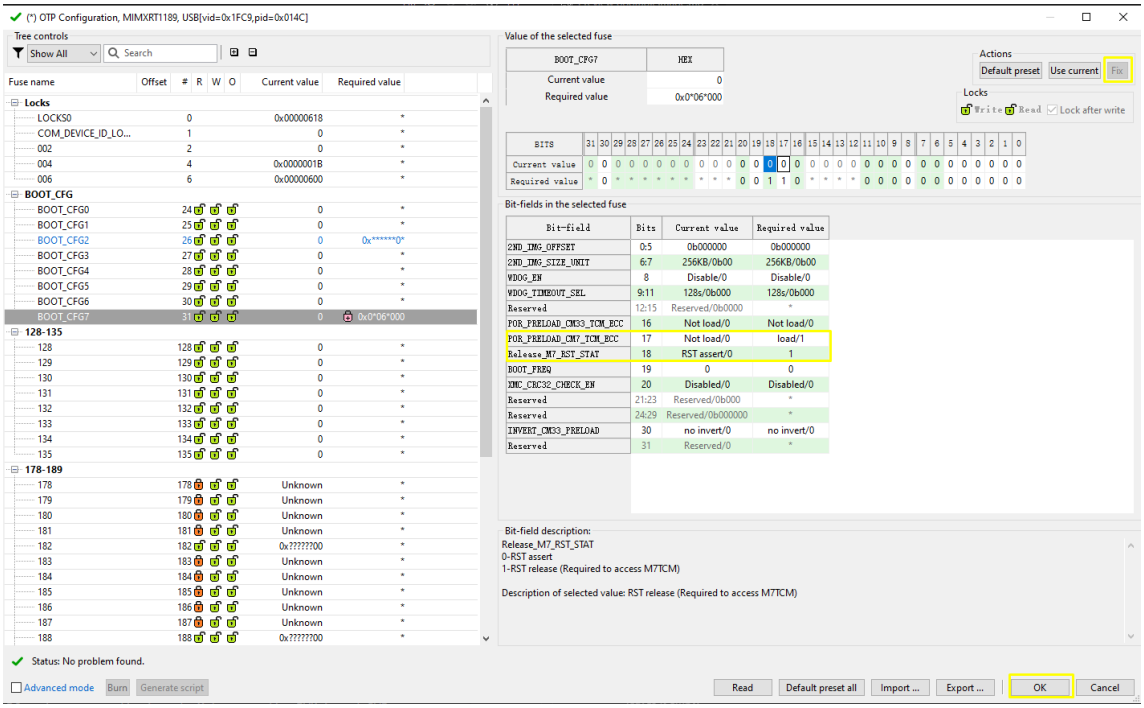
1. Prepare the multicore_trigger_demo_cm33 flash target image (flexspi_nor_debug or flexspi_nor_release) and specify it in Source executable image.
2. Use Additional images to specify CM7 application image running from ITCM memory. See *Table 1* for how to fill in necessary information for this additional image.

Parameter	Value	Explanation
Image offset	0xFF	The offset in bytes from the beginning of the current container header to beginning of the image. In our case, the CM7 TCM image is flashed to 0x2810_0000. The container header is put to 0x2800_1000 and the yield offset = 0xF_F000
Load address	0x303	The secured alias of CM7 ITCM in CM33/ROM memory space. ROM will copy image to this address
Entry point	0x0	The start address of vector table from CM7 core address space
Core ID	cortex m7	Specify the core ID
Image type	Executal	SPecify the image type
Encrypt	No	Specify the encryption



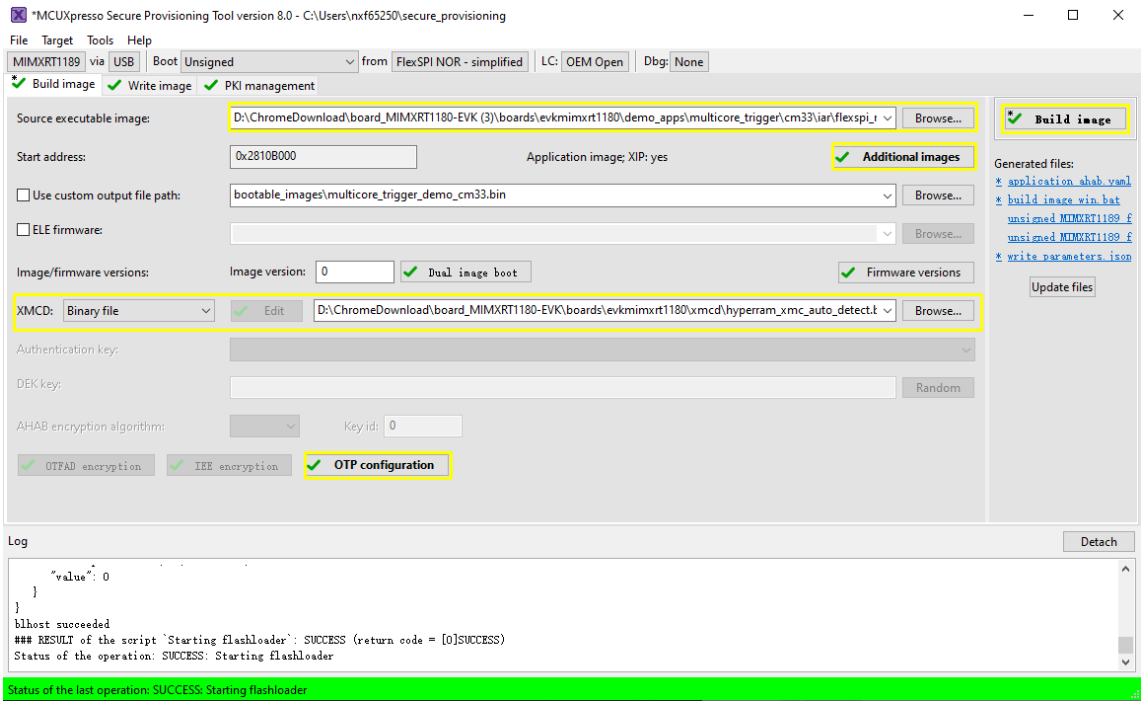
3. For ROM to successful copy image from flash to CM7 TCM. CM7 TCM ECC should be pre-configured. To archive this, a Fuse need to be set. Click OTP configuration, set POR_PRELOAD_CM7_TCM_ECC and RELEASE_M7_RST_STAT fuses in BOOT_CFG7 to 1.

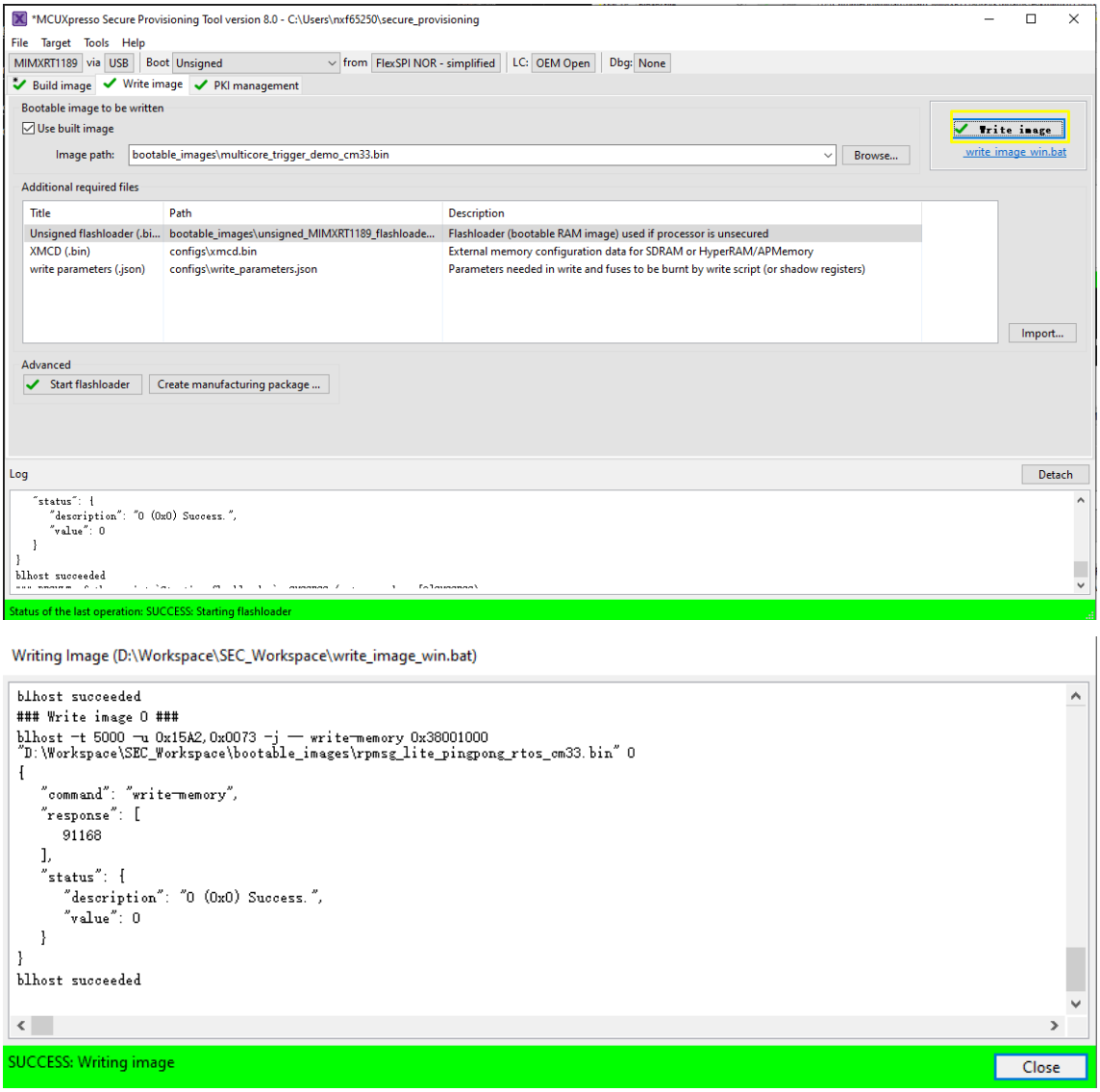
   Note: Be very careful that any setting on OTP cannot be reverted once set. There is possibility that the chip get bricked if you are fusing the wrong fuse word. Only do this if you really need to try this boot scenario.



4. (Optional) ELE firmware.

   It is needed for a certain application image which needs a special ELE FW service. For most SDK demos, simply leave it empty.





After a successful write, change the SW5[1..4] to QSPI boot mode 0100 and reset the board. You can observe that the image is running from the POR boot.

Note:

1. The new image constructed under **Build images** is still not a complete image, and the Memory Configuration Block is missing. When writing image, SPT downloads the contents of the Memory Configuration Block to FLASH.
2. When CM33 + CM7 image runs with the following target combination, it is required to set CM33_SET_TRDC to 1U in multicore_trigger demo, also requires to set CM33_SET_TRDC to 1U in CM7 demo if it exists.

```
- CM33 flexspi + CM7 flexspi, flexspi means flexspi_nor_debug/release, flexspi_nor_sdram_debug/
↪release, flexspi_nor_hyperram_debug/release
- CM33 HYPERRAM + CM7 HYPERRAM, HYPERRAM means hyperram_txt_debug/release.
```

3. Only those cm7 projects whose linkages are similar with hello_world_demo_cm7, support POR run via multicore_trigger_cm33.

Parent topic: Use SPT tool and multicore_trigger image to kick off cm7 binary image

Parent topic:*Bootable image generation*

RAW/POR image switch In current delivered SDK package, most demos generate RAW images. To enhance customer OOBE experience, the following RT1180 SDK CM33 project/targets generate POR image, which means that they can POR run after you debug/download it in IDE.

- IAR/GCC/MDK, *hello_world_demo_cm33* *flexspi_nor_debug/release*, *flexspi_nor_hyperram_debug/release*
- IAR/GCC/MDK, *multicore_hello_world_cm33* *flexspi_nor_debug/release*, *flexspi_nor_hyperram_debug/release*
- MCUX CM33 image are all POR images except the multicore_trigger demo.

Note:

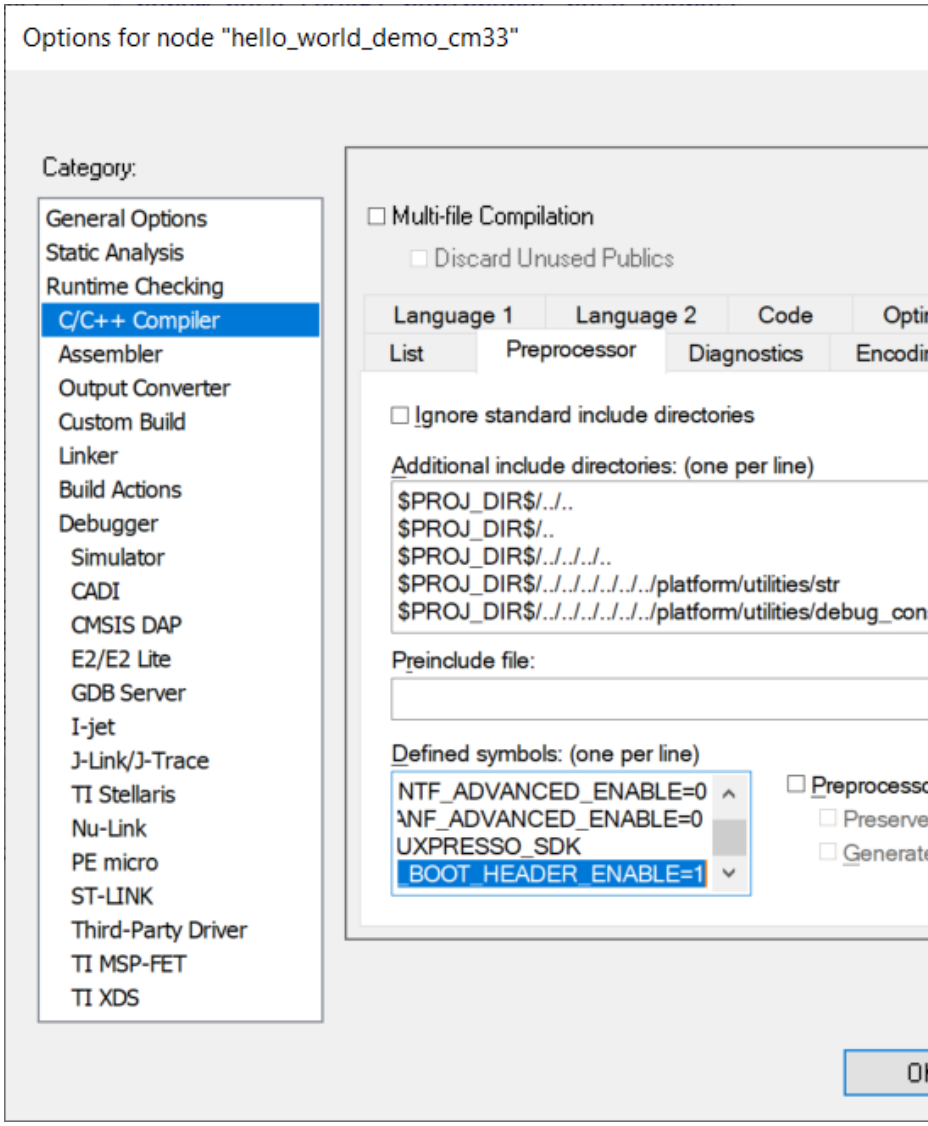
- These POR images are not signed, just for develop convenience, not recommended for product usage.
- It is highly recommended to use SPT for image download.

For the SDK CM33 IAR/MDK/ARMGCC project, flexspi_nor targets, as well as for MCUX project, debug/release targets. It is easy to switch RAW/POR image, via the project macro XIP_BOOT_HEADER_ENABLE setting. *Table 1*describes the rules.

XIP_BOOT_HEADER_ENABLE setting	Image type
1	POR
0	RAW

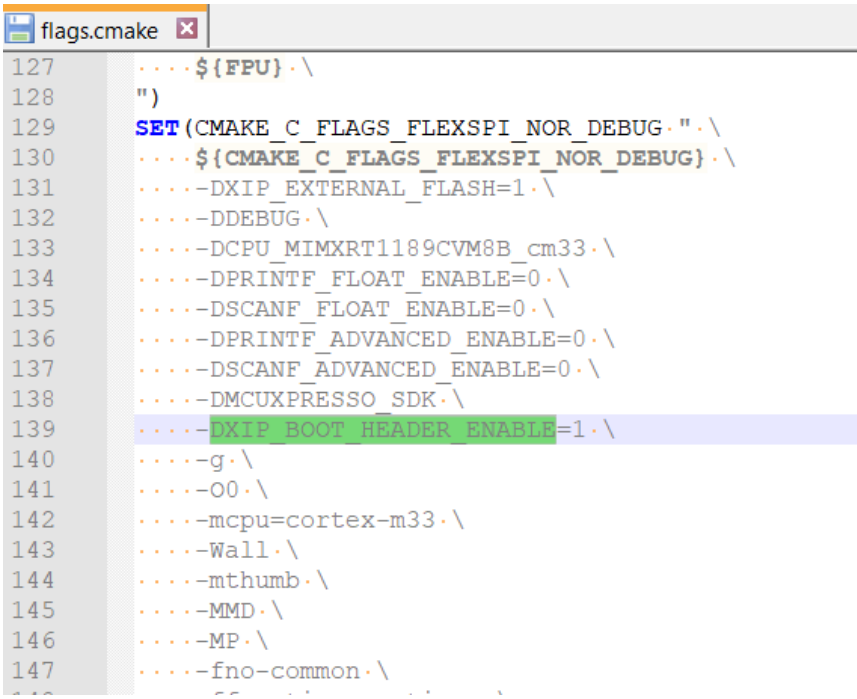
- For IAR/ARMGCC, change the macro XIP_BOOT_HEADER_ENABLE in the compiler setting.
- For MDK, change the macro XIP_BOOT_HEADER_ENABLE in compiler and link setting simultaneously.

For detailed settings, see the following sections.



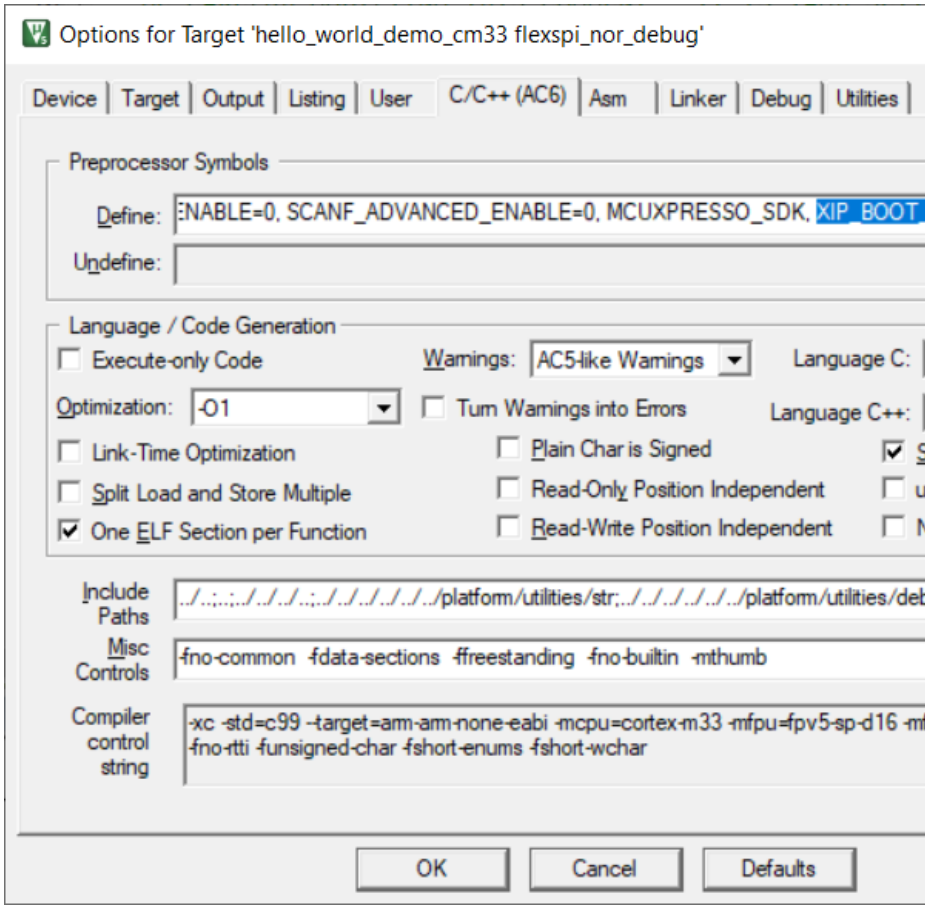
IAR settings for image type switch

Parent topic:RAW/POR image switch

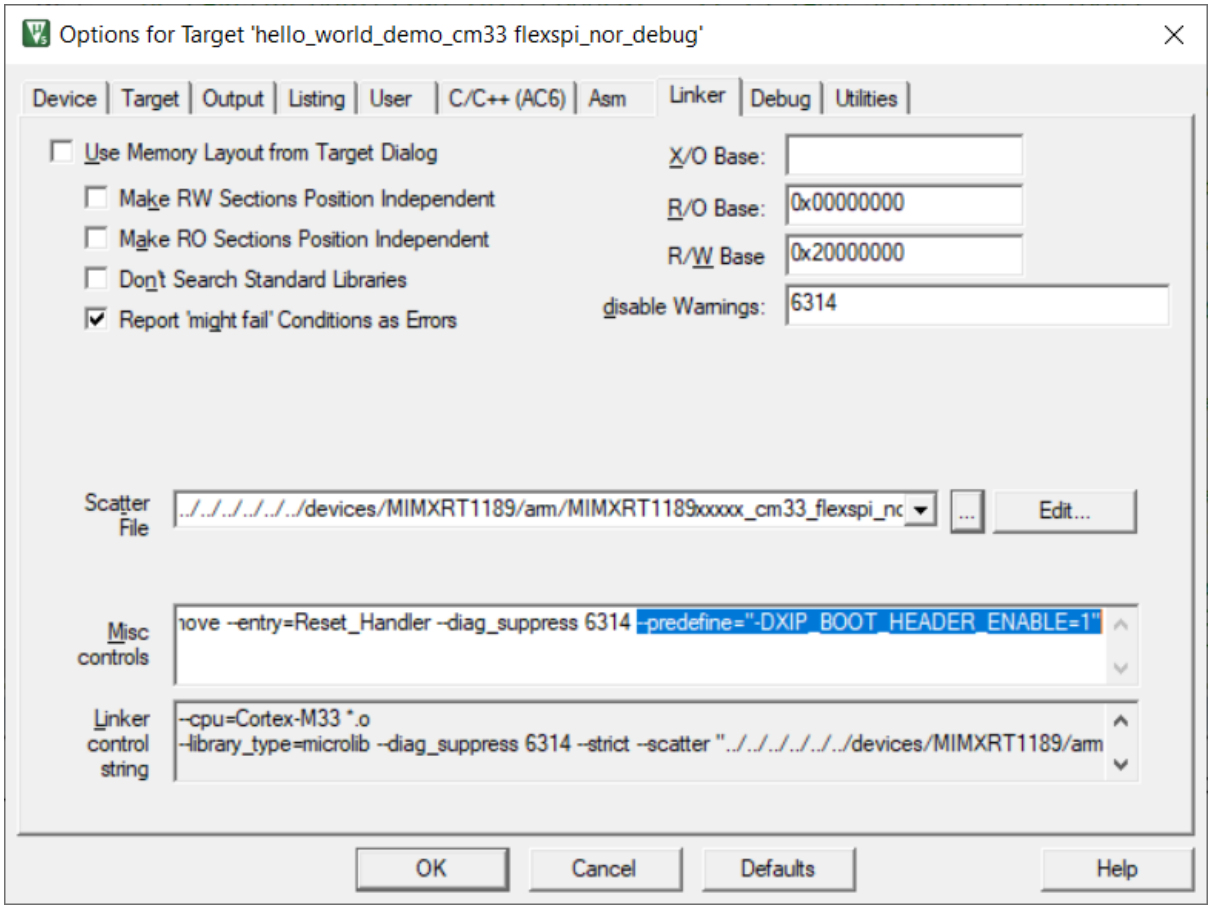


ARMGCC settings for image type switch

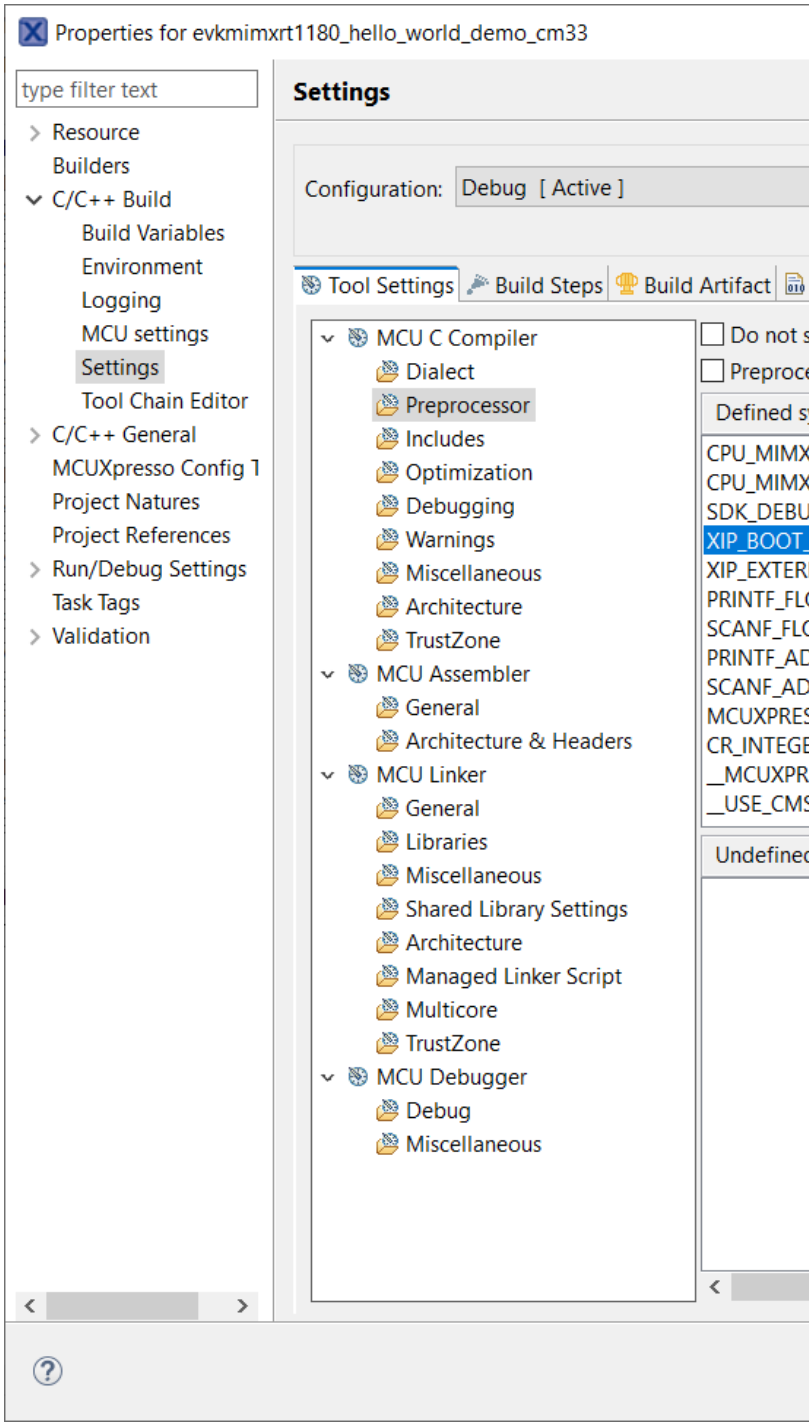
Parent topic:RAW/POR image switch



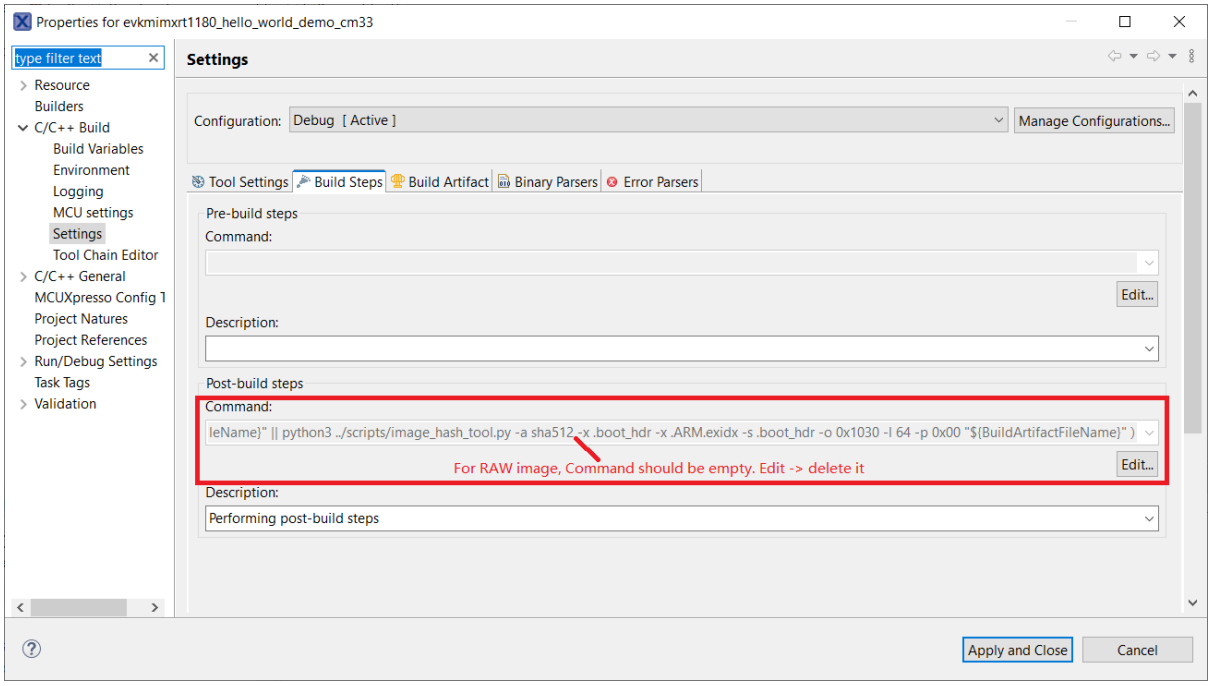
MDK setting for image type switch



Parent topic:RAW/POR image switch



MCUxpresso settings for image type switch

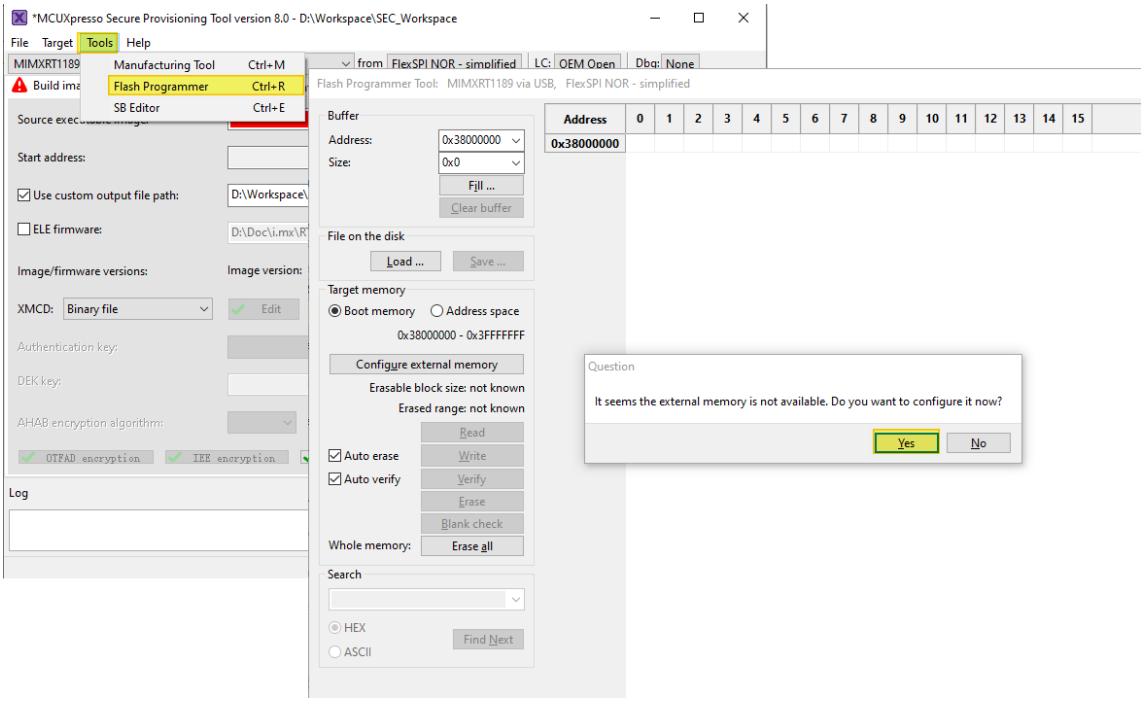


Parent topic:RAW/POR image switch

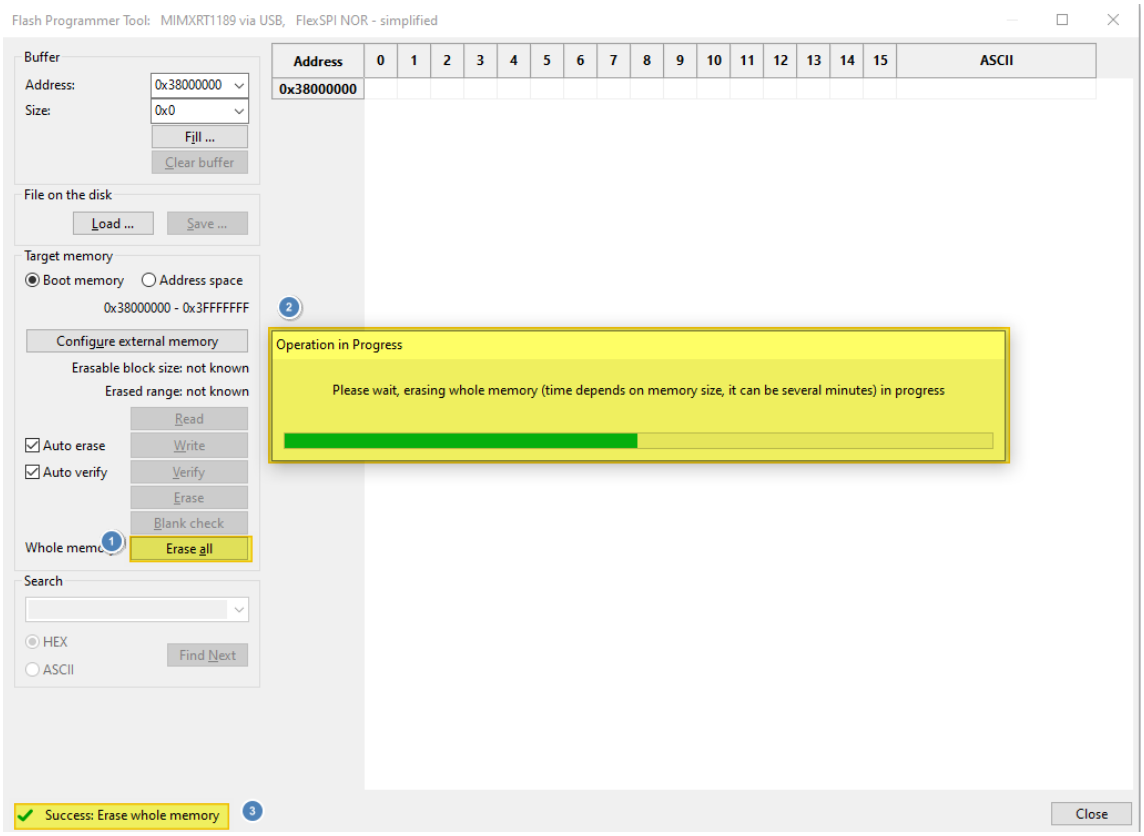
Parent topic:*Bootable image generation*

Use Secure Provisiong Tool to erase flash In worst cases a debugger cannot successfully connect to the board. A major reason for this is there are POR boot image inside the on board flash memory which causes the whole system in a trouble state. Secure Provisioning Tool provide a reliable way to erase the flash in this worst case. To achieve this

1. Set SW5[1:4] to 0001.
2. Connect the board to your PC from both J53 (Debugging) and J33 (USB OTG 1), then power up the board.
3. Start Secure Provisioning Tool, select Tool > Flash Programmer or Ctrl + R.
4. Click Yes to prompt up questions, wait memory initialization finish.



5. Click Erase all button and wait a minute for the erase process to finish.



6. After Success: Erase whole memory shows up, the board should have restored to a good state to work in various mode.

Parent topic:*Bootable image generation*

MCUXpresso configuration tools

MCUXpresso Config Tools can help configure the processor and generate initialization code for the on-chip peripherals. The tools are able to modify any existing example project, or create a configuration for the selected board or processor. The generated code is used with MCUXpresso SDK version 2.x.

Table 1 describes the tools included in the MCUXpresso config tools.

Config tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	
Clock tool	For system clock configuration.	
Peripherals tools	For configuration of other peripherals	
TEE tool	Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.	

MCUXpresso Config Tools can be accessed in the following products:

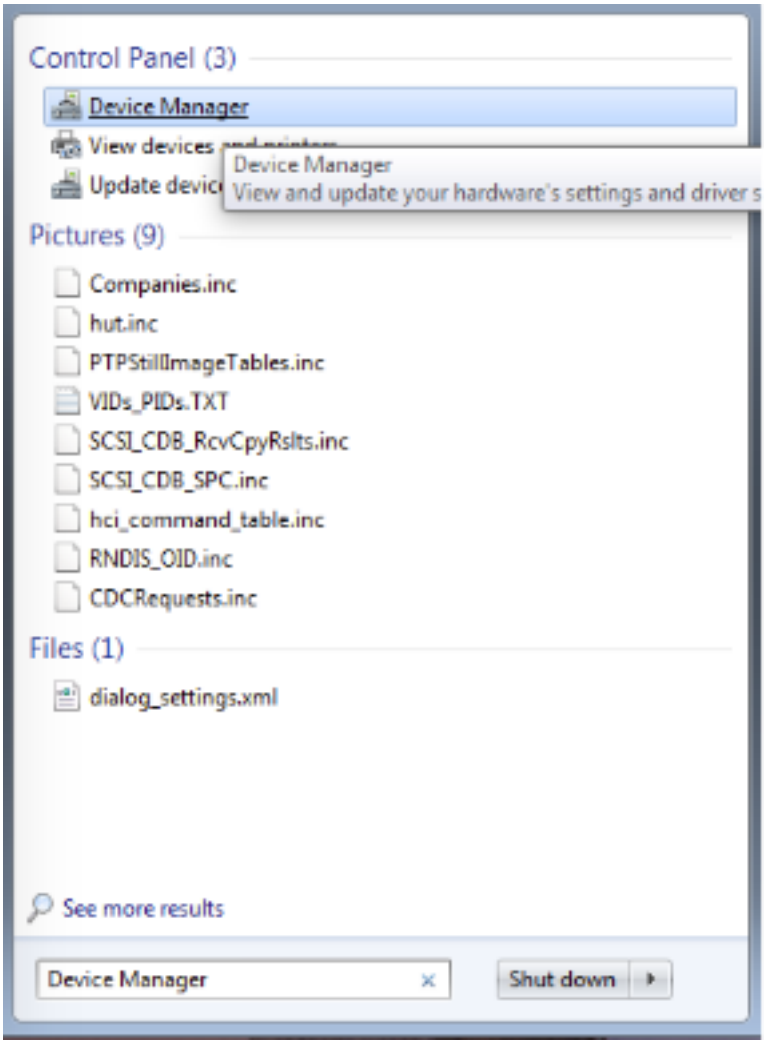
- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from MCUXPRESSO. Recommended for customers using IAR Embedded Workbench, Keil MDK µVision, or Arm GCC.
- **Online version** available on MCUXPRESSO. Recommended ding a quick evaluation of the processor or use the tool without installation.

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

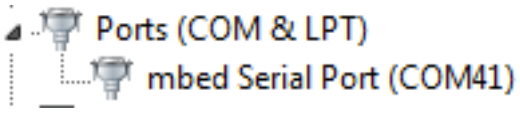
How to determine COM port

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

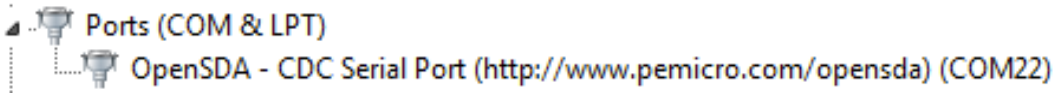
1. To determine the COM port, open the Windows operating system Device Manager. This can be achieved by going to the Windows operating system **Start** menu and typing **Device Manager** in the search bar, as shown in *Figure 1*.



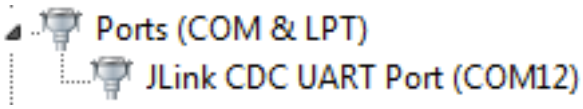
2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports. Depending on the NXP board you're using, the COM port can be named differently.
   1. OpenSDA – CMSIS-DAP/mbed/DAPLink interface:

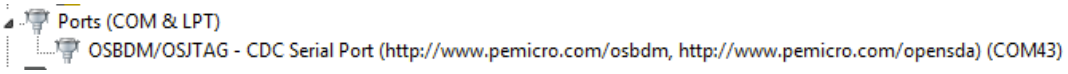
   2. OpenSDA – P&E Micro:

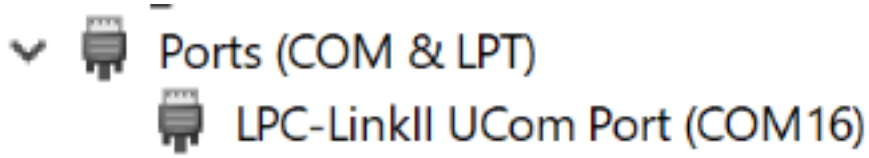
   3. OpenSDA – J-Link:

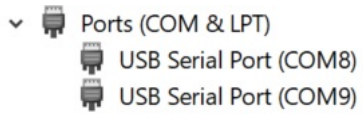
   4. P&E Micro OSJTAG:

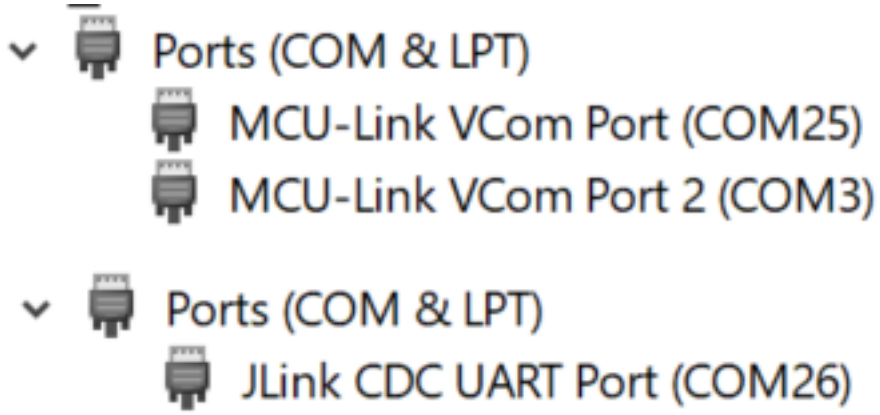
   5. LPC-Link2:



6. FTDI UART:



7. MCU-Link::



Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with a variety of factory programmed debug interface configurations. *Table 1* lists the hardware platforms supported by the MCUXpresso SDK, their default debug interface, and any version information that helps differentiate a specific interface configuration.

Hardware platform	Default interface	OpenSDA details*1*
EVK-MC56F83000	P&E Micro OSJTAG	N/A
EVK-MIMXRT595	CMSIS-DAP	N/A
EVK-MIMXRT685	CMSIS-DAP	N/A
FRDM-K22F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-K28F	DAPLink	OpenSDA v2.1
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2.1
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2.1
FRDM-K32W042	CMSIS-DAP	N/A
FRDM-K64F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
FRDM-K66F	J-Link OpenSDA	OpenSDA v2.1
FRDM-K82F	CMSIS-DAP	OpenSDA v2.1
FRDM-KE15Z	DAPLink	OpenSDA v2.1
FRDM-KE16Z	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.2
FRDM-KL02Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL03Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL25Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL26Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL27Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL28Z	P&E Micro OpenSDA	OpenSDA v2.1
FRDM-KL43Z	P&E Micro OpenSDA	OpenSDA v1.0

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details*1*
FRDM-KL46Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL81Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KL82Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KV10Z	CMSIS-DAP	OpenSDA v2.1
FRDM-KV11Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KV31F	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KW24	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-KW36	DAPLink	OpenSDA v2.2
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
HVP-KE18F	DAPLink	OpenSDA v2.2
HVP-KV46F150M	P&E Micro OpenSDA	OpenSDA v1
HVP-KV11Z75M	CMSIS-DAP	OpenSDA v2.1
HVP-KV58F	CMSIS-DAP	OpenSDA v2.1
HVP-KV31F120M	P&E Micro OpenSDA	OpenSDA v1
JN5189DK6	CMSIS-DAP	N/A
LPC54018 IoT Module	N/A	N/A
LPCXpresso54018	CMSIS-DAP	N/A
LPCXpresso54102	CMSIS-DAP	N/A
LPCXpresso54114	CMSIS-DAP	N/A
LPCXpresso51U68	CMSIS-DAP	N/A
LPCXpresso54608	CMSIS-DAP	N/A
LPCXpresso54618	CMSIS-DAP	N/A
LPCXpresso54628	CMSIS-DAP	N/A
LPCXpresso54S018M	CMSIS-DAP	N/A
LPCXpresso55s16	CMSIS-DAP	N/A
LPCXpresso55s28	CMSIS-DAP	N/A
LPCXpresso55s69	CMSIS-DAP	N/A
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0
MIMXRT1170-EVK	CMSIS-DAP	N/A
MIMXRT1180-EVK	CMSIS-DAP	OpenSDA v2.2
TWR-K21D50M	P&E Micro OSJTAG	N/AOpenSDA v2.0
TWR-K21F120M	P&E Micro OSJTAG	N/A
TWR-K22F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K24F120M	CMSIS-DAP/mbed	OpenSDA v2.1
TWR-K60D100M	P&E Micro OSJTAG	N/A
TWR-K64D120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K64F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K80F150M	CMSIS-DAP	OpenSDA v2.1
TWR-K81F150M	CMSIS-DAP	OpenSDA v2.1
TWR-KE18F	DAPLink	OpenSDA v2.1
TWR-KL28Z72M	P&E Micro OpenSDA	OpenSDA v2.1
TWR-KL43Z48M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KL81Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KL82Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KM34Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KM35Z75M	DAPLink	OpenSDA v2.2
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV11Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV31F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV46F150M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV58F220M	CMSIS-DAP	OpenSDA v2.1

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details*1*
TWR-KW24D512	P&E Micro OpenSDA	OpenSDA v1.0
USB-KW24D512	N/A External probe	N/A
USB-KW41Z	CMSIS-DAP\DAPLink	OpenSDA v2.1 or greater

1 The OpenSDA details is not applicable to LPC.

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

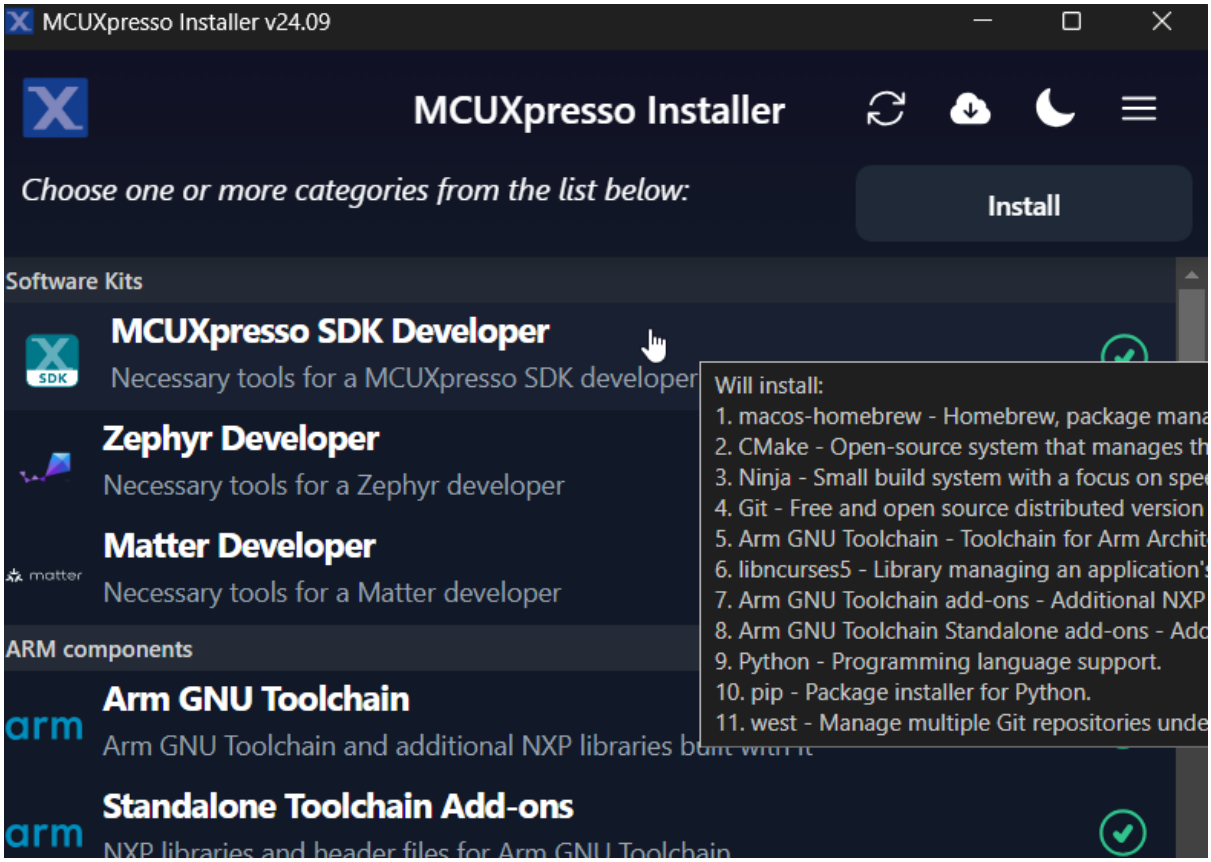
1.3.1 Getting Started with MCUXpresso SDK Repository

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.

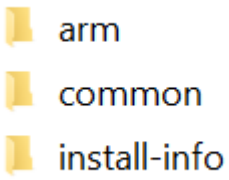
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

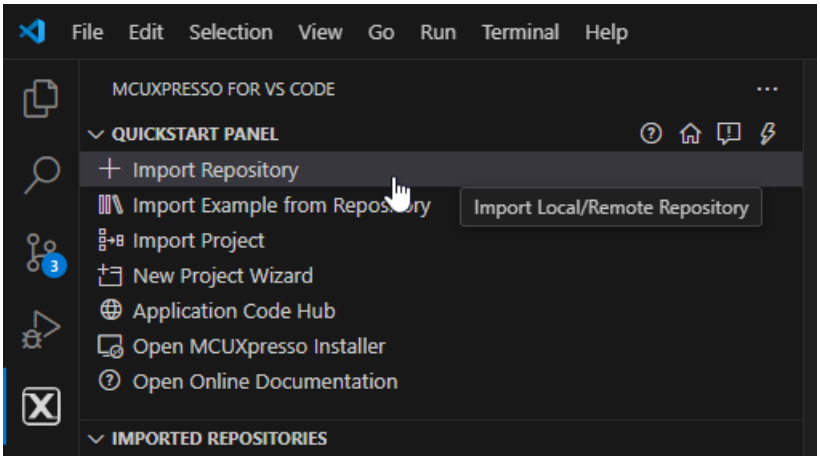
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the hello_world demo application as an example. However, these

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*.

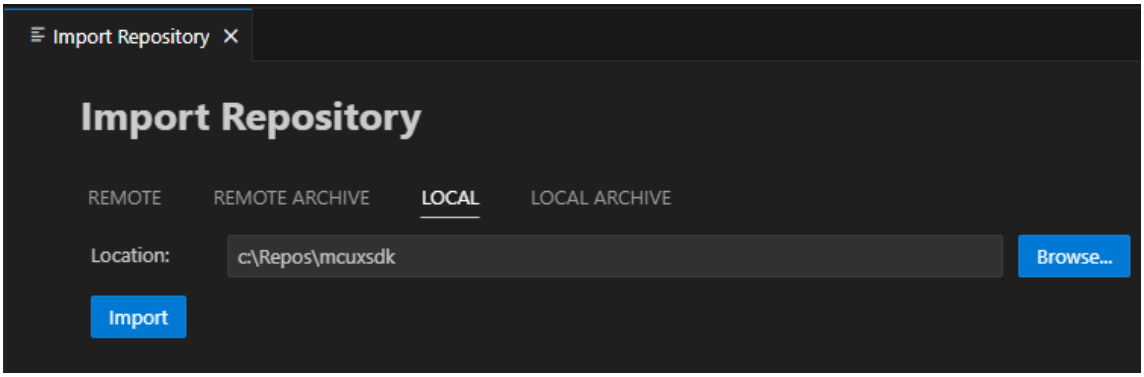
To build an example application:

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

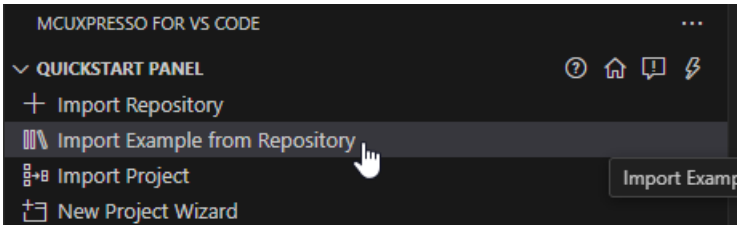


Note: You can import the SDK in several ways. Refer to MCUXpresso for VS Code Wiki for details.

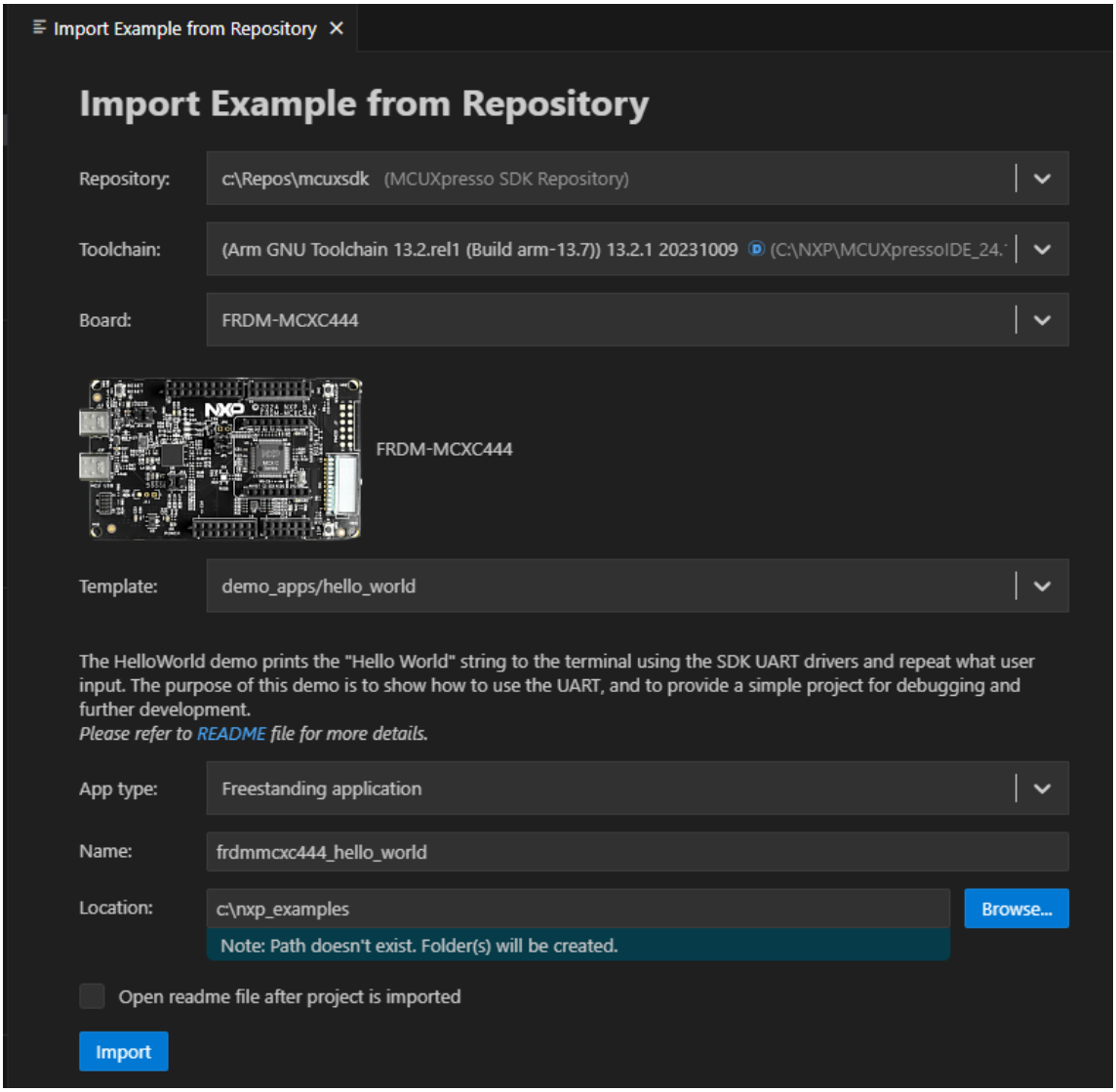
Select **Local** if you've already obtained the SDK as seen in *Get MCUXpresso SDK Repo*. Select your location and click **Import**.



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

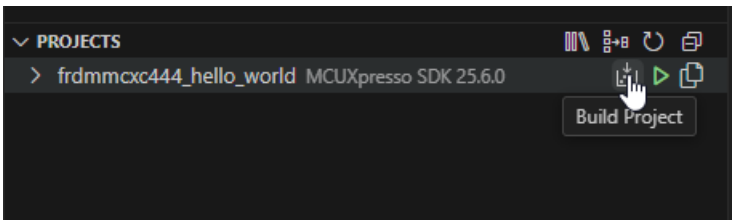


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

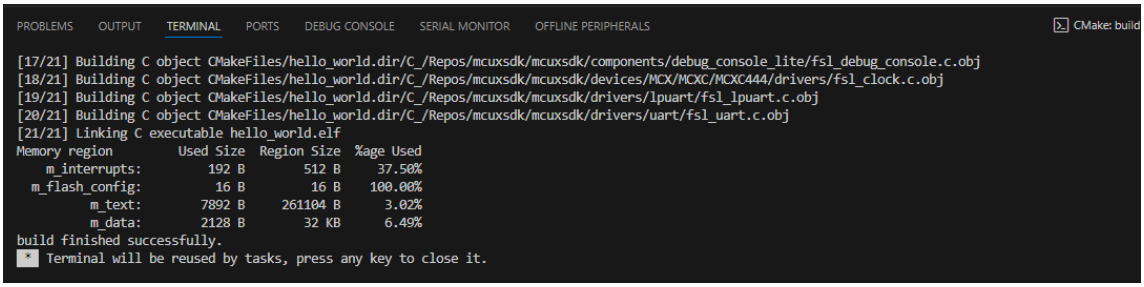


Note: The MCUXpresso SDK projects can be imported as **Repository applications** or **Freestanding applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Freestanding examples can be imported to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

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.

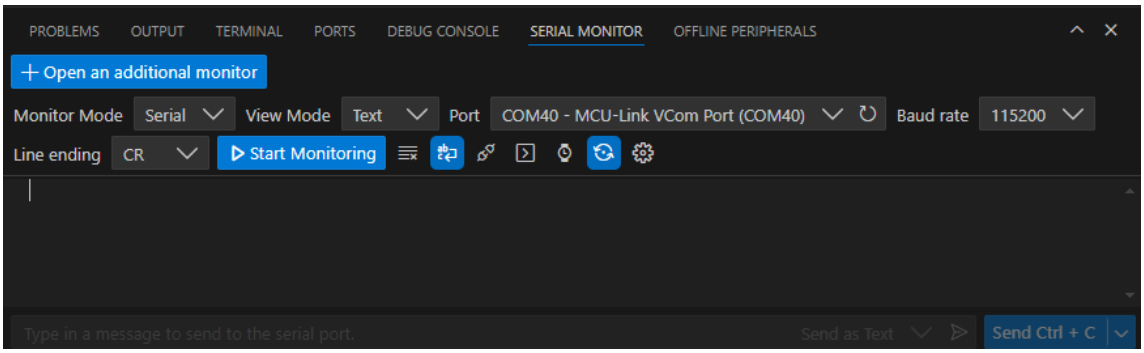


The integrated terminal will open at the bottom and will display the build output.

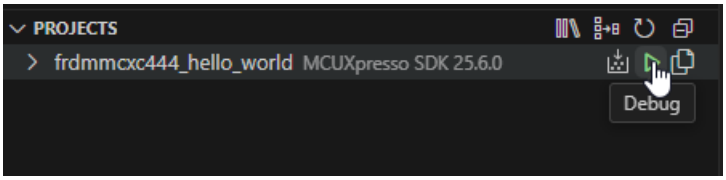


Run an example application **Note:** for full details on MCUXpresso for VS Code debug probe support, see MCUXpresso for VS Code Wiki.

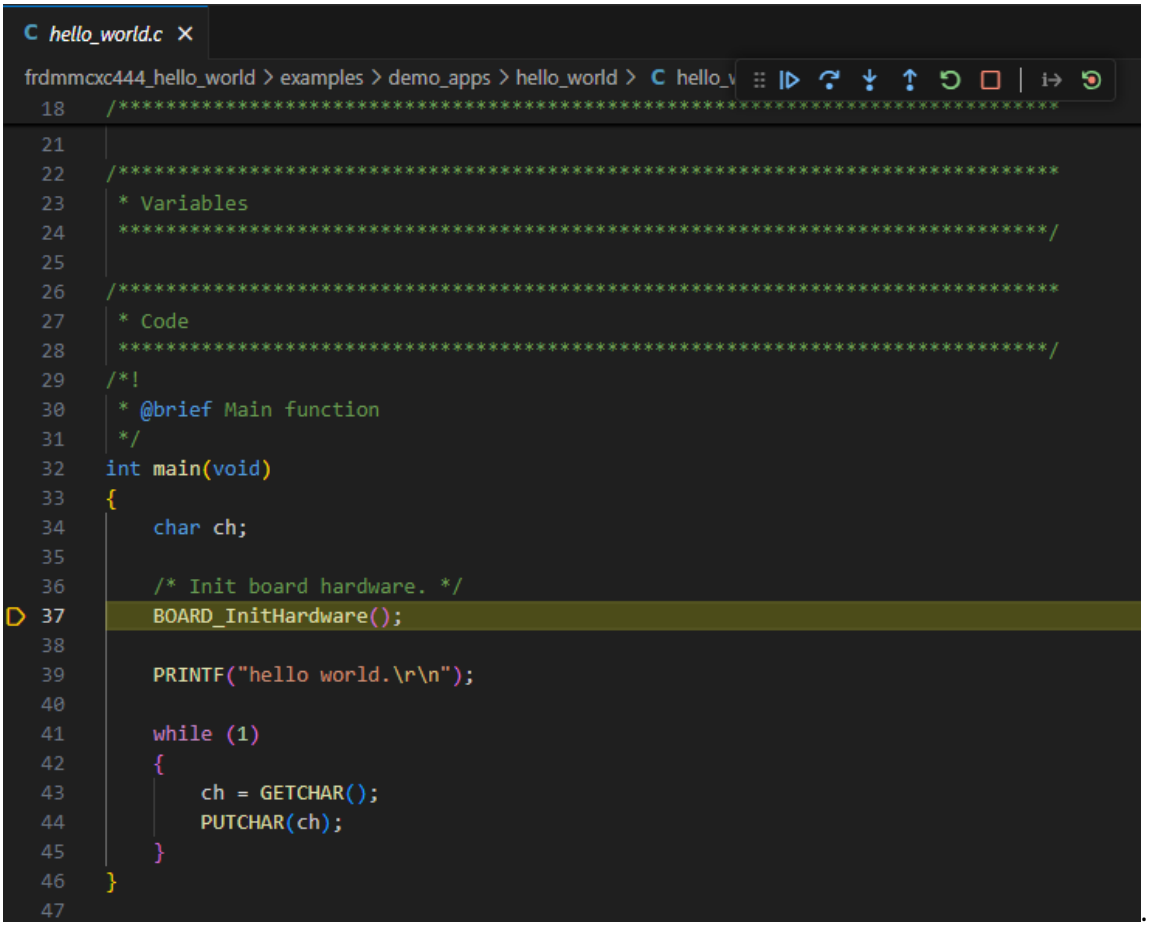
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



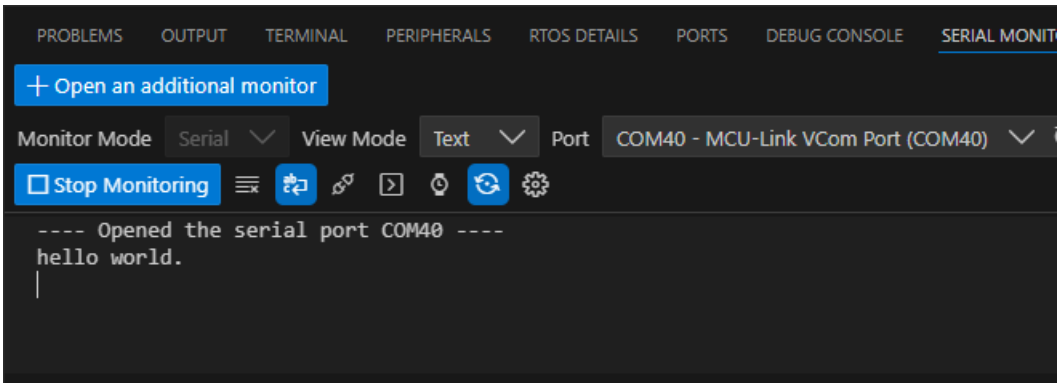
2. Navigate to the **PROJECTS** view and click the play button to initiate a debug session.



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

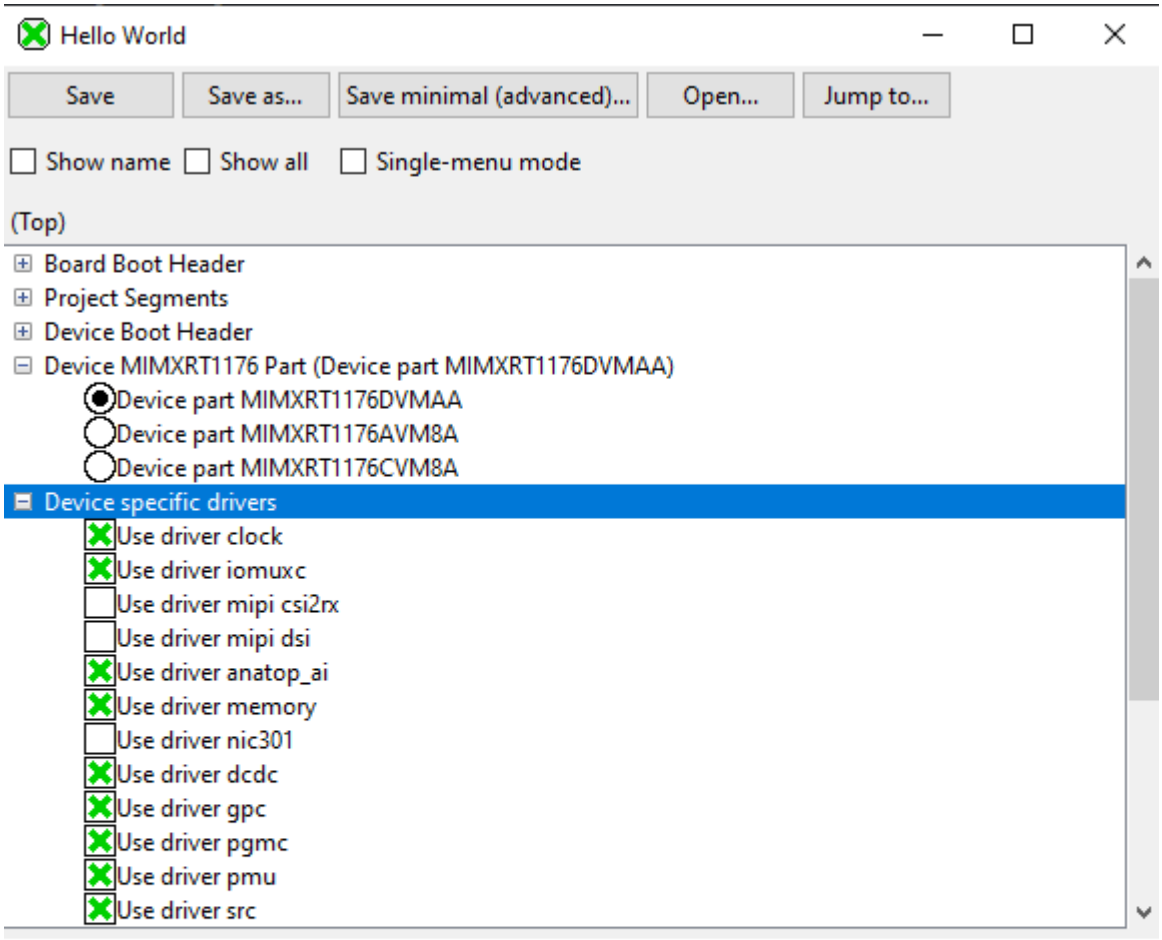
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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:

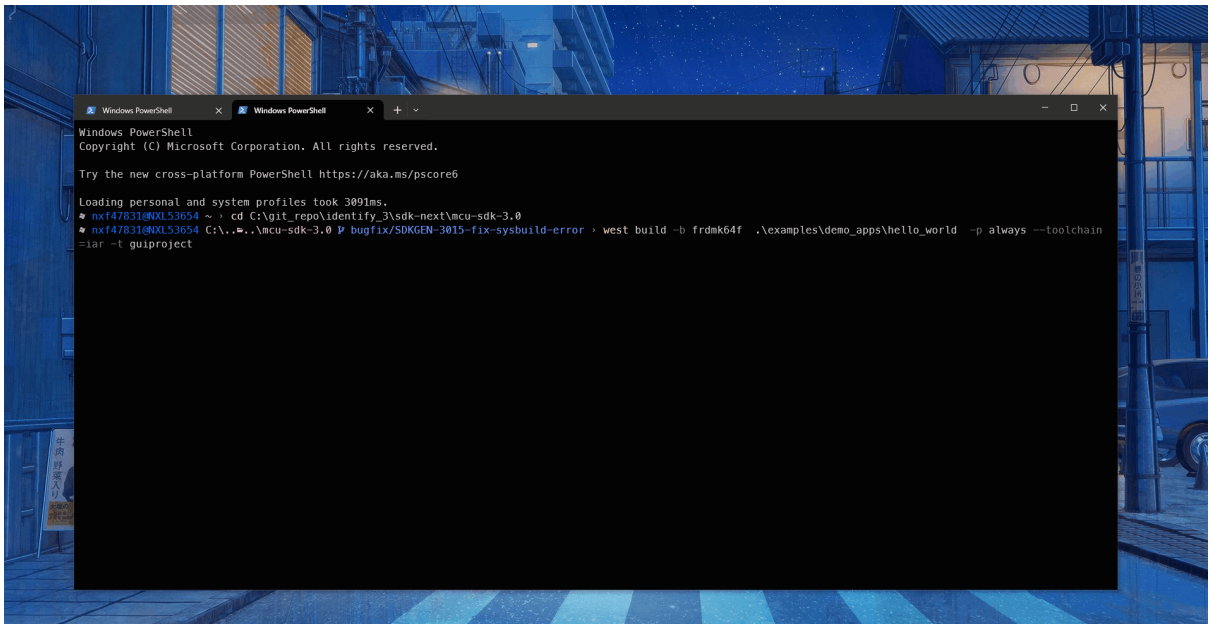
```
west debug -r linkserver
```

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 Release Notes

1.4.1 MCUXpresso SDK Release Notes

Overview

The MCUXpresso SDK is a comprehensive software enablement package designed to simplify and accelerate application development with Arm Cortex-M-based devices from NXP, including its general purpose, crossover and Bluetooth-enabled MCUs. MCUXpresso SW and Tools for DSC

further extends the SDK support to current 32-bit Digital Signal Controllers. The MCUXpresso SDK includes production-grade software with integrated RTOS (optional), integrated enabling software technologies (stacks and middleware), reference software, and more.

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

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

MCUXpresso SDK

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

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

Development tools

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

- MCUXpresso IDE, Rev. 25.06.xx
- IAR Embedded Workbench for Arm, version is 9.60.4
- Keil MDK, version is 5.42
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

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

Development board	MCU devices			
MIMX EVK	MIMXRT1181CVP2B, MIMXRT1182CVP2B, MIMXRT1187AVM8B, MIMXRT1187CVM8C, MIMXRT1189CVM8B, MIMXRT1189XVM8C	MIMXRT1181CVP2C, MIMXRT1182CVP2C, MIMXRT1187AVM8C, MIMXRT1187XVM8B, **MIMXRT1189CVM8C**,	MIMXRT1181XVP2B, MIMXRT1182XVP2B, MIMXRT1187CVM8B, MIMXRT1187XVM8C, MIMXRT1189XVM8B,	MIMXRT1181XVP2C, MIMXRT1182XVP2C,

MCUXpresso SDK release package

The MCUXpresso SDK release package content is aligned with the silicon subfamily it supports. This includes the boards, CMSIS, devices, middleware, and RTOS support.

Device support The device folder contains the whole software enablement available for the specific System-on-Chip (SoC) subfamily. This folder includes clock-specific implementation, device register header files, device register feature header files, and the system configuration source files. Included with the standard SoC support are folders containing peripheral drivers, toolchain support, and a standard debug console. The device-specific header files provide a direct access to the microcontroller peripheral registers. The device header file provides an overall SoC memory mapped register definition. The folder also includes the feature header file for each peripheral on the microcontroller. The toolchain folder contains the startup code and linker files for each supported toolchain. The startup code efficiently transfers the code execution to the main() function.

Board support The boards folder provides the board-specific demo applications, driver examples, and middleware examples.

Demo application and other examples The demo applications demonstrate the usage of the peripheral drivers to achieve a system level solution. Each demo application contains a readme file that describes the operation of the demo and required setup steps. The driver examples demonstrate the capabilities of the peripheral drivers. Each example implements a common use case to help demonstrate the driver functionality.

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Middleware

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

MCU Boot MCU Boot (formerly KBOOT) NXP/Freescale proprietary loader

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

Ethermind BT/BLE Stack nxp_bt_ble_stack

coreHTTP coreHTTP

wpa_supplicant-rtos NXP Wi-Fi WPA Supplicant

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).

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

FreeModbus FreeModbus Library

Simple Open EtherCAT Master Simple Open EtherCAT Master (SOEM) is an open source EtherCAT master stack that is used to write custom EtherCAT Master applications. For more information on how to use SOEM, see the Getting Started with MCUXpresso SDK for SOEM document.

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

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

MCU Boot Open source MCU Bootloader.

mbedTLS mbedtls SSL/TLS library v3.x

mbedTLS mbedtls SSL/TLS library v2.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

llhttp HTTP parser llhttp

LittleFS LittleFS filesystem stack

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

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

NAND Flash Management Stack NAND Flash Management Stack

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

ELE Crypto Crypto functionality provided Edgelock Core

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.

Flashing CM7 Image After Bootable CM33 Image Breaks Build

Issue Description In SPI boot mode ($\mathrm{SW5}[1..4] = 0100$), after using IDE/Secure Provisioning Tool to flash a bootable CM33 image, if any $\mathrm{flexspi_nor}$ target CM7 image is programmed using IDE, the SOC will enter an unresponsive state and all further debugging attempts will fail.

Prevention To avoid this issue:

- After flashing a bootable CM33 image, always erase the flash before flashing any non-bootable CM7 image.

Recovery Steps If you accidentally enter this state:

1. Change to Internal Boot mode or Serial Download mode ($\mathrm{SW}[1..4] = 0000/0001$)
2. Use IDE/Secure Provisioning Tool to erase the flash
3. The system should now be restored to normal operation

Note for Customer Boards For customer boards without a boot mode switch, we are working on a permanent fix which is expected to roll out soon.

Reference

Ticket	Description	Version
MCUX-76778	Flash a CM7 image following a CM33 bootable image will break the build	25.03.00

Segger version

The lowest required J-Link version is v7.94g.

AWS examples are not fully tested

The examples under boards/evkmimxrt1180/aws_examples are not fully tested. The associated features and functionalities are only for the evaluation purpose. Full feature verification will be covered in a future release.

Missing fsl_cache when create new project with board or kit

When creating a new C/C++ project, if evkmimxrt1180 or evkmimxrt1180_om13790host kit is selected, fsl_cache component gets missed and causes build failure. To avoid the build failiure, manually select it in **SDK Wizard** page.

Debugging problems

When debugging hyperram_txt_debug or hyperram_txt_release target, user may encounter issue when debug with breakpoint. The reason is because code is cached. The default breakpoint type is auto/software breakpoint, it will lead soc core and debugger misaligned. As a workaround, please set breakpoint type to hardware breakpoint.

In addition, hyperram_txt targets can't debug on IAR versions: 9.50.1, 9.50.2, 9.60.1. The resolution is: delete the selected contents as shown in *Figure 1* in IAR path: *arm\config\flashloader\NXP\FlashIMXRT1180-EVK_FlexSPI_M33.board*.

Other issues

The following are additional known issues in this release.

Ticket	Description	History
MCUX-66649	Build erros when create new project wizard with various component	2.15.000 RT1180
MCUX-65840	Semihost terminal is not working well with JLink	2.15.000 RT1180
MCUX-66442	Edgefast a2dp_sink does not work properly on release target of MDK toolchain	2.15.000 RT1180
MCUX-65874	WIFI wifi_cli and wifi_wpa_supplicant will show no iperf summary in UDP server mode with both ipv4 and ipv6 in CM33 core	2.15.000 RT1180

Security ELE requires ping every 24 hours

The security ELE requires ping every 24 hours, which is mandatory.

Most of RT118x demos donot ping ELE every 24 hours, because those demos focus on the function demonstrate only. It is still MUST to ping ELE every 24 hours if demo run duration > 24 hours.

Check hello_world_demo_cm33 for example on how to ping ELE.

CM33 NPW project in MCUXpresso does not support POR run by default

When creating a new CM33 C/C++ project in MCUXpresso, the default is RAW image, which means it does not support POR run.

To make a default POR image, check the "**RAW/POR image switch**" section in Getting Started document.

Example hello_world_ns fails with mdk toolchain

The example hello_world_ns fails with mdk toolchain while switching to the non-secure world.

Examples: hello_world_ns, hello_world_s **Affected toolchains**: mdk **Affected platforms**: evkmimxrt1180

The aws_examples don't work

The example stucks during TLS connection.

Affected toolchains: mcux **Affected platforms:** evkmimxrt1180

Examples hello_world_ns, secure_faults_ns, and secure_faults_trdc_ns have incorrect library path in GUI projects

When the affected examples are generated as GUI projects, the library linking the secure and non-secure worlds has an incorrect path set. This causes linking errors during project compilation.

Examples: hello_world_ns, hello_world_s, secure_faults_ns, secure_faults_s, secure_faults_trdc_ns, secure_faults_trdc_s

Affected toolchains: mdk, iar

Workaround: In the IDE project settings for the non-secure (_ns) project, find the linked library (named hello_world_s_CMSE_lib.o, or similar, depending on the example project) and replace the path to the library with <build_directory>/<secure_world_project_folder>/<IDE>/, replacing the subdirectory names with the build directory, the secure world project name, and IDE name.

1.5 ChangeLog

1.5.1 MCUXpresso SDK Changelog

Board Support Files

board

[25.06.00]

- Initial version
  - Replaced the XXX_SERIES macros with the CPU_xxx macros in the common configuration file
  - Added the legacy IAR supporting Mac file

clock_config

[25.06.00]

- Initial version
  - Updated the clock_config.c to align with the latest part number and the clock tool model
  - Removed the OSC RC 400M dating control, which is not supported by the hardware

pin_mux

[25.06.00]

- Initial version
  - Updated pin_mux.c to align with the latest part number and the pin tool model

---

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.

AOI

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

ASRC

[2.1.3]

- Bug Fixes
  - Fixed function did not match the specified channel pair issue.

[2.1.2]

- Improvements
  - Correct feature name in source file by changing FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_A to FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_ASRPM.
  - Removed the asrc_clock_source_t from driver header file, as SOC header file will provide detail definition.
- Bug Fixes
  - Fixed the ASRC_SetChannelPairConfig/ASRC_ChannelPairEnable functions missing functionality when using channel pair B/C.
  - Fixed violations of the MISRA C-2012 rules 10.7.

[2.1.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1□10.4, 12.2.

[2.1.0]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 10.1, 17.7, 11.9, 8.6, 12.2, 11.6.

[2.0.1]

- Improvements
  - Added feature macro FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_ASPRM for ASRC parameter register.
- Bug Fixes
  - Fixed the unused build warning in asrc edma driver.

[2.0.0]

- Initial version.

ASRC EDMA Driver

[2.2.0]

- Bug Fixes
  - Fixed the “watermark” and “channel” was defined in struct asrc_p2p_edma_config_t but never used issue.

[2.1.0]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 10.1, 17.7, 11.9, 8.6, 12.2, 11.6.

[2.0.1]

- Improvements
  - Added feature macro FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_ASPRM for ASRC parameter register.
- Bug Fixes
  - Fixed the unused build warning in asrc edma driver.

[2.0.0]

- Initial version.

BBNSM

[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.

CACHE ARMv7-M7

[2.0.4]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

CLOCK

[2.2.0]

- New Features
  - Added APIs to set/get CLKO1/O2.

[2.1.5]

- Improvements
  - update OSC_RC_400M STAT registers to read only.

[2.1.4]

- Bug Fixes
  - Fix edma base ptrs and edma clocks did not match issue.

[2.1.3]

- Improvements
  - Move s_clockSourceName array to c from header.

[2.1.2]

- Comment improvements for CLOCK_InitPfd()

[2.1.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 8.6, 10.3.
  - Fixed potential infinite loop in CLOCK_GetCoreSysClkFreq() function.

[2.1.0]

- Improvements
  - Added CLOCK_GetFreqFromObs() function.
- Bug Fixes
  - Fixed bug in CLOCK_InitArmPll() function.
  - Fixed bug in CLOCK_InitSysPll1() function.

[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.

DAC12

[2.1.2]

- Bug Fixes
  - Fixed CERT INT31-C issue.

[2.1.1]

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

[2.1.0]

- Improvements
  - Defined the macro "FSL_FEATURE_HAS_NO_ITRM_REGISTER" to distinguish different scenes that ITRM register may not equipped one some devices.

[2.0.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.8, 17.7.

[2.0.0]

- Initial version.

DCDC

[2.0.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 8.6.

[2.0.0]

- Initial version.

ECAT

[2.0.1]

- Fix Coverity warning
  - CID: 41891133
  - CID: 41897853

[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.

ELE_BASE_API

[1.0.0]

- initial version.

EQDC

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EWM

[2.0.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed the hard fault in EWM_Deinit.

[2.0.0]

- Initial version.

FLEXCAN

[2.14.5]

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

[2.14.4]

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

[2.14.3]

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

[2.14.2]

- Improvements
  - Add Coverage Justification for uncovered code.
  - Adjust API FLEXCAN_TransferAbortReceive order.
  - Update FLEXCAN_Enable to enter Freeze Mode first when enter Disable mode on some platform.
  - Added while loop timeout for following API:
    - FLEXCAN_EnterFreezeMode
    - FLEXCAN_ExitFreezeMode
    - FLEXCAN_Enable
    - FLEXCAN_Reset
    - FLEXCAN_TransferSendBlocking
    - FLEXCAN_TransferReceiveBlocking
    - FLEXCAN_TransferFDSendBlocking
    - FLEXCAN_TransferFDReceiveBlocking

    * FLEXCAN_TransferReceiveFifoBlocking
    * FLEXCAN_TransferReceiveEnhancedFifoBlocking

- Bug Fixes
  - Remove remote frame feature in CANFD mode because there is no remote frame in the CANFD format.
  - Remove legacy Rx FIFO disabled branch in FLEXCAN_SubHandlerForLegacyRxFIFO and FLEXCAN_SubHandlerForDataTransfered.

[2.14.1]

- Bug Fixes
  - Fixed register IMASK2-4 IFLAG2-4 HR_TIME_STAMPn access issue on FlexCAN instances with different number of MBs.
  - Fixed bit field MBDSR1-3 access issue on FlexCAN instances with different number of MBs.
- Improvements
  - Unified following API as same parameter and return value type:
    * FLEXCAN_GetMbStatusFlags
    * FLEXCAN_ClearMbStatusFlags
    * FLEXCAN_EnableMbInterrupts
    * FLEXCAN_DisableMbInterrupts
  - Add workaround for ERR050443 and ERR052403.
  - Update message buffer read process in API FLEXCAN_ReadRxMb and FLEXCAN_ReadFDRxMb to make critical section as short as possible.
  - Simplify API FLEXCAN_DriverDataIRQHandler implementation by remove parameter type.

[2.14.0]

- Improvements
  - Support external time tick feature.
  - Support high resolution timestamp feature.
  - Enter Freeze Mode first when enter Disable Mode on some platform.
  - Add feature macro for Pretended Networking because some FlexCAN instance do not have this feature.
  - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
  - Add new FlexCAN IRQ Handler FLEXCAN_DriverDataIRQHandler and FLEXCAN_DriverEventIRQHandler. Thses IRQ Handlers are used on soc which FlexCAN interrupts are grouped by specific function and assigned to different vector.
  - Update macro FLEXCAN_WAKE_UP_FLAG and FLEXCAN_PNWAKE_UP_FLAG to simplify code.
  - Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKMSK_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_NO_SLFWAK_SUPPORT.

  - Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKSRC_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_GLITCH_FILTER.
- Bug Fixes
  - Fixed wrong interrupt and status flag helper macro in enumeration _flexcan_flags and API FLEXCAN_DisableInterrupts.
  - Fixed interrupt flag helper macro typo issue.
  - Remove flags which will are unassociated with interrupt in macro FLEXCAN_MEMORY_ERROR_INT_FLAG.
  - Remove flags which will are unassociated with interrupt in macro FLEXCAN_ERROR_AND_STATUS_INT_FLAG.
  - Fixed array out-of-bounds access when read enhanced Rx FIFO.

[2.13.1]

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

[2.13.0]

- Improvements
  - Support payload endianness selection feature.

[2.12.0]

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

[2.11.8]

- Improvements
  - Synchronize flexcan driver update on s32z platform.

[2.11.7]

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

[2.11.6]

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

[2.11.5]

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

[2.11.4]

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

[2.11.3]

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

[2.11.2]

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

[2.11.1]

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

[2.11.0]

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

[2.10.1]

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

[2.10.0]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.7]

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

[2.8.6]

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

[2.8.5]

- Improvements
  - Make driver aarch64 compatible.

[2.8.4]

- Bug Fixes
  - Fixed FlexCan_Errata_6032 to disable all interrupts.

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.2]

- Improvements
  - Add CANFD frame data length enumeration.

[2.6.1]

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

[2.6.0]

- Improvements
  - Enable CANFD ISO mode in FLEXCAN_FDInit API.
  - Enable the transceiver delay compensation feature when enable FD operation and set bitrate switch.
  - Implementation memory error control in FLEXCAN_Init API.
  - Improve FLEXCAN_FDCalculateImprovedTimingValues API to get same value for FPRESDIV and PRESDIV.
  - Added memory error configuration for user.

        * enableMemoryErrorControl
        * enableNonCorrectableErrorEnterFreeze
    - Added memory error related APIs.
        * FLEXCAN_GetMemoryErrorReportStatus
        * FLEXCAN_GetMemoryErrorStatusFlags
        * FLEXCAN_ClearMemoryErrorStatusFlags
        * FLEXCAN_EnableMemoryErrorInterrupts
        * FLEXCAN_DisableMemoryErrorInterrupts
- Bug Fixes
    - Fixed the issue of sent duff CAN frame after call FLEXCAN_FDInit() API.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Bug Fixes
    - MISRA C-2012 issue check.
        * Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-11.8, rule-12.2, rule-13.4, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-16.4, rule-17.3, rule-5.8, rule-8.3, rule-8.5.
    - Fixed the issue that API FLEXCAN_SetFDRxMbConfig lacks inactive message buff.
    - Fixed the issue of Pa082 warning.
    - Fixed the issue of dead lock in the function of interruption handler.
    - Fixed the issue of Legacy Rx Fifo EDMA transfer data fail in evkmimxrt1060 and evkmimxrt1064.
    - Fixed the issue of setting CANFD Bit Rate Switch.

  - Fixed the issue of operating unknown pointer risk.
    - when used the pointer “handle->mbFrameBuf[mbIdx]” to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
  - Added assert to check current CAN clock source affected by other clock gates in current device.
    - In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).

- Improvements
  - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
  - Optimized the function of interruption handler.
  - Added two APIs for FLEXCAN EDMA driver.
    - FLEXCAN_PrepareTransfConfiguration
    - FLEXCAN_StartTransferDatafromRxFIFO
  - Added new API for FLEXCAN driver.
    - FLEXCAN_GetTimeStamp
      - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
    - FLEXCAN_EnterFreezeMode
    - FLEXCAN_ExitFreezeMode
  - Added new configuration for user.
    - disableSelfReception
    - enableListenOnlyMode
  - Renamed the two clock source enum macros based on CLKSRC bit field value directly.
    - The CLKSRC bit value had no property about Oscillator or Peripheral type in lots of devices, it acted as two different clock input source only, but the legacy enum macros name contained such property, that misled user to select incorrect CAN clock source.
  - Created two new enum macros for the FLEXCAN driver.
    - kFLEXCAN_ClkSrc0
    - kFLEXCAN_ClkSrc1
  - Deprecated two legacy enum macros for the FLEXCAN driver.
    - kFLEXCAN_ClkSrcOsc
    - kFLEXCAN_ClkSrcPeri
  - Changed the process flow for Remote request frame response..
    - Created a new enum macro for the FLEXCAN driver.
      - kStatus_FLEXCAN_RxRemote

  - Changed the process flow for kFLEXCAN_StateRxRemote state in the interrupt handler.
    * Should the TX frame not register to the pointer of frame handle, interrupt handler would not be able to read the remote response frame from the mail box to ram, so user should read the frame by manual from mail box after a complete remote frame transfer.

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.0]

- Initial version.

FLEXCAN_EDMA

[2.12.1]

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

[2.12.0]

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

[2.11.7]

- Refer FLEXCAN driver change log 2.7.0 to 2.11.7

FLEXIO

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
  - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

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

[2.1.0]

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

[2.0.4]

- Bug Fixes
  - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.2]

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

[2.0.1]

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

FLEXIO_A-FORMAT

[1.0.0]

- New Features
  - The polling mode was added to read or configure encoder data
  - The interrupt mode was added to read or configure encoder data

---

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_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_T-FORMAT

[1.0.0]

- New Features
  - The polling mode was added to read or configure encoder data
  - The interrupt mode was added to read or configure encoder data

---

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

FLEXSPI

[2.8.1]

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

[2.8.0]

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

[2.7.0]

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

[2.6.4]

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

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.5]

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

[2.3.4]

- Bug Fixes
  - Updated flexspi_config_t structure and FlexSPI_Init to support new feature FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_CONBINATION.

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

- Bug Fixes
  - Fixed AHB receive buffer size configuration issue. The FLEXSPI_AHBRXBUFCR0_BUFSZ field should configure 64 bits size, and currently the AHB receive buffer size is in bytes which means 8-bit, so the correct configuration should be config->ahbConfig.buffer[i].bufferSize / 8.

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

FLEXSPI EDMA Driver

[2.3.3]

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

[2.3.2]

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

[2.0.2]

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

[2.0.0]

- Initial version.

FLEXSPI_FLR

[2.0.0]

- Initial version.

GPC

[2.1.0]

- Improvements
  - Added GPC_SS_SoftwareTriggerPMICStandby() function.
  - Added GPC_SS_SystemSleepTriggerPMICStandby() function.

[2.0.0]

- Initial version.

GPT

[2.0.6]

- Bug Fixes
  - Fix CERT INT30-C issues.

[2.0.5]

- Improvements
  - Support workaround for ERR003777. This workaround helps switching the clock sources.

[2.0.4]

- Bug Fixes
  - Fixed compiler warning when built with FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL flag enabled.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 5.3 by customizing function parameter.

[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 10.1, 10.3, 10.4, 10.6, 10.8, 17.7.

[2.0.0]

- Initial version.

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.

IEE

[2.1.1]

- Fixed MISRA issues.

[2.1.0]

- Add region lock function IEE_LockRegionConfig() and driver clock control.

[2.0.0]

- Initial version.

IEE_APC

[2.0.2]

- Updated to newer version of implementation in HW.

[2.0.1]

- Fixed MISRA issues.

[2.0.0]

- Initial version.

IOMUXC

[2.0.1]

- Doxygen improvement.

[2.0.0]

- Initial version.

KPP

[2.1.1]

- Bug Fixes
  - Fixed coverity MSG issues with CERT INT30-C, CERT ARR30-C compliance.

[2.1.0]

- Improvements
  - Optimize rowData debounce method to adapt to multi-key detection
  - Modify the KPP_keyPressScanning type to status_t.

[2.0.1]

- Bug Fixes
  - Fixed the violations of MISRA 2012 rules:
    * Rule 10.3 10.4 10.6 14.4 17.7

[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.

LPI2C

[2.6.2]

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

[2.6.1]

- Bug Fixes
  - Fixed coverity issues.

[2.6.0]

- New Feature
  - Added common IRQ handler entry LPI2C_DriverIRQHandler.

[2.5.7]

- Improvements
  - Added support for separated IRQ handlers.

[2.5.6]

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

[2.5.5]

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

[2.5.4]

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

[2.5.3]

- Improvements
  - Added handler for LPI2C7 and LPI2C8.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
  - Updated LPI2C_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes
  - Fixed bug in LPI2C_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
  - Fixed bug in LPI2C_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.

  - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
  - Fixed bug in the usage of LPI2C_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.
  - Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
  - Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

[2.3.0]

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

[2.2.0]

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

[2.1.12]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.1.11]

- Bug Fixes
  - Fixed the bug that, during master non-blocking transfer, after the last byte is sent/received, the kLPI2C_MasterNackDetectFlag is expected, so master should not check and clear kLPI2C_MasterNackDetectFlag when remainingBytes is zero, in case FIFO is emptied when stop command has not been sent yet.
  - Fixed the bug that, during non-blocking transfer slave may nack master while master is busy filling tx FIFO, and NDF may not be handled properly.

[2.1.10]

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

[2.1.9]

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

[2.1.8]

- Bug Fixes
  - Fixed the transfer issue with LPI2C_MasterTransferNonBlocking, kLPI2C_TransferNoStopFlag, with the wait transfer done through callback in a way of not doing a blocking transfer.

  - Fixed the issue that STOP signal did not appear in the bus when NAK event occurred.

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

LPI2C_EDMA

[2.4.5]

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

[2.4.4]

- Improvements
  - Added support for 2KB data transfer

[2.4.3]

- Improvements
  - Added support for separated IRQ handlers.

[2.4.2]

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

[2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1

LPIT

[2.1.3]

- Bug Fixes
  - Fixed doxygen generation warnings.

[2.1.2]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.1.1]

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

[2.1.0]

- Improvements
  - Add new function LPIT_SetTimerValue to set timeout period.

[2.0.2]

- Improvements
  - Improved LPIT_SetTimerPeriod implementation, configure timeout value with LPIT ticks minus 1 generate more correct interval.
  - Added timeout value configuration check for LPIT_SetTimerPeriod, at least input 3 ticks for calling LPIT_SetTimerPeriod.
- Bug Fixes
  - Fixed MISRA C-2012 rule 17.7 violations.

[2.0.1]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.

[2.0.0]

- Initial version.

LPSPI

[2.7.3]

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

[2.7.2]

- Bug Fixes
  - Fixed coverity issues.

[2.7.1]

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

[2.7.0]

- New Feature
  - Added common IRQ handler entry LPSPI_DriverIRQHandler.

[2.6.10]

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

[2.6.9]

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

[2.6.8]

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

[2.6.7]

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

[2.6.6]

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

[2.6.5]

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

[2.6.4]

- Bug Fixes
  - Added LPSPI6_DriverIRQHandler for LPSPI6 instance.

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

- Bug Fixes
  - Fixed bug in LPSPI_SetPCSContinous when disabling PCS continous mode.

[2.2.0]

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

[2.1.0]

- Improvements
  - Improved LPSPI_SlaveTransferHandleIRQ to fill up TX FIFO instead of write one data to TX register which improves the slave transmit performance.
  - Added new functional APIs LPSPI_SelectTransferPCS and LPSPI_SetPCSContinous to support changing PCS selection and PCS continous mode.
- Bug Fixes

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed the bug that the clock source should be divided by the PRESCALE setting in LPSPI_MasterSetDelayTimes function.

  - Fixed the bug that LPSPI_MasterTransferBlocking function would hang in some corner cases.
- Optimization
  - Added #ifndef/#endif to allow user to change the default TX value at compile time.

[2.0.0]

- Initial version.

LPSPI_EDMA

[2.4.9]

- Improvements
  - Removed unused code from LPSPI_SeparateEdmaReadData().

[2.4.8]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

- Improvements
  - Add the TXMSK wait after TCR setting.

[2.4.0]

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

LPTMR

[2.2.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPUART

[2.10.0]

- New Feature
  - Added support to configure RTS watermark.

[2.9.4]

- Improvements
  - Merged duplicate code.

[2.9.3]

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

[2.9.2]

- Bug Fixes
  - Fixed coverity issues.

[2.9.1]

- Bug Fixes
  - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.7]

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

[2.7.6]

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

[2.7.5]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Bug Fixes
  - Added missing interrupt enable masks kLPUART_Match1InterruptEnable and kLPUART_Match2InterruptEnable.
  - Fixed bug in LPUART_EnableInterrupts, LPUART_DisableInterrupts and LPUART_GetEnabledInterrupts that the BAUD[LBKDIE] bit field should be soc specific.

  - Fixed bug in LPUART_TransferHandleIRQ that idle line interrupt should be disabled when rx data size is zero.
  - Deleted unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag, since firstly their function are the same as kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag, secondly to obtain them one data word must be read out thus interfering with the receiving process.
  - Fixed bug in LPUART_GetStatusFlags that the STAT[LBKDIF], STAT[MA1F] and STAT[MA2F] should be soc specific.
  - Fixed bug in LPUART_ClearStatusFlags that tx/rx FIFO is reset by mistake when clearing flags.
  - Fixed bug in LPUART_TransferHandleIRQ that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
  - Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
  - Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
  - Added APIs LPUART_GetRxFifoCount/LPUART_GetTxFifoCount to get rx/tx FIFO data count.
  - Added APIs LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
  - Fixed MISRA advisory 17.7 issues.

[2.4.0]

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

[2.3.1]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.3.0]

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

[2.2.8]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule-10.3, rule-14.4, rule-15.5.
  - Eliminated Pa082 warnings by assigning volatile variables to local variables and using local variables instead.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.8, 14.4, 11.6, 17.7.
- Improvements
  - Added check for kLPUART_TransmissionCompleteFlag in LPUART_WriteBlocking, LPUART_TransferHandleIRQ, LPUART_TransferSendDMACallback and LPUART_SendEDMACallback to ensure all the data would be sent out to bus.
  - Rounded up the calculated sbr value in LPUART_SetBaudRate and LPUART_Init to achieve more acurate baudrate setting. Changed osr from uint32_t to uint8_t since osr's bigest value is 31.
  - Modified LPUART_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

- Improvements
  - Added hardware flow control function support.
  - Added idle-line-detecting feature in LPUART_TransferNonBlocking function. If an idle line is detected, a callback is triggered with status kStatus_LPUART_IdleLineDetected returned. This feature may be useful when the received Bytes is less than the expected received data size. Before triggering the callback, data in the FIFO (if has FIFO) is read out, and no interrupt will be disabled, except for that the receive data size reaches 0.

  - Enabled the RX FIFO watermark function. With the idle-line-detecting feature enabled, users can set the watermark value to whatever you want (should be less than the RX FIFO size). Data is received and a callback will be triggered when data receive ends.

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - Update transactional APIs.

LPUART_EDMA

[2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0

MECC

[2.1.1]

- Bug fixes:
  - Add volatile to variable counter to fix armgcc 13.2.1 -Os optimization issue.

[2.1.0]

- Bug fixes:
  - Removed Ocram1StartAddress, Ocram1EndAddress, Ocram2StartAddress, Ocram2EndAddress in mecc_config_t structure. Use startAddress and endAddress as instead.
  - Removed static function MECC_GetInstance().
- New Features:
  - Added new function MECC_GetPendingFlags().
  - Added new members: enableReadDataWait, enableReadAddrPipeline, enableWriteDataPipeline, enableWriteAddrPipeline in mecc_config_t structure to support pipeline features.

[2.0.2]

- Bug fixes:
  - Fixed MISRA 2012 issue: 10.3, 10.4.

[2.0.1]

- Bug fixes:
  - Fixed MISRA 2012 issue: 10.1, 10.3, 10.4, 10.6.

[2.0.0]

- Initial version.

MEMORY

[2.1.0]

- Bug Fixes
  - Provide correct address translation based on TCM_SIZE config.

[2.0.0]

- Initial version.

MSGINTR

[2.0.2]

- Improvements
  - Conditional compile IRQ handlers.

[2.0.1]

- Bug Fixes
  - Fixed MISRA issue rule 8.4, 11.9, 17.7.

[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.

NETC

[2.10.3]

- New Features
  - Supported rrt member in netc_tb_ipf_cfge_t.
  - Moved timer reference clock definition to soc to support i.MX95/i.MX943.
- Bug Fixes
  - Got right size of TX timestamp response frame.

[2.10.2]

- Bug Fixes
  - Correct transmit start time bit width of transmit buffer descriptor format.

[2.10.1]

- New Features
  - Supported IP version class commands in VSI-PSI message driver.
  - Added VSI3 definition.

[2.10.0]

- Bug Fixes
  - Fixed structures used for time gate control list and stream gate request commands memset.
- New Features
  - Added public API to query table entry in Frame Modification Table.
- Improvements
  - Optimized NETC Transmit functions: cached configuration in driver handle instead of accessing directly device registers at each packet transmit, and removed calls to memset/memcpy in the transmit path.

[2.9.1]

- Improvements
  - NETC_TimerInit() will always ignore user config->atomicMode and set it to 1 internally. This guarantees that period updates, that change both the integer and fractional part are always done atomically.

[2.9.0]

- Bug Fixes
  - Fixed padding in netc_tb_sgi_rsp_data_t union structure for query operations on OEXEN and IRXEN parameters.
  - Fixed structure use for rate policer and stream gate request commands memset.
  - Updated ERRATA 052134 to 052206.
  - Fixed MII mode setting.
  - Fixed i.MX943 getting function instance.
- New Features
  - Added API to Reset IRX and OEX flags in stream gate instance entry.
  - Added API to configure the priority to traffic class map.
  - Added APIs to query table entry and get maximum entry number for Frame Modification Table.
  - Added APIs to configure Frame preemption.
  - Added APIs to configure PSRCR and PGCR registers to implement HSR feature.
  - Moved PHY WRAPPER init sequence (NETC_PHYInit) implementation to SoC. And Added i.MX943 support.
- Improvements
  - keep netc_tb_sgi_rsp_data_t local to the low level driver functions for SGI table entry query.

  - Converted to use preinitVsi callback for VSI pre-init.
  - Added note for ERRATA 052167 to remind that actual MAC Tx IPG is longer than configured when transmitting back-to-back packets in MII half duplex. When using MII protocol, using full-duplex mode is recommended instead of half-duplex. If using MII half-duplex mode, additional bandwidth loss should be expected and accounted for due to extended IPG.

[2.8.2]

- Bug Fixes
  - Fixed ingress port filter table frame attribute flags mask field issue.

[2.8.1]

- Bug Fixes
  - Fixed MAC/VLAN filter operations through VSI-PSI message.
  - Fixed NETC_PortConfigTxIpgPreamble compile.
- Improvements
  - Enabled standard VLAN EtherTypes for i.MX95 VSIs for VLAN support.
  - Added netc_timer_exttrig_index_t definition for i.MX95.
  - Updated default BPCR[STAMVD] value setting to align the register reset value.

[2.8.0]

- Bug Fixes
  - Fixed ERRATA 052024.
  - Fixed ERRATA 052129.
  - Fixed ERRATA 052134.
  - Fixed ERRATA 052031.
  - Fixed ERRATA 051994.
  - Fixed ERRATA 051936.
- New Features
  - Added interface to reset the mark frame red parameter.
  - Added support for FRER sequence generation reset.
  - Added NETC Switch Tag support.
  - Added the Tx offload feature support.
- Improvements
  - Simplify NETC_TimerGetFreeRunningTime. Hardware synchronizes reads from high/low registers for the free running time. No need to do it in software.

[2.7.2]

- Bug Fixes
  - Fixed MISRA issue rule 4.10, 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 16.1, 16.4, 17.7.

[2.7.1]

- Bug Fixes
  - Fixed Coverity issue with array out of bounds access.

[2.7.0]

- New Features
  - Added VSI-PSI messaging driver.
- Bug Fixes
  - Fixed the issue that EP/SWT_ReceiveFrame don't return error status when some errors occur.

[2.6.1]

- Bug Fixes
  - Updated the MAC loopback configuration as Reference Manual.

[2.6.0]

- New Features
  - Added API to get transmit max SDU for specified port Traffic Class.
  - Added API to query entry from the Rate Policing table.
  - Added API to retrieve maximum rate policer entries.
  - Added API to set switch port default VID separately.
  - Added API to set max frame size separately.
  - Added API to query FRER resource.
- Bug Fixes
  - Fixed the issue that stream gate query functions don't check return status.
  - Fixed the ISF table query function operation issue.
  - Fixed the wrong configuration of Tx max SDU check.
  - Fixed ERRATA 051524.
  - Fixed ERRATA 051649.
  - Fixed ERRATA 051707.
  - Fixed ERRATA 051710.
  - Fixed ERRATA 051711.
- Improvements
  - Factorized qbv basetime workaround code, and stop using synchronized time for the workaround code. Synchronized time functionality should be reserved for gPTP operation.

[2.5.1]

- Improvements
  - Conditional compile NETC_ETH_LINK_PM0_COMMAND_CONFIG_HD_FCEN register.

[2.5.0]

- New Features
  - Added PHY WRAPPER driver.
  - Added C45 support for internal MDIO.
  - Added 10G support.
- Bug Fixes
  - Fixed ERRATA 051130.
  - Fixed master bus and memory access.
- Improvements
  - Moved platform specific code to soc driver.
  - Split switch code.

[2.4.0]

- New Features
  - Added the interrupt control functions for port MAC module.
  - Added setting parameters including half-duplex back pressure, port timestamp capture point, RGMII Tx clock stop state during low power idle, ports default traffic class gating states and timer atomic writing setting.
  - Added NETC_TimerInitHandle() to initialize a timer handle without modifying hardware state. Required to be able to read timer from another CPU.
  - Added NETC_TimerGetFreeRunningTime() to be able to read free running timer.
  - Added support for ingress stream gate query.
- Improvements
  - Added necessary default settings in the GetDefaultConfig functions in case some features can't work after initialization.
  - Updated loopback function according to new bit field in CRR.
  - Deleted the useless error check for ERRATA051243.
  - Updated NETC_TimerGetCurrentTime() to avoid using synchronized time and be able to read the time from different threads/cpus without locking.
  - Deleted the useless priority check in NETC_PortConfigTcCBS().
- Bug Fixes
  - Fixed typo in NETC_PortConfig.

[2.3.2]

- Bug Fixes
  - Added workaround for ERRATA051587.

[2.3.1]

- Bug Fixes
  - Fixed MISRA issue rule 10.3, 10.4, 10.8, 11.6, 11.7.

[2.3.0]

- Bug Fixes
  - Added SWT_PortStop() API for ERRATA051398.
  - Fixed the build error by add feature macro for port FCS Error Action feature.
  - Removed duplicate code from NETC_PortEthMacGracefulStop() API.
  - Fixed MISRA issue rule 8.6, 10.4, 11.9, 14.4.

[2.2.2]

- Bug Fixes
  - Fixed the issue that NETC_PortSetSpeed() would overwrite the full PCR register.

[2.2.1]

- Improvements
  - Fixed cpp build warning.

[2.2.0]

- Bug Fixes
  - Fixed the issue that NETC_ConfigTGSAdminList() doesn't clear the previous command response data status filed.
  - Fixed the issue that EP_ReceiveFrameCopy(&handle, 0, NULL, 0, NULL) can't drop error frame.
  - Fixed the issue that SWT_GetTimestampRefResp can't get Switch Tx TS Resp with no MgmtRxBdRing.
  - Fixed the issue that RGMII Half Duplex mode misconfigured.
  - Fixed the issue that missing workaround for ERR050679, ERR051246 and ERR051254.
  - Fixed the issue that missing feature macro for ERR051130, ERR051202, ERR051260.
  - Fixed the issue that ep/swt_tx_opt struct use wrong vlan tag tpid value.
  - Fixed MISRA issue rule 5.8, 8.3, 8.12, 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 11.6, 11.8, 12.2, 14.4, 15.5, 15.6, 16.1, 16.3, 16.4, 17.7.
  - Fixed the issue that internal MDIO read function uses wrong register.
  - Fixed the issue that SWT_TxPortTGSEnable()/EP_TxPortTGSEnable() still uses the default timer after enabling the 1588 timer.
  - Remove the resetCount parameter from get port discard statistic APIs because the registers required by this function have been removed from hardware design.
  - Fixed the issue that SWT/EP_ReclaimTxDesc() can't call reclaim callback for each full frame.
  - Fixed the issue in NETC_TimerAdjustFreq().
- New Features
  - Added the support for 1588 One-Step timestamp when chip doesn't have ERR051255.
  - Added APIs to get dynamic table remaining available entry numbers.
  - Added APIs to get static table number of entries.

- Improvements
  - Return detail error status instead of kStatus_Fail in NTMP APIs.
  - Rename feature macros and move them into the feature file.
  - Optimize the implementation of the NETC_TimerAddOffset() function to avoid change the TMR_CNT_L/H registers, and add required procedure for call NETC_TimerAddOffset() API in the comments.
  - Update the SWT_FMDUpdateTableEntry()/SWT_FMDQueryTableEntry() APIs to make them use internel table buffer.
  - Update SWT_TxPortTGSEnable()/EP_TxPortTGSEnable() to make it can config the default administration gate control list gates' state.
  - Use TMR_SRT_L/H instead of TMR_CUR_TIMER when want to get current 1588 timer value.

[2.1.0]

- Bug Fixes
  - Fixed the issue that EP_RxL2MFInit doesn't set the multicast promiscuous correctly.
  - Fixed the timer add offset issue.
  - Fixed the issue that all ENETC/Switch PCIe functions must be enabled firstly before triggering EP/SWT NTMP access and MSIX messages.
  - Fixed RT1180 NETC errata 051202: Configure Tx MAC to wait until 32 bytes of data are built up in the transmit FIFO before beginning transmission on the link.
  - Added workaround for RT1180 NETC errata 051130: C Egress time gate scheduling can get corrupted when functional level reset is applied or when time gating is disabled.
  - Fixed bugs in statistic APIs.
- New Features
  - Added the support for EP VSI transmission and PSI-VSI message exchanging.
  - Added EP receive regular frame zero-copy support.
  - Integrated EMDIO support in NETC MDIO driver for accessing PHY when EP/Switch function isn't enabled.
  - Added Timer and Switch MSIX table configuration support.
  - Added update entry APIs for IPF/VF/FDB/L2MCF/IS/ISF/SGI/RP/FM/ET/ISEQG tables, and search entry APIs for FDB/L2MCF table.
  - Added Ingress buffer pool table config APIs.
  - Added MAC Tx padding and Rx min/max frame size configuration to support Tx/Rx frames smaller than 64 bytes.
  - Added API to do graceful Stop for ETH MAC.
- Improvements
  - Used Rx buffer address array provided by application instead of buffer start address with contiguous memory to make the Rx buffer setup more flexible.
  - Added ring and userData parameter in the Tx reclaim callback.
  - Updated NETC hardware layer folder name from 'hw' to 'netc_hw'.
  - Stored necessary EP and SWT configurations constant in handle structure instead of storing pointer which forces appliction to keep static configuration structure data.

  - Updated NETC_MsixXxx to EP_MsixXxx to differentiate with corresponding SWT/Timer MSIX configuration APIs.
  - Aligned TGSL/SGCL API with enet_qos high-level driver.
  - Updated EP/Switch config structure to include all port related config.
  - Updated Switch transfer API only send management frame (Direct enqueue and Switch Port Masquerading) and only receive Host Reason no-zero frames.
  - Updated EP transfer API only send/receive regular frames.
  - Updated Switch/EP handle to make them use independent cache maintain, alloc/free memory and reclaimCallback functions.

[2.0.0]

- Initial version.

OTFAD

[2.1.4]

- Bug fixes
  - Fixed MISRA 2012 issue: 10.1.

[2.1.3]

- Bug fixes
  - Fixed the error that waiting for both FLEXSPI AHB idle and SEQ idle.

[2.1.2]

- Bug fixes
  - Fixed MISRA 2012 issue: 10.4.

[2.1.1]

- Improvements:
  - Hided some bits in CR and SR registers for selected platforms.
  - Fixed doxygen issues.

[2.1.0]

- Improvements:
  - Used boolean type to define 1-bit field concepts.

[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

PMU

[2.1.3]

- Bug Fixes
  - Modify PMU_BIAS_CTRL value from 0x11009002 to 0x110081EE.

[2.1.2]

- Bug Fixes
  - Updated PMU_StaticEnablePllLdo() with disabling LDO current limit after LDO is stable to minimize ARM PLL jitter in cold temperature.

[2.1.1]

- Bug Fixes
  - Fixed bugs in FBB configuration.

[2.1.0]

- Improvements
  - Added PMU_EnableFBB() function.
  - Added PMU_EnableFBBStandbyMode() function.
  - Added PMU_EnableLdoStandbyMode() function.

[2.0.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 8.4.

[2.0.0]

- Initial version.

PWM

[2.9.1]

- Improvements
  - Add new API PWM_SetupFaultsExt and PWM_SetupFaultInputFilterExt to support FlexPWM which has more than one fault input channels.
  - Support fault 4-7 interrupt and its flag.
- Bug Fixes
  - Fixed violations of the CERT INT31-C.

[2.9.0]

- Improvements
  - Support PWMX channel output for edge aligned PWM.
  - Forbid submodule 0 counter initialize with master sync and master reload mode.
  - Clarify kPWM_BusClock meaning.
  - Verify pulseCnt within 65535 when update period register.

[2.8.4]

- Improvements
  - Support workaround for ERR051989. This function helps realize no phase delay between submodule 0 and other submodule.

[2.8.3]

- Bug Fixes
  - Fixed MISRA C-2012 Rule 15.7

[2.8.2]

- Bug Fixes
  - Fixed warning conversion from 'int' to 'uint16_t' on API PWM_Init.
  - Fixed warning unused variable 'reg' on API PWM_SetPwmForceOutputToZero.

[2.8.1]

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

[2.8.0]

- Improvements
  - Added API PWM_UpdatePwmPeriodAndDutycycle to update the PWM signal's period and dutycycle for a PWM submodule.
  - Added API PWM_SetPeriodRegister and PWM_SetDutycycleRegister to merge duplicate code in API PWM_SetupPwm, PWM_UpdatePwmDutycycleHighAccuracy and PWM_UpdatePwmPeriodAndDutycycle

[2.7.1]

- Improvements
  - Supported UPDATE_MASK bit in MASK register.

[2.7.0]

- Improvements
  - Supported platforms which don't have Capture feature with channel A and B.
  - Supported platforms which don't have Submodule 3.
  - Added assert function in API PWM_SetPhaseDelay to prevent wrong argument.

[2.6.1]

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

[2.6.0]

- Improvements
  - Added API PWM_SetPhaseDelay to set the phase delay from the master sync signal of submodule 0.
  - Added API PWM_SetFilterSampleCountthe to set number of consecutive samples that must agree prior to the input filter.
  - Added API PWM_SetFilterSamplePeriod to set set the sampling period of the fault pin input filter.

[2.5.1]

- Bug Fixes
  - Fixed MISRA C-2012 rules: 10.1, 10.3, 10.4 , 10.6 and 10.8.
  - Fixed the issue that PWM_UpdatePwmDutycycle() can't update duty cycle status value correct.

[2.5.0]

- Improvements
  - Added API PWM_SetOouputToIdle to set pwm channel output to idle.
  - Added API PWM_GetPwmChannelState to get the pwm channel output duty cycle value.
  - Added API PWM_SetPwmForceOutputToZero to set the pwm channel output to zero logic.
  - Added API PWM_SetChannelOutput to set the pwm channel output state.
  - Added API PWM_SetClockMode to set the value of the clock prescaler.
  - Added API PWM_SetupPwmPhaseShift to set PWM which a special phase shift and 50% duty cycle.
  - Added API PWM_SetVALxValue/PWM_GetVALxValue to set/get PWM VALs registers values directly.

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

- Improvements
  - Moved the configuration of fault input filter into a new API to avoid be initialized multiple times.
- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fix rules, containing: rule-10.2, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-14.4, rule-16.4.

[2.0.1]

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

[2.0.0]

- Initial version.

QTMR

[2.3.0]

- Improvements
  - Support for platforms which QTMR registers are 32-bit.

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
  - Added API QTMR_SetPwmOutputToIdle to set the generated pwm signal to the configured idle value.
  - Added API QTMR_GetPwmOutputStatus to return the output status of the generated pwm signal.
  - Added API QTMR_GetPwmChannelStatus to return the channel dutycycle value.
  - Added API QTMR_SetPwmClockMode to set clock mode change peripheral clock frequency.
- Bug Fixes
  - Fixed the issue that pwm duty cycle could not be 0 and 100.

[2.1.0]

- Bug Fixes
  - Fixed the issue QTMR_SetTimerPeriod needs to decrement down count by 1, and added new APIs to configure the LOAD register, COMP register.

[2.0.2]

- Bug Fixes
  - Fixed the issue introduced by previous code correction for improving the output signal accuracy.

[2.0.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.3, 11.5, 11.9.
- Improvements
  - Improved the output signal accuracy.

[2.0.0]

- Initial version.

RGPIO

[2.1.0]

- New feature:
  - Added API RGPIO_EnablePortInput()
  - Added API RGPIO_SetPinInterruptConfig()
  - Added API RGPIO_GetPinsInterruptFlags()
  - Added API RGPIO_ClearPinsInterruptFlags()

[2.0.3]

- Improvements:
  - Enhanced FGPIO_PinInit to enable clock internally.

[2.0.2]

- Bug fix
  - MISRA C-2012 issue fixed.
    - Fix rules, containing: rule-10.3, rule-14.4, rule-15.5.

[2.0.1]

- API Interface Change:
  - Refined naming of API while keep all original APIs with marking them as deprecated. The original API will be removed in the next release. The main change is to update API with prefix of _PinXXX() and _PortXXX().

[2.0.0]

- Initial version.

ROMAPI

[1.1.2]

- Improvements
- Fix Misra issues

[1.1.1]

- Improvements
  - Update the comments of “clear cache” function.

[1.1.0]

- New features
  - Support B0 silicon

[1.0.0]

- initial version.

RTWDOG

[2.1.4]

- Bug Fixes
  - Fixed CERT INT30-C, INT31-C issue.
  - Make API RTWDOG_CountToMesec return 0 if result overflow.

[2.1.3]

- Improvements
  - Waited the over status after CS register operation in case next CS operation causes problem.

[2.1.2]

- Bug Fixes
  - Fixed doxygen issue.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-10.3, rule-10.8, rule-11.9, rule-14.4, rule-15.5.

[2.1.0]

- Improvements
  - Added an API to enable or disable the window mode.
  - Added an API to convert a raw count value to millisecond.
  - Used AT_QUICKACCESS_SECTION_CODE macro to decorate RTWDOG_Init, and copied this function from flash to QUICKACCESS section.

[2.0.1]

- Bug Fixes
  - Fixed bug in the RTWDOG_Init; added check for register's unlock status when configuring the RTWDOG in RTWDOG_init.

[2.0.0]

- Initial version.

S3MU

- 2.0.2 Fix macro BIT redefined warning when compiling with Zephyr.
- 2.0.1 Update kStatusGroup_SNT to kStatusGroup_ELEMU.
- 2.0.0 Initial version of S3MU driver.

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

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.

SEMC

[2.7.1]

- Bug Fixes
  - Fixed the wrong write operation to INTR register. The INTR register is a W1C register, so the right write operation is write directly to it to clear.

[2.7.0]

- Improvements
  - Add new autofreshTimes parameter in semc_sdram_config_t.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.6.0]

- Bug Fixes
  - Fixed the SEMC SRAM function bug that some configuration options can't be set.
  - Correct legacy SEMC SRAM function feature macros.
- Improvements
  - Add new SEMC SRAM function feature macros.

[2.5.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 14.3.
  - Fixed SEMC_ConfigureDBI bug that RDX not set correctly.

[2.5.0]

- Bug Fixes
  - Fixed definitions of bitfields of BMCR0 and BMCR1 - wrong field order and incorrect semantical naming
  - The fix alters the driver API regarding configuration of AXI bus queue reordering

[2.4.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 5.6.

[2.4.2]

- Improvements
  - Deleted meaningless parameter in memory size conversion function.

[2.4.1]

- Bug Fixes
  - Fixed PSRAM A8 configuration issue, which should be 0x06U for PSRAM while pix mux bit width is 0x04U, based on different pix mux bit width.

[2.4.0]

- Improvements
  - Improved nor and sram timing configuration on sync mode.

[2.3.1]

- Bug Fixes
  - Updated refresh timer period(RT) timing setting, which updated into (RT+1)*(Prescaler period) for SDRAM.
  - Supported new DBI control register 2 to configure CSX interval time(CEITV).
  - Fixed violations of the MISRA C-2012 Rule 10.8.
  - Fixed doxygen warning.

[2.3.0]

- New Features
  - Limited burst length as 1 according to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1.
  - Supported 8 bits column address for SDRAM.

[2.2.1]

- New Features
  - Added queue weight control, which can control queue a/b is working or not.
  - Updated NAND FLASH configuration API which disables and enables SEMC between configure control registers.
  - Added ONFI parameter Integrity CRC check for SEMC flash component.

[2.2.0]

- New Features
  - Supported up to 4 PSRAM CS.
  - Added programmable delay line for DQS.
  - Added ready/wait feature for SRAM in asynchronous mode.

[2.1.0]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.
  - Updated parameter type from uint16_t into uint32_t for send IP command API.

[2.0.4]

- Bug Fixes
  - Fixed the SEMC queueA and queueB weight configuration issue.
  - Fixed the wrong configuration of DBICR1 register in SEMC_ConfigureDBI.

[2.0.3]

- Bug Fixes
  - Added feature macro to control WDS&WDH bit setting for NOR synchronous transfer.

[2.0.2]

- Bug Fixes
  - Changed SEMC NAND configuration structure and verify SEMC NAND related APIs.
  - Added extended SEMC clock enable.

[2.0.1]

- Bug Fixes
  - Fixed data size mask configure in SEMC_ConfigureIPCommand API.
  - Updated the command mode in IP command type.

[2.0.0]

- Initial version.

SINC

[2.1.5]

- Bug Fixes
  - Fixed building warning.

[2.1.4]

- Bug Fixes
  - Fixed building issue.

[2.1.3]

- Bug Fixes
  - Fixed function 'SINC_SetChannelProtectionOption' logic operation error.

[2.1.2]

- Bug Fixes
  - Fixed the typo issue of missing character 'U' in the feature macro 'FSL_FEATRE_SINC_CACFR_HAS_NO_PTMUX'.

[2.1.1]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.4 and 10.8 issues.

[2.1.0]

- Improvements
  - Added support for chips that each instance equipped with 5 channels.

[2.0.2]

- Improvements
  - Added comments for over sample ratio.

[2.0.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 15.6, 10.4, 10.1, 10.3 and 10.7.

[2.0.0]

- Initial version.

SPDIF

[2.0.7]

- Improvements
  - Add feature macro FSL_FEATURE_SPDIF_HAS_NO_SIC_REGISTER to handle nonexistent SIC register.

[2.0.6]

- Bug Fixes
  - Fixed the Q/U channel interrupt enabled unexpectly while Q/U transfer pointer is NULL.

[2.0.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.3.

[2.0.4]

- Bug Fixes
  - Added udata/qdata buffer address validation in driver IRQ handler to ensure that NULL pointer dereferences do not occur.

[2.0.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.

[2.0.2]

- Bug Fixes
  - Corrected operator used for size value assertion in SPDIF_ReadBlocking/SPDIF_WriteBlocking.

[2.0.1]

- Bug Fixes
  - Corrected the feature macro name used to define s_edmaPrivateHandle.

[2.0.0]

- Initial version.

SPDIF DMA Driver

[2.0.8]

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

[2.0.7]

- Bug Fixes
  - Fixed the incompatibility issue with edma4 driver.

[2.0.6]

- Bug Fixes
  - Add feature macro to determine whether to use the API MEMORY_ConvertMemoryMapAddress to translate TCD addresses for DLAST_SGA.

[2.0.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.3.

[2.0.4]

- Bug Fixes
  - Added udata/qdata buffer address validation in driver IRQ handler to ensure that NULL pointer dereferences do not occur.

[2.0.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.

[2.0.2]

- Bug Fixes
  - Corrected operator used for size value assertion in SPDIF_ReadBlocking/SPDIF_WriteBlocking.

[2.0.1]

- Bug Fixes
  - Corrected the feature macro name used to define s_edmaPrivateHandle.

[2.0.0]

- Initial version.

SRAMC

[2.0.0]

- Initial version.

SYSCTR

[2.0.1]

- Bug Fixes
  - Fixed MISRA issue 10.4.

[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.

TEMPSENSOR

[2.1.2]

- Bug Fixes
  - Fixed the bug of incorrect default value of temperature sensor registers in initialization state.

[2.1.1]

- Improvements
  - CTRL0 register fields are not needed for customer, they are trim registers for the IP that are determined during calibration.

[2.1.0]

- Improvements
  - Supported directly access to TEMPSENSOR registers.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.8.

[2.0.2]

- Bug Fixes
  - Fixed bug that FINISH flag not cleared after temperature read out.

[2.0.1]

- Improvements
  - Updated temperature calculation formula, to get more accurate result with high or low temperature..

[2.0.0]

- Initial version.

TPM

[2.4.1]

- Improvements
  - Add Coverage Justification for uncovered code.

[2.4.0]

- New Feature
  - Added while loop timeout for MOD CnV CnSC and SC register write sequence.
  - Change the return type from void to status_t for following API:
    - TPM_DisableChannel
    - TPM_EnableChannel
    - TPM_SetupOutputCompare
    - TPM_SetTimerPeriod
    - TPM_StopTimer

[2.3.6]

- Bug Fixes
  - Fixed CERT INT30-C INT31-C issue for TPM_SetupDualEdgeCapture.

[2.3.5]

- New Feature
  - Added IRQ handler entry for TPM2.

[2.3.4]

- New Feature
  - Added common IRQ handler entry TPM_DriverIRQHandler.

[2.3.3]

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

[2.3.2]

- Bug Fixes
  - Fixed ERR008085 TPM writing the TPMx_MOD or TPMx_CnV registers more than once may fail when the timer is disabled.

[2.3.1]

- Bug Fixes
  - Fixed compilation error when macro FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is 1.

[2.3.0]

- Improvements
  - Create callback feature for TPM match and timer overflow interrupts.

[2.2.4]

- Improvements
  - Add feature macros(FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_EN, FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_SYNC).

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Bug Fixes
  - Fixed CCM issue by splitting function from TPM_SetupPwm() function to reduce function complexity.
  - Fixed violations of MISRA C-2012 rule 17.7.

[2.2.0]

- Improvements
  - Added TPM_SetChannelPolarity to support select channel input/output polarity.
  - Added TPM_EnableChannelExtTrigger to support enable external trigger input to be used by channel.
  - Added TPM_CalculateCounterClkDiv to help calculates the counter clock prescaler.
  - Added TPM_GetChannelValue to support get TPM channel value.
  - Added new TPM configuration.
    - syncGlobalTimeBase
    - extTriggerPolarity
    - chnlPolarity
  - Added new PWM signal configuration.
    - secPauseLevel
- Bug Fixes
  - Fixed TPM_SetupPwm can't configure 0% combined PWM issues.

[2.1.1]

- Improvements
  - Add feature macro for PWM pause level select feature.

[2.1.0]

- Improvements
  - Added TPM_EnableChannel and TPM_DisableChannel APIs.
  - Added new PWM signal configuration.
    - pauseLevel - Support select output level when counter first enabled or paused.
    - enableComplementary - Support enable/disable generate complementary PWM signal.
    - deadTimeValue - Support deadtime insertion for each pair of channels in combined PWM mode.
- Bug Fixes
  - Fixed issues about channel MSnB:MSnA and ELSnB:ELSnA bit fields and CnV register change request acknowledgement. Writes to these bits are ignored when the interval between successive writes is less than the TPM clock period.

[2.0.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.4 ,10.7 and 14.4.

[2.0.7]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4 and 17.7.

[2.0.6]

- Bug Fixes
  - Fixed Out-of-bounds issue.

[2.0.5]

- Bug Fixes
  - Fixed MISRA-2012 rules.
    - Rule 10.6, 10.7

[2.0.4]

- Bug Fixes
  - Fixed ERR050050 in functions TPM_SetupPwm/TPM_UpdatePwmDutycycle. When TPM was configured in EPWM mode as PS = 0, the compare event was missed on the first reload/overflow after writing 1 to the CnV register.

[2.0.3]

- Bug Fixes
  - MISRA-2012 issue fixed.
    - Fixed rules: rule-12.1, rule-17.7, rule-16.3, rule-14.4, rule-1.3, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-10.6, and rule-18.1.

[2.0.2]

- Bug Fixes
  - Fixed issues in functions TPM_SetupPwm/TPM_UpdateChnlEdgeLevelSelect /TPM_SetupInputCapture/TPM_SetupOutputCompare/TPM_SetupDualEdgeCapture, wait acknowledgement when the channel is disabled.

[2.0.1]

- Bug Fixes
  - Fixed TPM_UpdateChnIEdgeLevelSelect ACK wait issue.
  - Fixed the issue that TPM_SetupdualEdgeCapture could not set FILTER register.
  - Fixed TPM_UpdateChnEdgeLevelSelect ACK wait issue.

[2.0.0]

- Initial version.

TRDC

[2.3.1]

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

[2.3.0]

- New Features
  - Added API TRDC_EnableProcessorDomainAssignment to disable/enable DAC.
- Bug Fixes
  - Fixed MISRA violation of missing function declaration.
  - Fixed wrong operation of domain mask in TRDC_MbcNseClearAll and TRDC_MrcDomainNseClear.

[2.2.1]

- Bug Fixes
  - Fixed MISRA violations of rule 10.3.

[2.2.0]

- New Features
  - Added new APIs to support SoC with more than one processor core.

[2.1.1]

- Bug Fixes
  - Fixed MISRA violations of rule 10.3, 10.6, 10.7 and 18.1.

[2.1.0]

- Improvements
  - Modified some of the addressing method according to the device header file's update.
  - Modified flash address configuration structure and default setting.
- New Features
  - Added API to set flash logic window array address.
- Bug Fixes
  - Fixed wrong return value of TRDC_GetFlashLogicalWindowPbase.
  - Fixed bug in TRDC_GetAndClearFirstSpecificDomainError that the error status is cleared by mistake before obtained by software.
  - Fixed MISRA violations of rule 10.3, 10.4, 10.6 and 10.8.

[2.0.0]

- Initial version.

TSTMR

[2.1.0]

- New Features
  - Support configured clock frequency.
  - Add TSTMR_Init and TSTMR_init APIs.
- Improvements
  - Change TSTMR_DelayUs from static inline function to normal function.

[2.0.4]

- Bugfix
  - Fix MISRA C-2012 Rule 10.4 and 14.4 issues.

[2.0.3]

- Bugfix
  - Fix CERT INT30-C that Unsigned integer operation TSTMR_ReadTimeStamp(base) - startTime may wrap.

[2.0.2]

- Improvements
  - Support 24MHz clock source.
- Bugfix
  - Fix MISRA C-2012 Rule 10.4 issue.
  - Read of TSTMR HIGH must follow TSTMR LOW atomically: require masking interrupt around 2 LSB / MSB accesses.

[2.0.1]

- Bugfix
  - Restrict to read with 32-bit accesses only.
  - Restrict that TSTMR LOW read occurs first, followed by the TSTMR HIGH read.

[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.

VREF

[2.4.0]

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

[2.3.0]

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

[2.2.2]

- Bug Fixes
  - Fixed typo in vref_buffer_mode_t enumerator.

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.0]

- Initial version.

XBAR

[2.2.0]

- New Features
  - Support register write protection.

[2.1.2]

- Correct bits of SEL registers

[2.1.1]

- Fix wrong offset of ctrl registers

[2.1.0]

- unify bit fields width for xbar instance index and xbar input/output signal index.

[2.0.4]

- Improvements
  - Rename feature macro name.

[2.0.3]

- Improvements
  - Improved to support 32-bit width peripheral.

[2.0.2]

- Bug Fixes
  - Fixed MISRA C-2012 violations.

[2.0.1]

- Bug Fixes
  - Fixed the xbar instance base offset error.

[2.0.0]

- Initial version.

1.6 Driver API Reference Manual

This section provides a link to the Driver API RM, detailing available drivers and their usage to help you integrate hardware efficiently.

MIMXRT1189_drivers

1.7 Middleware Documentation

Find links to detailed middleware documentation for key components. While not all onboard middleware is covered, this serves as a useful reference for configuration and development.

1.7.1 Multicore

multicore

1.7.2 MCU Boot

mcuboot_opensource

1.7.3 eIQ

eIQ

1.7.4 FreeMASTER

freemaster

1.7.5 AWS IoT

aws_iot

1.7.6 NXP Wi-Fi

wifi-bluetooth-802.15.4

1.7.7 FreeRTOS

FreeRTOS

1.7.8 Wireless EdgeFast Bluetooth PAL

EdgeFast Bluetooth

1.7.9 lwIP

lwip

1.7.10 File systemFatfs

fatfs

Chapter 2

MIMXRT1186

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 AOI: Crossbar AND/OR/INVERT Driver

void AOI_Init(AOI_Type *base)
Initializes an AOI instance for operation.

This function un-gates the AOI clock.

Parameters

- base – AOI peripheral address.

void AOI_Deinit(AOI_Type *base)
Deinitializes an AOI instance for operation.

This function shutdowns AOI module.

Parameters

- base – AOI peripheral address.

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

Gets the Boolean evaluation associated.

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;

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

Parameters

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

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

Configures an AOI event.

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

Example:

```
aoi_event_config_t demoEventLogicStruct;

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

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

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

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

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

Parameters

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

FSL_AOI_DRIVER_VERSION

Version 2.0.2.

enum _aoi_input_config

AOI input configurations.

The selection item represents the Boolean evaluations.

Values:

enumerator kAOI_LogicZero

Forces the input to logical zero.

enumerator kAOI_InputSignal

Passes the input signal.

enumerator kAOI_InvInputSignal

Inverts the input signal.

enumerator kAOI_LogicOne

Forces the input to logical one.

enum _aoi_event

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

Values:

enumerator kAOI_Event0

Event 0 index

enumerator kAOI_Event1

Event 1 index

enumerator kAOI_Event2

Event 2 index

enumerator kAOI_Event3

Event 3 index

typedef enum *_aoi_input_config* aoi_input_config_t

AOI input configurations.

The selection item represents the Boolean evaluations.

typedef enum *_aoi_event* aoi_event_t

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

typedef struct *_aoi_event_config* aoi_event_config_t

AOI event configuration structure.

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

AOI

AOI peripheral address

struct _aoi_event_config

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

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

Public Members

aoi_input_config_t PT0AC
Product term 0 input A

aoi_input_config_t PT0BC
Product term 0 input B

aoi_input_config_t PT0CC
Product term 0 input C

aoi_input_config_t PT0DC
Product term 0 input D

aoi_input_config_t PT1AC
Product term 1 input A

aoi_input_config_t PT1BC
Product term 1 input B

aoi_input_config_t PT1CC
Product term 1 input C

aoi_input_config_t PT1DC
Product term 1 input D

aoi_input_config_t PT2AC
Product term 2 input A

aoi_input_config_t PT2BC
Product term 2 input B

aoi_input_config_t PT2CC
Product term 2 input C

aoi_input_config_t PT2DC
Product term 2 input D

aoi_input_config_t PT3AC
Product term 3 input A

aoi_input_config_t PT3BC
Product term 3 input B

aoi_input_config_t PT3CC
Product term 3 input C

aoi_input_config_t PT3DC
Product term 3 input D

2.3 ASRC: Asynchronous sample rate converter

2.4 ASRC Driver

uint32_t ASRC_GetInstance(ASRC_Type *base)
Get instance number of the ASRC peripheral.

Parameters

- base – ASRC base pointer.

void ASRC_Init(ASRC_Type *base, uint32_t asrcPeripheralClock_Hz)

brief Initializes the asrc peripheral.

This API gates the asrc clock. The asrc module can't operate unless ASRC_Init is called to enable the clock.

param base asrc base pointer. param asrcPeripheralClock_Hz peripheral clock of ASRC.

void ASRC_Deinit(ASRC_Type *base)

De-initializes the ASRC peripheral.

This API gates the ASRC clock and disable ASRC module. The ASRC module can't operate unless ASRC_Init

Parameters

- base – ASRC base pointer.

void ASRC_SoftwareReset(ASRC_Type *base)

Do software reset .

This software reset bit is self-clear bit, it will generate a software reset signal inside ASRC. After 9 cycles of the ASRC processing clock, this reset process will stop and this bit will cleared automatically.

Parameters

- base – ASRC base pointer

status_t ASRC_SetChannelPairConfig(ASRC_Type *base, *asrc_channel_pair_t* channelPair, *asrc_channel_pair_config_t* *config, uint32_t inputSampleRate, uint32_t outputSampleRate)

ASRC configure channel pair.

Parameters

- base – ASRC base pointer.
- channelPair – index of channel pair, reference _asrc_channel_pair.
- config – ASRC channel pair configuration pointer.
- inputSampleRate – input audio data sample rate.
- outputSampleRate – output audio data sample rate.

uint32_t ASRC_GetOutSamplesSize(ASRC_Type *base, *asrc_channel_pair_t* channelPair, uint32_t inSampleRate, uint32_t outSampleRate, uint32_t inSamplesize)

Get output sample buffer size.

Note: This API is depends on the ASRC output configuration, should be called after the ASRC_SetChannelPairConfig.

Parameters

- base – asrc base pointer.
- channelPair – ASRC channel pair number.
- inSampleRate – input sample rate.
- outSampleRate – output sample rate.
- inSamplesize – input sampleS size.

Return values
output – buffer size in byte.

uint32_t ASRC_MapSamplesWidth(ASRC_Type *base, *asrc_channel_pair_t* channelPair, uint32_t *inWidth, uint32_t *outWidth)

Map register sample width to real sample width.

Note: This API is depends on the ASRC configuration, should be called after the ASRC_SetChannelPairConfig.

Parameters

- base – asrc base pointer.
- channelPair – asrc channel pair index.
- inWidth – ASRC channel pair number.
- outWidth – input sample rate.

Return values
input – sample mask value.

uint32_t ASRC_GetRemainFifoSamples(ASRC_Type *base, *asrc_channel_pair_t* channelPair, uint32_t *buffer, uint32_t outSampleWidth, uint32_t remainSamples)

Get left samples in fifo.

Parameters

- base – asrc base pointer.
- channelPair – ASRC channel pair number.
- buffer – input sample numbers.
- outSampleWidth – output sample width.
- remainSamples – output sample rate.

Return values
remain – samples number.

static inline void ASRC_ModuleEnable(ASRC_Type *base, bool enable)

ASRC module enable.

Parameters

- base – ASRC base pointer.
- enable – true is enable, false is disable

static inline void ASRC_ChannelPairEnable(ASRC_Type *base, *asrc_channel_pair_t* channelPair, bool enable)

ASRC enable channel pair.

Parameters

- base – ASRC base pointer.
- channelPair – channel pair mask value, reference _asrc_channel_pair_mask.
- enable – true is enable, false is disable.

static inline void ASRC_EnableInterrupt(ASRC_Type *base, uint32_t mask)

ASRC interrupt enable This function enable the ASRC interrupt with the provided mask.

Parameters

- base – ASRC peripheral base address.
- mask – The interrupts to enable. Logical OR of _asrc_interrupt_mask.

static inline void ASRC_DisableInterrupt(ASRC_Type *base, uint32_t mask)

ASRC interrupt disable This function disable the ASRC interrupt with the provided mask.

Parameters

- base – ASRC peripheral base address.
- mask – The interrupts to disable. Logical OR of _asrc_interrupt_mask.

static inline uint32_t ASRC_GetStatus(ASRC_Type *base)

Gets the ASRC status flag state.

Parameters

- base – ASRC base pointer

Returns

ASRC Tx status flag value. Use the Status Mask to get the status value needed.

static inline bool ASRC_GetChannelPairInitialStatus(ASRC_Type *base, *asrc_channel_pair_t* channel)

Gets the ASRC channel pair initialization state.

Parameters

- base – ASRC base pointer
- channel – ASRC channel pair.

Returns

ASRC Tx status flag value. Use the Status Mask to get the status value needed.

static inline uint32_t ASRC_GetChannelPairFifoStatus(ASRC_Type *base, *asrc_channel_pair_t* channelPair)

Gets the ASRC channel A fifo a status flag state.

Parameters

- base – ASRC base pointer
- channelPair – ASRC channel pair.

Returns

ASRC channel pair a fifo status flag value. Use the Status Mask to get the status value needed.

static inline void ASRC_ChannelPairWriteData(ASRC_Type *base, *asrc_channel_pair_t* channelPair, uint32_t data)

Writes data into ASRC channel pair FIFO. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.
- data – Data needs to be written.

static inline uint32_t ASRC_ChannelPairReadData(ASRC_Type *base, *asrc_channel_pair_t* channelPair)

Read data from ASRC channel pair FIFO. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.

Return values

value – read from fifo.

static inline uint32_t ASRC_GetInputDataRegisterAddress(ASRC_Type *base, *asrc_channel_pair_t* channelPair)

Get input data fifo address. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.

static inline uint32_t ASRC_GetOutputDataRegisterAddress(ASRC_Type *base, *asrc_channel_pair_t* channelPair)

Get output data fifo address. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.

status_t ASRC_SetIdealRatioConfig(ASRC_Type *base, *asrc_channel_pair_t* channelPair, uint32_t inputSampleRate, uint32_t outputSampleRate)

ASRC configure ideal ratio. The ideal ratio should be used when input clock source is not avalible.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.
- inputSampleRate – input audio data sample rate.
- outputSampleRate – output audio data sample rate.

status_t ASRC_TransferSetChannelPairConfig(ASRC_Type *base, *asrc_handle_t* *handle, *asrc_channel_pair_config_t* *config, uint32_t inputSampleRate, uint32_t outputSampleRate)

ASRC configure channel pair.

Parameters

- base – ASRC base pointer.
- handle – ASRC transactional handle pointer.
- config – ASRC channel pair configuration pointer.
- inputSampleRate – input audio data sample rate.
- outputSampleRate – output audio data sample rate.

void ASRC_TransferCreateHandle(ASRC_Type *base, *asrc_handle_t* *handle, *asrc_channel_pair_t* channelPair, *asrc_transfer_callback_t* inCallback, *asrc_transfer_callback_t* outCallback, void *userData)

Initializes the ASRC handle.

This function initializes the handle for the ASRC transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – ASRC base pointer
- handle – ASRC handle pointer.
- channelPair – ASRC channel pair.
- inCallback – Pointer to the user callback function.
- outCallback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

status_t ASRC_TransferNonBlocking(ASRC_Type *base, *asrc_handle_t* *handle, *asrc_transfer_t* *xfer)

Performs an interrupt non-blocking convert on asrc.

Note: This API returns immediately after the transfer initiates, application should check the wait and check the callback status.

Parameters

- base – asrc base pointer.
- handle – Pointer to the asrc_handle_t structure which stores the transfer state.
- xfer – Pointer to the ASRC_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_ASRCBusy – Previous receive still not finished.

status_t ASRC_TransferBlocking(ASRC_Type *base, *asrc_channel_pair_t* channelPair, *asrc_transfer_t* *xfer)

Performs an blocking convert on asrc.

Note: This API returns immediately after the convert finished.

Parameters

- base – asrc base pointer.
- channelPair – channel pair index.
- xfer – Pointer to the ASRC_transfer_t structure.

Return values

kStatus_Success – Successfully started the data receive.

status_t ASRC_TransferGetConvertedCount(ASRC_Type *base, *asrc_handle_t* *handle, size_t *count)

Get converted byte count.

Parameters

- base – ASRC base pointer.

- handle – Pointer to the asrc_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_ASRCIdle – There is not a non-blocking transaction currently in progress.

void ASRC_TransferAbortConvert(ASRC_Type *base, *asrc_handle_t* *handle)

Aborts the current convert.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – ASRC base pointer.
- handle – Pointer to the asrc_handle_t structure which stores the transfer state.

void ASRC_TransferTerminateConvert(ASRC_Type *base, *asrc_handle_t* *handle)

Terminate all ASRC convert.

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC_TransferAbortConvert.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

void ASRC_TransferHandleIRQ(ASRC_Type *base, *asrc_handle_t* *handle)

ASRC convert interrupt handler.

Parameters

- base – ASRC base pointer.
- handle – Pointer to the asrc_handle_t structure.

FSL_ASRC_DRIVER_VERSION

Version 2.1.3

ASRC return status .

Values:

enumerator kStatus_ASRCIdle

ASRC is idle.

enumerator kStatus_ASRCInIdle

ASRC in is idle.

enumerator kStatus_ASRCOutIdle

ASRC out is idle.

enumerator kStatus_ASRCBusy

ASRC is busy.

enumerator kStatus_ASRCInvalidArgument
ASRC invalid argument.

enumerator kStatus_ASRCClockConfigureFailed
ASRC clock configure failed

enumerator kStatus_ASRCChannelPairConfigureFailed
ASRC clock configure failed

enumerator kStatus_ASRCConvertError
ASRC clock configure failed

enumerator kStatus_ASRCNotSupport
ASRC not support

enumerator kStatus_ASRCQueueFull
ASRC queue is full

enumerator kStatus_ASRCOutQueueIdle
ASRC out queue is idle

enumerator kStatus_ASRCInQueueIdle
ASRC in queue is idle

enum _asrc_channel_pair
ASRC channel pair mask.

Values:

enumerator kASRC_ChannelPairA
channel pair A value

enumerator kASRC_ChannelPairB
channel pair B value

enumerator kASRC_ChannelPairC
channel pair C value

ASRC support sample rate .

Values:

enumerator kASRC_SampleRate_8000HZ
asrc sample rate 8KHZ

enumerator kASRC_SampleRate_11025HZ
asrc sample rate 11.025KHZ

enumerator kASRC_SampleRate_12000HZ
asrc sample rate 12KHZ

enumerator kASRC_SampleRate_16000HZ
asrc sample rate 16KHZ

enumerator kASRC_SampleRate_22050HZ
asrc sample rate 22.05KHZ

enumerator kASRC_SampleRate_24000HZ
asrc sample rate 24KHZ

enumerator kASRC_SampleRate_30000HZ
asrc sample rate 30KHZ

enumerator kASRC_SampleRate_32000HZ
asrc sample rate 32KHZ

enumerator kASRC_SampleRate_44100HZ
asrc sample rate 44.1KHZ

enumerator kASRC_SampleRate_48000HZ
asrc sample rate 48KHZ

enumerator kASRC_SampleRate_64000HZ
asrc sample rate 64KHZ

enumerator kASRC_SampleRate_88200HZ
asrc sample rate 88.2KHZ

enumerator kASRC_SampleRate_96000HZ
asrc sample rate 96KHZ

enumerator kASRC_SampleRate_128000HZ
asrc sample rate 128KHZ

enumerator kASRC_SampleRate_176400HZ
asrc sample rate 176.4KHZ

enumerator kASRC_SampleRate_192000HZ
asrc sample rate 192KHZ

The ASRC interrupt enable flag .

Values:

enumerator kASRC_FPInWaitStateInterruptEnable
FP in wait state mask

enumerator kASRC_OverLoadInterruptMask
overload interrupt mask

enumerator kASRC_DataOutputCInterruptMask
data output c interrupt mask

enumerator kASRC_DataOutputBInterruptMask
data output b interrupt mask

enumerator kASRC_DataOutputAInterruptMask
data output a interrupt mask

enumerator kASRC_DataInputCInterruptMask
data input c interrupt mask

enumerator kASRC_DataInputBInterruptMask
data input b interrupt mask

enumerator kASRC_DataInputAInterruptMask
data input a interrupt mask

The ASRC interrupt status .

Values:

enumerator kASRC_StatusDSLCounterReady
DSL counter

enumerator kASRC_StatusTaskQueueOverLoad
task queue overload

enumerator kASRC_StatusPairCOutputOverLoad
pair c output overload

enumerator kASRC_StatusPairBOutputOverLoad
pair b output overload

enumerator kASRC_StatusPairAOutputOverLoad
pair a output overload

enumerator kASRC_StatusPairCInputOverLoad
pair c input overload

enumerator kASRC_StatusPairBInputOverLoad
pair b input overload

enumerator kASRC_StatusPairAInputOverLoad
pair a input overload

enumerator kASRC_StatusPairCOutputOverflow
pair c output overflow

enumerator kASRC_StatusPairBOutputOverflow
pair b output overflow

enumerator kASRC_StatusPairAOutputOverflow
pair a output overflow

enumerator kASRC_StatusPairCInputUnderflow
pair c input underflow

enumerator kASRC_StatusPairBInputUnderflow
pair b input under flow

enumerator kASRC_StatusPairAInputUnderflow
pair a input underflow

enumerator kASRC_StatusFPInWaitState
FP in wait state

enumerator kASRC_StatusOverloadError
overload error

enumerator kASRC_StatusInputError
input error status

enumerator kASRC_StatusOutputError
output error status

enumerator kASRC_StatusPairCOutputReady
pair c output ready

enumerator kASRC_StatusPairBOutputReady
pair b output ready

enumerator kASRC_StatusPairAOutputReady
pair a output ready

enumerator kASRC_StatusPairCInputReady
pair c input ready

enumerator kASRC_StatusPairBInputReady
pair b input ready

enumerator kASRC_StatusPairAInputReady
pair a input ready

enumerator kASRC_StatusPairAInterrupt
pair A interrupt

enumerator kASRC_StatusPairBInterrupt
pair B interrupt

enumerator kASRC_StatusPairCInterrupt
pair C interrupt

ASRC channel pair status .

Values:

enumerator kASRC_OutputFifoNearFull
channel pair output fifo near full

enumerator kASRC_InputFifoNearEmpty
channel pair input fifo near empty

enum _asrc_ratio
ASRC ideal ratio.

Values:

enumerator kASRC_RatioNotUsed
ideal ratio not used

enumerator kASRC_RatioUseInternalMeasured
ideal ratio use internal measure ratio, can be used for real time streaming audio

enumerator kASRC_RatioUseIdealRatio
ideal ratio use manual configure ratio, can be used for the non-real time streaming audio

enum _asrc_audio_channel
Number of channels in audio data.

Values:

enumerator kASRC_ChannelsNumber1
channel number is 1

enumerator kASRC_ChannelsNumber2
channel number is 2

enumerator kASRC_ChannelsNumber3
channel number is 3

enumerator kASRC_ChannelsNumber4
channel number is 4

enumerator kASRC_ChannelsNumber5
channel number is 5

enumerator kASRC_ChannelsNumber6
channel number is 6

enumerator kASRC_ChannelsNumber7
channel number is 7

enumerator kASRC_ChannelsNumber8
channel number is 8

enumerator kASRC_ChannelsNumber9
channel number is 9

enumerator kASRC_ChannelsNumber10
channel number is 10

enum _asrc_data_width
data width

Values:

enumerator kASRC_DataWidth24Bit
data width 24bit

enumerator kASRC_DataWidth16Bit
data width 16bit

enumerator kASRC_DataWidth8Bit
data width 8bit

enum _asrc_data_align
data alignment

Values:

enumerator kASRC_DataAlignMSB
data alignment MSB

enumerator kASRC_DataAlignLSB
data alignment LSB

enum _asrc_sign_extension
sign extension

Values:

enumerator kASRC_NoSignExtension
no sign extension

enumerator kASRC_SignExtension
sign extension

typedef enum *_asrc_channel_pair* asrc_channel_pair_t
ASRC channel pair mask.

typedef enum *_asrc_ratio* asrc_ratio_t
ASRC ideal ratio.

typedef enum *_asrc_audio_channel* asrc_audio_channel_t
Number of channels in audio data.

typedef enum *_asrc_data_width* asrc_data_width_t
data width

typedef enum *_asrc_data_align* asrc_data_align_t
data alignment

typedef enum *_asrc_sign_extension* asrc_sign_extension_t
sign extension

typedef struct *_asrc_channel_pair_config* asrc_channel_pair_config_t
asrc channel pair configuation

typedef struct *_asrc_transfer* asrc_transfer_t
SAI transfer structure.

typedef struct *_asrc_handle* asrc_handle_t
asrc handler

typedef void (*asrc_transfer_callback_t)(ASRC_Type *base, *asrc_handle_t* *handle, *status_t* status, void *userData)
ASRC transfer callback prototype.

typedef struct *_asrc_in_handle* asrc_in_handle_t
asrc in handler

typedef struct *_asrc_out_handle* asrc_out_handle_t
output handler

ASRC_XFER_QUEUE_SIZE
ASRC transfer queue size, user can refine it according to use case.

FSL_ASRC_CHANNEL_PAIR_COUNT
ASRC channel pair count.

FSL_ASRC_CHANNEL_PAIR_FIFO_DEPTH
ASRC FIFO depth.

ASRC_ASRCTR_AT_MASK(index)
ASRC register access macro.

ASRC_ASRCTR_RATIO_MASK(index)

ASRC_ASRCTR_RATIO(ratio, index)

ASRC_ASRIER_INPUT_INTERRUPT_MASK(index)

ASRC_ASRIER_OUTPUTPUT_INTERRUPT_MASK(index)

ASRC_ASRCNCR_CHANNEL_COUNTER_MASK(index)

ASRC_ASRCNCR_CHANNEL_COUNTER(counter, index)

ASRC_ASRCFG_PRE_MODE_MASK(index)

ASRC_ASRCFG_PRE_MODE(mode, index)

ASRC_ASRCFG_POST_MODE_MASK(index)

ASRC_ASRCFG_POST_MODE(mode, index)

ASRC_ASRCFG_INIT_DONE_MASK(index)

ASRC_ASRCSR_INPUT_CLOCK_SOURCE_MASK(index)

ASRC_ASRCSR_INPUT_CLOCK_SOURCE(source, index)

ASRC_ASRCSR_OUTPUT_CLOCK_SOURCE_MASK(index)

ASRC_ASRCSR_OUTPUT_CLOCK_SOURCE(source, index)

ASRC_ASRCDR_INPUT_PRESCALER_MASK(index)

ASRC_ASRCDR_INPUT_PRESCALER(prescaler, index)

ASRC_ASRCDR_INPUT_DIVIDER_MASK(index)

ASRC_ASRCDR_INPUT_DIVIDER(divider, index)

ASRC_ASRCDR_OUTPUT_PRESCALER_MASK(index)

ASRC_ASRCDR_OUTPUT_PRESCALER(prescaler, index)

ASRC_ASRCDR_OUTPUT_DIVIDER_MASK(index)

ASRC_ASRCDR_OUTPUT_DIVIDER(divider, index)

ASCR_ASRCDR_OUTPUT_CLOCK_DIVIDER_PRESCALER(value, index)

ASCR_ASRCDR_INPUT_CLOCK_DIVIDER_PRESCALER(value, index)

ASRC_IDEAL_RATIO_HIGH(base, index)

ASRC_IDEAL_RATIO_LOW(base, index)

ASRC_ASRMCR(base, index)

ASRC_ASRMCR1(base, index)

ASRC_ASRDI(base, index)

ASRC_ASRDO(base, index)

ASRC_ASRDI_ADDR(base, index)

ASRC_ASRDO_ADDR(base, index)

ASRC_ASRFST_ADDR(base, index)

ASRC_GET_CHANNEL_COUNTER(base, index)

struct _asrc_channel_pair_config

#include <fsl_asrc.h> asrc channel pair configuation

Public Members

asrc_audio_channel_t audioDataChannels

audio data channel numbers

asrc_clock_source_t inClockSource

input clock source, reference the clock source definition in SOC header file

uint32_t inSourceClock_Hz

input source clock frequency

asrc_clock_source_t outClockSource

output clock source, reference the clock source definition in SOC header file

uint32_t outSourceClock_Hz

output source clock frequency

asrc_ratio_t sampleRateRatio

sample rate ratio type

asrc_data_width_t inDataWidth
input data width

asrc_data_align_t inDataAlign
input data alignment

asrc_data_width_t outDataWidth
output data width

asrc_data_align_t outDataAlign
output data alignment

asrc_sign_extension_t outSignExtension
output extension

uint8_t outFifoThreshold
output fifo threshold

uint8_t inFifoThreshold
input fifo threshold

bool bufStallWhenFifoEmptyFull
stall Pair A conversion in case of Buffer near empty full condition

struct __asrc__transfer
#include <fsl_asrc.h> SAI transfer structure.

Public Members

void *inData
Data address to convert.

size_t inDataSize
input data size.

void *outData
Data address to store converted data

size_t outDataSize
output data size.

struct __asrc__in__handle
#include <fsl_asrc.h> asrc in handler

Public Members

asrc_transfer_callback_t callback
Callback function called at convert complete

uint32_t sampleWidth
data width

uint32_t sampleMask
data mask

uint32_t fifoThreshold
fifo threshold

uint8_t *asrcQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSamples[(4U)]
Data bytes need to convert

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

struct __asrc_out_handle
#include <fsl_asrc.h> output handler

Public Members

asrc_transfer_callback_t callback
Callback function called at convert complete

uint32_t sampleWidth
data width

uint32_t fifoThreshold
fifo threshold

uint8_t *asrcQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSamples[(4U)]
Data bytes need to convert

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

struct __asrc_handle
#include <fsl_asrc.h> ASRC handle structure.

Public Members

ASRC_Type *base
base address

uint32_t state
Transfer status

void *userData
Callback parameter passed to callback function

asrc_audio_channel_t audioDataChannels
audio channel number

asrc_channel_pair_t channelPair
channel pair mask

asrc_in_handle_t in
asrc input handler

asrc_out_handle_t out
asrc output handler

2.5 ASRC EDMA Driver

void ASRC_TransferInCreateHandleEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle, *asrc_channel_pair_t* channelPair, *asrc_edma_callback_t* callback, *edma_handle_t* *inDmaHandle, const *asrc_p2p_edma_config_t* *periphConfig, void *userData)

Initializes the ASRC IN eDMA handle.

This function initializes the ASRC DMA handle, which can be used for other ASRC transactional APIs. Usually, for a specified ASRC channel pair, call this API once to get the initialized handle.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair
- handle – ASRC eDMA handle pointer.
- callback – Pointer to user callback function.
- inDmaHandle – DMA handler for ASRC in.
- periphConfig – peripheral configuration.
- userData – User parameter passed to the callback function.

void ASRC_TransferOutCreateHandleEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle, *asrc_channel_pair_t* channelPair, *asrc_edma_callback_t* callback, *edma_handle_t* *outDmaHandle, const *asrc_p2p_edma_config_t* *periphConfig, void *userData)

Initializes the ASRC OUT eDMA handle.

This function initializes the ASRC DMA handle, which can be used for other ASRC transactional APIs. Usually, for a specified ASRC channel pair, call this API once to get the initialized handle.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair
- handle – ASRC eDMA handle pointer.
- callback – Pointer to user callback function.
- outDmaHandle – DMA handler for ASRC out.
- periphConfig – peripheral configuration.
- userData – User parameter passed to the callback function.

status_t ASRC_TransferSetChannelPairConfigEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle, *asrc_channel_pair_config_t* *asrcConfig, uint32_t inSampleRate, uint32_t outSampleRate)

Configures the ASRC P2P channel pair.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.
- asrcConfig – asrc configurations.

- inSampleRate – ASRC input sample rate.
- outSampleRate – ASRC output sample rate.

uint32_t ASRC_GetOutSamplesSizeEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle, uint32_t inSampleRate, uint32_t outSampleRate, uint32_t inSamplesize)

Get output sample buffer size can be transferred by edma.

Note: This API is depends on the ASRC output configuration, should be called after the ASRC_TransferSetChannelPairConfigEDMA.

Parameters

- base – asrc base pointer.
- handle – ASRC channel pair edma handle.
- inSampleRate – input sample rate.
- outSampleRate – output sample rate.
- inSamplesize – input sampleS size.

Return values

output – buffer size in byte.

status_t ASRC_TransferEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle, *asrc_transfer_t* *xfer)

Performs a non-blocking ASRC m2m convert using EDMA.

Note: This interface returns immediately after the transfer initiates.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a ASRC eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_ASRCQueueFull – ASRC EDMA driver queue is full.

void ASRC_TransferInAbortEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle)

Aborts a ASRC IN 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 ASRC_TransferTerminalP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

void ASRC_TransferOutAbortEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle)

Aborts a ASRC OUT 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 ASRC_TransferTerminalP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

void ASRC_TransferInTerminalEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle)

Terminate In ASRC Convert.

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC_TransferAbortPP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

void ASRC_TransferOutTerminalEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle)

Terminate Out ASRC Convert.

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC_TransferAbortPP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

FSL_ASRC_EDMA_DRIVER_VERSION

Version 2.2.0

typedef struct *_asrc_edma_handle* asrc_edma_handle_t

typedef void (*asrc_edma_callback_t)(ASRC_Type *base, *asrc_edma_handle_t* *handle, *status_t* status, void *userData)

ASRC eDMA transfer callback function for finish and error.

typedef void (*asrc_start_peripheral_t)(bool start)

ASRC trigger peripheral function pointer.

typedef struct *_asrc_p2p_edma_config* asrc_p2p_edma_config_t

destination peripheral configuration

typedef struct *_asrc_in_edma_handle* asrc_in_edma_handle_t

@ brief asrc in edma handler

typedef struct *_asrc_out_edma_handle* asrc_out_edma_handle_t

@ brief asrc out edma handler

ASRC_XFER_IN_QUEUE_SIZE

ASRC IN edma QUEUE size.

<

ASRC_XFER_OUT_QUEUE_SIZE

struct _asrc_p2p_edma_config

#include <fsl_asrc_edma.h> destination peripheral configuration

Public Members

asrc_start_peripheral_t startPeripheral
trigger peripheral start

struct __asrc__in__edma__handle
#include <fsl_asrc_edma.h> @ brief asrc in edma handler

Public Members

edma_handle_t *inDmaHandle
DMA handler for ASRC in

uint8_t tcd[(4U + 1U) * sizeof(*edma_tcd_t*)]
TCD pool for eDMA send.

uint32_t sampleWidth
input data width

uint32_t fifoThreshold
ASRC input fifo threshold

uint32_t *asrcQueue[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 state
Internal state for ASRC eDMA transfer

const *asrc_p2p_edma_config_t* *peripheralConfig
peripheral configuration pointer

struct __asrc__out__edma__handle
#include <fsl_asrc_edma.h> @ brief asrc out edma handler

Public Members

edma_handle_t *outDmaHandle
DMA handler for ASRC out

uint8_t tcd[(((4U) * 2U) + 1U) * sizeof(*edma_tcd_t*)]
TCD pool for eDMA send.

uint32_t sampleWidth
output data width

uint32_t fifoThreshold
ASRC output fifo threshold

uint32_t *asrcQueue[((4U) * 2U)]
Transfer queue storing queued transfer.

size_t transferSize[((4U) * 2U)]
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 state
Internal state for ASRC eDMA transfer

const *asrc_p2p_edma_config_t* *peripheralConfig
peripheral configuration pointer

struct _asrc_edma_handle
#include <fsl_asrc_edma.h> ASRC DMA transfer handle.

Public Members

asrc_in_edma_handle_t in
asrc in handler

asrc_out_edma_handle_t out
asrc out handler

asrc_channel_pair_t channelPair
channel pair

void *userData
User callback parameter

asrc_edma_callback_t callback
Callback for users while transfer finish or error occurs

2.6 Battery-Backed Non-Secure Module

void BBNSM_Init(BBNSM_Type *base)
Init the BBNSM section.

Parameters

- base – BBNSM peripheral base address

void BBNSM_Deinit(BBNSM_Type *base)
Deinit the BBNSM section.

Parameters

- base – BBNSM peripheral base address

FSL_BBNSM_DRIVER_VERSION
Version 2.0.0

enum _bbnsm_interrupts
List of BBNSM interrupts.

Values:

enumerator kBBNSM_RTC_AlarmInterrupt
RTC time alarm interrupt

enumerator kBBNSM_RTC_RolloverInterrupt
RTC rollover interrupt

enum _bbnsm_status_flags
List of BBNSM flags.

Values:

enumerator kBBNSM_RTC_AlarmInterruptFlag
RTC time alarm interrupt flag

enumerator kBBNSM_RTC_RolloverInterruptFlag
RTC rollover interrupt flag

enumerator kBBNSM_PWR_ON_InterruptFlag
power on interrupt flag

enumerator kBBNSM_PWR_OFF_InterruptFlag
power off interrupt flag

enumerator kBBNSM_EMG_OFF_InterruptFlag
emergency power off interrupt flag

typedef enum *_bbnsm_interrupts* bbnsm_interrupts_t
List of BBNSM interrupts.

typedef enum *_bbnsm_status_flags* bbnsm_status_flags_t
List of BBNSM flags.

typedef struct *_bbnsm_rtc_config* bbnsm_rtc_config_t
BBNSM config structure.

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

The config struct can be made const so it resides in flash

void BBNSM_RTC_Init(BBNSM_Type *base, const *bbnsm_rtc_config_t* *config)
Ungates the BBNSM clock and configures the peripheral for basic operation.

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

Parameters

- base – BBNSM peripheral base address
- config – Pointer to the user's BBNSM rtc configuration structure.

void BBNSM_RTC_Deinit(BBNSM_Type *base)
Stops the RTC timer.

Parameters

- base – BBNSM peripheral base address

void BBNSM_RTC_GetDefaultConfig(*bbnsm_rtc_config_t* *config)
Fills in the BBNSM RTC config struct with the default settings.

The default values are as follows.

```
config->rtccalenable = false;
config->rtccalvalue = 0U;
```

Parameters

- config – Pointer to the user's BBNSM configuration structure.

status_t BBNSM_RTC_SetAlarm(BBNSM_Type *base, uint32_t alarmSeconds)

Sets the BBNSM RTC alarm time.

The function sets the RTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error. Please note, that RTC alarm has limited resolution because only 32 most significant bits of RTC counter are compared to RTC Alarm register. If the alarm time is beyond RTC resolution, the function does not set the alarm and returns an error.

Parameters

- base – BBNSM peripheral base address
- alarmSeconds –

Returns

kStatus_Success: success in setting the BBNSM RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed or is beyond resolution

uint32_t BBNSM_RTC_GetAlarm(BBNSM_Type *base)

Returns the BBNSM RTC alarm time.

Parameters

- base – BBNSM peripheral base address

struct _bbnsm_rtc_config

#include <fsl_bbnsm.h> BBNSM config structure.

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

The config struct can be made const so it resides in flash

Public Members

bool rtcCalEnable

true: RTC calibration mechanism is enabled; false: No calibration is used

uint32_t rtcCalValue

Defines signed calibration value for RTC; This is a 5-bit 2's complement value, range from -16 to +15

2.7 CACHE: ARMV7-M7 CACHE Memory Controller

static inline void L1CACHE_EnableICache(void)

Enables cortex-m7 L1 instruction cache.

static inline void L1CACHE_DisableICache(void)

Disables cortex-m7 L1 instruction cache.

static inline void L1CACHE_InvalidateICache(void)

Invalidate cortex-m7 L1 instruction cache.

void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)

Invalidate cortex-m7 L1 instruction cache by range.

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

Parameters

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

static inline void L1CACHE_EnableDCache(void)

Enables cortex-m7 L1 data cache.

static inline void L1CACHE_DisableDCache(void)

Disables cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCache(void)

Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_CleanDCache(void)

Cleans cortex-m7 L1 data cache.

static inline void L1CACHE_CleanInvalidateDCache(void)

Cleans and Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cortex-m7 L1 data cache by range.

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

Parameters

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

static inline void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cortex-m7 L1 data cache by range.

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

Parameters

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

static inline void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

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

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

Parameters

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

void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all instruction caches by range.

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

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

Parameters

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

void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all data caches by range.

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

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

Parameters

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

void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Cleans all data caches by range.

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

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size

if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates all data caches by range.

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

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

Parameters

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

FSL_CACHE_DRIVER_VERSION

cache driver version 2.0.4.

2.8 DAC12: 12-bit Digital-to-Analog Converter Driver

void DAC12_GetHardwareInfo(DAC_Type *base, *dac12_hardware_info_t* *info)

Get hardware information about this module.

Parameters

- base – DAC12 peripheral base address.
- info – Pointer to info structure, see to dac12_hardware_info_t.

void DAC12_Init(DAC_Type *base, const *dac12_config_t* *config)

Initialize the DAC12 module.

Parameters

- base – DAC12 peripheral base address.
- config – Pointer to configuration structure, see to dac12_config_t.

void DAC12_GetDefaultConfig(*dac12_config_t* *config)

Initializes the DAC12 user configuration structure.

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

```
config->fifoWatermarkLevel = 0U;
config->fifoWorkMode = kDAC12_FIFODisabled;
config->referenceVoltageSource = kDAC12_ReferenceVoltageSourceAlt1;
config->fifoTriggerMode = kDAC12_FIFOTriggerByHardwareMode;
config->referenceCurrentSource = kDAC12_ReferenceCurrentSourceAlt0;
config->speedMode = kDAC12_SpeedLowMode;
```

(continues on next page)

(continued from previous page)

```
config->speedMode = false;
config->currentReferenceInternalTrimValue = 0x4;
```

Parameters

- config – Pointer to the configuration structure. See “dac12_config_t”.

void DAC12_Deinit(DAC_Type *base)

De-initialize the DAC12 module.

Parameters

- base – DAC12 peripheral base address.

static inline void DAC12_Enable(DAC_Type *base, bool enable)

Enable the DAC12’s converter or not.

Parameters

- base – DAC12 peripheral base address.
- enable – Enable the DAC12’s converter or not.

static inline void DAC12_ResetConfig(DAC_Type *base)

Reset all internal logic and registers.

Parameters

- base – DAC12 peripheral base address.

static inline void DAC12_ResetFIFO(DAC_Type *base)

Reset the FIFO pointers.

FIFO pointers should only be reset when the DAC12 is disabled. This function can be used to configure both pointers to the same address to reset the FIFO as empty.

Parameters

- base – DAC12 peripheral base address.

static inline uint32_t DAC12_GetStatusFlags(DAC_Type *base)

Get status flags.

Parameters

- base – DAC12 peripheral base address.

Returns

Mask of current status flags. See to _dac12_status_flags.

static inline void DAC12_ClearStatusFlags(DAC_Type *base, uint32_t flags)

Clear status flags.

Note: Not all the flags can be cleared by this API. Several flags need special condition to clear them according to target chip’s reference manual document.

Parameters

- base – DAC12 peripheral base address.
- flags – Mask of status flags to be cleared. See to _dac12_status_flags.

static inline void DAC12_EnableInterrupts(DAC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – DAC12 peripheral base address.

- mask – Mask value of interrupts to be enabled. See to _dac12_interrupt_enable.

static inline void DAC12_DisableInterrupts(DAC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – DAC12 peripheral base address.
- mask – Mask value of interrupts to be disabled. See to _dac12_interrupt_enable.

static inline void DAC12_EnableDMA(DAC_Type *base, bool enable)

Enable DMA or not.

When DMA is enabled, the DMA request will be generated by original interrupts. The interrupts will not be presented on this module at the same time.

static inline void DAC12_SetData(DAC_Type *base, uint32_t value)

Set data into the entry of FIFO buffer.

When the DAC FIFO is disabled, and the one entry buffer is enabled, the DAC converts the data in the buffer to analog output voltage. Any write to the DATA register will replace the data in the buffer and push data to analog conversion without trigger support. When the DAC FIFO is enabled. Writing data would increase the write pointer of FIFO. Also, the data would be restored into the FIFO buffer.

Parameters

- base – DAC12 peripheral base address.
- value – Setting value into FIFO buffer.

static inline void DAC12_DoSoftwareTrigger(DAC_Type *base)

Do trigger the FIFO by software.

When the DAC FIFO is enabled, and software trigger is used. Doing trigger would increase the read pointer, and the data in the entry pointed by read pointer would be converted as new output.

Parameters

- base – DAC12 peripheral base address.

static inline uint32_t DAC12_GetFIFOReadPointer(DAC_Type *base)

Get the current read pointer of FIFO.

Parameters

- base – DAC12 peripheral base address.

Returns

Read pointer index of FIFO buffer.

static inline uint32_t DAC12_GetFIFOWritePointer(DAC_Type *base)

Get the current write pointer of FIFO.

Parameters

- base – DAC12 peripheral base address.

Returns

Write pointer index of FIFO buffer

FSL_DAC12_DRIVER_VERSION

DAC12 driver version 2.1.2.

enum _dac12_status_flags

DAC12 flags.

Values:

enumerator kDAC12_OverflowFlag

FIFO overflow status flag, which indicates that more data has been written into FIFO than it can hold.

enumerator kDAC12_UnderflowFlag

FIFO underflow status flag, which means that there is a new trigger after the FIFO is nearly empty.

enumerator kDAC12_WatermarkFlag

FIFO wartermark status flag, which indicates the remaining FIFO data is less than the watermark setting.

enumerator kDAC12_NearlyEmptyFlag

FIFO nearly empty flag, which means there is only one data remaining in FIFO.

enumerator kDAC12_FullFlag

FIFO full status flag, which means that the FIFO read pointer equals the write pointer, as the write pointer increase.

enum _dac12_interrupt_enable

DAC12 interrupts.

Values:

enumerator kDAC12_UnderOrOverflowInterruptEnable

Underflow and overflow interrupt enable.

enumerator kDAC12_WatermarkInterruptEnable

Watermark interrupt enable.

enumerator kDAC12_NearlyEmptyInterruptEnable

Nearly empty interrupt enable.

enumerator kDAC12_FullInterruptEnable

Full interrupt enable.

enum _dac12_fifo_size_info

DAC12 FIFO size information provided by hardware.

Values:

enumerator kDAC12_FIFOSize2

FIFO depth is 2.

enumerator kDAC12_FIFOSize4

FIFO depth is 4.

enumerator kDAC12_FIFOSize8

FIFO depth is 8.

enumerator kDAC12_FIFOSize16

FIFO depth is 16.

enumerator kDAC12_FIFOSize32

FIFO depth is 32.

enumerator kDAC12_FIFOSize64

FIFO depth is 64.

enumerator kDAC12_FIFOSize128
FIFO depth is 128.

enumerator kDAC12_FIFOSize256
FIFO depth is 256.

enum _dac12_fifo_work_mode
DAC12 FIFO work mode.

Values:

enumerator kDAC12_FIFODisabled
FIFO disabled and only one level buffer is enabled. Any data written from this buffer goes to conversion.

enumerator kDAC12_FIFOWorkAsNormalMode
Data will first read from FIFO to buffer then go to conversion.

enumerator kDAC12_FIFOWorkAsSwingMode
In Swing mode, the FIFO must be set up to be full. In Swing back mode, a trigger changes the read pointer to make it swing between the FIFO Full and Nearly Empty state. That is, the trigger increases the read pointer till FIFO is nearly empty and decreases the read pointer till the FIFO is full.

enum _dac12_reference_voltage_source
DAC12 reference voltage source.

Values:

enumerator kDAC12_ReferenceVoltageSourceAlt1
The DAC selects DACREF_1 as the reference voltage.

enumerator kDAC12_ReferenceVoltageSourceAlt2
The DAC selects DACREF_2 as the reference voltage.

enum _dac12_fifo_trigger_mode
DAC12 FIFO trigger mode.

Values:

enumerator kDAC12_FIFOTriggerByHardwareMode
Buffer would be triggered by hardware.

enumerator kDAC12_FIFOTriggerBySoftwareMode
Buffer would be triggered by software.

enum _dac12_reference_current_source
DAC internal reference current source.

Analog module needs reference current to keep working . Such reference current can generated by IP itself, or by on-chip PMC's "reference part". If no current reference be selected, analog module can't working normally ,even when other register can still be assigned, DAC would waste current but no function. To make the DAC work, either kDAC12_ReferenceCurrentSourceAltx should be selected.

Values:

enumerator kDAC12_ReferenceCurrentSourceDisabled
None of reference current source is enabled.

enumerator kDAC12_ReferenceCurrentSourceAlt0
Use the internal reference current generated by the module itself.

enumerator kDAC12_ReferenceCurrentSourceAlt1

Use the ZTC(Zero Temperature Coefficient) reference current generated by on-chip power management module.

enumerator kDAC12_ReferenceCurrentSourceAlt2

Use the PTAT(Proportional To Absolution Temperature) reference current generated by power management module.

enum _dac12_speed_mode

DAC analog buffer speed mode for conversion.

Values:

enumerator kDAC12_SpeedLowMode

Low speed mode.

enumerator kDAC12_SpeedMiddleMode

Middle speed mode.

enumerator kDAC12_SpeedHighMode

High speed mode.

typedef enum *_dac12_fifo_size_info* dac12_fifo_size_info_t

DAC12 FIFO size information provided by hardware.

typedef enum *_dac12_fifo_work_mode* dac12_fifo_work_mode_t

DAC12 FIFO work mode.

typedef enum *_dac12_reference_voltage_source* dac12_reference_voltage_source_t

DAC12 reference voltage source.

typedef enum *_dac12_fifo_trigger_mode* dac12_fifo_trigger_mode_t

DAC12 FIFO trigger mode.

typedef enum *_dac12_reference_current_source* dac12_reference_current_source_t

DAC internal reference current source.

Analog module needs reference current to keep working . Such reference current can generated by IP itself, or by on-chip PMC's "reference part". If no current reference be selected, analog module can't working normally ,even when other register can still be assigned, DAC would waste current but no function. To make the DAC work, either kDAC12_ReferenceCurrentSourceAltx should be selected.

typedef enum *_dac12_speed_mode* dac12_speed_mode_t

DAC analog buffer speed mode for conversion.

typedef struct *_dac12_hardware_info* dac12_hardware_info_t

DAC12 hardware information.

DAC12_CR_W1C_FLAGS_MASK

Define "write 1 to clear" flags.

DAC12_CR_ALL_FLAGS_MASK

Define all the flag bits in DACx_CR register.

struct _dac12_hardware_info

#include <fsl_dac12.h> DAC12 hardware information.

Public Members

dac12_fifo_size_info_t fifoSizeInfo

The number of words in this device's DAC buffer.

struct dac12_config_t

#include <fsl_dac12.h> DAC12 module configuration.

Actually, the most fields are for FIFO buffer.

Public Members

uint32_t fifoWatermarkLevel

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

dac12_fifo_work_mode_t fifoWorkMode

FIFI's work mode about pointers.

dac12_reference_voltage_source_t referenceVoltageSource

Select the reference voltage source.

dac12_reference_current_source_t referenceCurrentSource

Select the trigger mode for FIFO. Select the reference current source.

dac12_speed_mode_t speedMode

Select the speed mode for conversion.

bool enableAnalogBuffer

Enable analog buffer for high drive.

2.9 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 enableContinuousLinkMode
: Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

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.

2.10 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

2.11 eDMA soc Driver

2.12 EQDC: Enhanced Quadrature Encoder/Decoder

void EQDC_Init(EQDC_Type *base, const *eqdc_config_t* *psConfig)
Initializes the EQDC module.

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

Parameters

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

void EQDC_GetDefaultConfig(*eqdc_config_t* *psConfig)

Gets an available pre-defined configuration.

The default value are:

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

Parameters

- psConfig – Pointer to configuration structure.

void EQDC_Deinit(EQDC_Type *base)

De-initializes the EQDC module.

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

Parameters

- base – EQDC peripheral base address.

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

Initializes the mode of operation.

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

Parameters

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

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

Initializes the mode of count.

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

Parameters

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

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

Enable watchdog for EQDC module.

Parameters

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

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

Set watchdog timeout value.

Parameters

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

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

Enable DMA for EQDC module.

Parameters

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

static inline void EQDC_SetBufferedRegisterLoadUpdateMode(EQDC_Type *base)

Set Buffered Register Load (Update) Mode.

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

Parameters

- base – EQDC peripheral base address

static inline void EQDC_ClearBufferedRegisterLoadUpdateMode(EQDC_Type *base)

Clear Buffered Register Load (Update) Mode.

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

Parameters

- base – EQDC peripheral base address

static inline void EQDC_SetEqdcLdok(EQDC_Type *base)

Set load okay.

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

Parameters

- base – EQDC peripheral base address.

static inline uint8_t EQDC_GetEqdcLdok(EQDC_Type *base)

Get load okay.

Parameters

- base – EQDC peripheral base address.

static inline void EQDC_ClearEqdcLdok(EQDC_Type *base)

Clear load okay.

Parameters

- base – EQDC peripheral base address.

static inline uint32_t EQDC_GetStatusFlags(EQDC_Type *base)

Get the status flags.

Parameters

- base – EQDC peripheral base address.

Returns

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

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

Clear the status flags.

Parameters

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

static inline uint16_t EQDC_GetSignalStatusFlags(EQDC_Type *base)

Get the signals' real-time status.

Parameters

- base – EQDC peripheral base address.

Returns

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

static inline *eqdc_count_direction_flag_t* EQDC_GetLastCountDirection(EQDC_Type *base)

Get the direction of the last count.

Parameters

- base – EQDC peripheral base address.

Returns

Direction of the last count.

static inline void EQDC_EnableInterrupts(EQDC_Type *base, uint32_t u32Interrupts)

Enable the interrupts.

Parameters

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

static inline void EQDC_DisableInterrupts(EQDC_Type *base, uint32_t u32Interrupts)

Disable the interrupts.

Parameters

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

static inline void EQDC_DoSoftwareLoadInitialPositionValue(EQDC_Type *base)

Load the initial position value to position counter.

Software trigger to load the initial position value (UINIT and LINIT) contents to position counter (UPOS and LPOS), so that to provide the consistent operation the position counter registers.

Parameters

- base – EQDC peripheral base address.

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

Set initial position value for EQDC module.

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

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

b. If CTRL2[LDMODE] is 0, "buffered" values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

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

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

Set position counter value.

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

Parameters

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

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

Set position counter modulus value.

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

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

b. If CTRL2[LDMODE] is 0, "buffered" values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- positionModulusValue – Position modulus value

static inline void EQDC_SetPositionCompare0Value(EQDC_Type *base, uint32_t u32PositionComp0Value)

Set position counter compare 0 value.

Set the position counter compare 0 value (UCOMP0, LCOMP0). After writing values to the UCOMP0 and LCOMP0 registers, the values are "buffered" into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

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

b. If CTRL2[LDMODE] is 0, "buffered" values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp0Value – Position modulus value

static inline void EQDC_SetPositionCompare1Value(EQDC_Type *base, uint32_t u32PositionComp1Value)

Set position counter compare 1 value.

Set the position counter compare 1 value (UCOMP1, LCOMP1). After writing values to the UCOMP1 and LCOMP1 registers, the values are "buffered" into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

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

b. If CTRL2[LDMODE] is 0, "buffered" values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp1Value – Position modulus value

static inline void EQDC_SetPositionCompare2Value(EQDC_Type *base, uint32_t u32PositionComp2Value)

Set position counter compare 2 value.

Set the position counter compare 2 value (UCOMP2, LCOMP2). After writing values to the UCOMP2 and LCOMP2 registers, the values are "buffered" into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

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

b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp2Value – Position modulus value

static inline void EQDC_SetPositionCompare3Value(EQDC_Type *base, uint32_t u32PositionComp3Value)

Set position counter compare 3 value.

Set the position counter compare 3 value (UCOMP3, LCOMP3). After writing values to the UCOMP3 and LCOMP3 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

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

b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp3Value – Position modulus value

static inline uint32_t EQDC_GetPosition(EQDC_Type *base)

Get the current position counter’s value.

Parameters

- base – EQDC peripheral base address.

Returns

Current position counter’s value.

static inline uint32_t EQDC_GetHoldPosition(EQDC_Type *base)

Get the hold position counter’s value.

The position counter (POS or UPOS, LPOS) value is loaded to hold position (POSH or UPOSH, LPOSH) when:

a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read.

b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter’s value.

static inline uint32_t EQDC_GetHoldPosition1(EQDC_Type *base)

Get the hold position counter1’s value.

The Upper Position Counter Hold Register 1(UPOSH1) shares the same address with UCOMP1. When read, this register means the value of UPOSH1, which is the upper 16 bits of POSH1. The Lower Position Counter Hold Register 1(LPOSH1) shares the same address

with LCOMP1. When read, this register means the value of LPOSH1, which is the lower 16 bits of POSH1. Position counter is captured into POSH1 on the rising edge of ICAP[1].

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter1's value.

static inline uint32_t EQDC_GetHoldPosition2(EQDC_Type *base)

Get the hold position counter2's value.

The Upper Position Counter Hold Register 2(UPOSH2) shares the same address with UCOMP2. When read,this register means the value of UPOSH2, which is the upper 16 bits of POSH2. The Lower Position Counter Hold Register 2(LPOSH2) shares the same address with LCOMP2. When read, this register means the value of LPOSH2, which is the lower 16 bits of POSH2. Position counter is captured into POSH2 on the rising edge of ICAP[2].

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter2's value.

static inline uint32_t EQDC_GetHoldPosition3(EQDC_Type *base)

Get the hold position counter3's value.

The Upper Position Counter Hold Register 3(UPOSH3) shares the same address with UCOMP3. When read,this register means the value of UPOSH3, which is the upper 16 bits of POSH3. The Lower Position Counter Hold Register 3(LPOSH3) shares the same address with LCOMP3. When read, this register means the value of LPOSH3, which is the lower 16 bits of POSH3. Position counter is captured into POSH3 on the rising edge of ICAP[3].

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter3's value.

static inline uint16_t EQDC_GetPositionDifference(EQDC_Type *base)

Get the position difference counter's value.

Parameters

- base – EQDC peripheral base address.

Returns

The position difference counter's value.

static inline uint16_t EQDC_GetHoldPositionDifference(EQDC_Type *base)

Get the hold position difference counter's value.

The position difference (POSD) value is loaded to hold position difference (POSDH) when:

a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read. When Period Measurement is enabled (CTRL3[PMEN] = 1), POSDH will only be udpated when reading POSD.

b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

Parameters

- base – EQDC peripheral base address.

Returns

Hold position difference counter's value.

static inline uint16_t EQDC_GetRevolution(EQDC_Type *base)

Get the revolution counter's value.

Get the revolution counter (REV) value.

Parameters

- base – EQDC peripheral base address.

Returns

The revolution counter's value.

static inline uint16_t EQDC_GetHoldRevolution(EQDC_Type *base)

Get the hold revolution counter's value.

The revolution counter (REV) value is loaded to hold revolution (REVH) when:

a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read.

b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

Parameters

- base – EQDC peripheral base address.

Returns

Hold position revolution counter's value.

static inline uint16_t EQDC_GetLastEdgeTime(EQDC_Type *base)

Get the last edge time.

Last edge time (LASTEDGE) is the time since the last edge occurred on PHASEA or PHASEB. The last edge time register counts up using the peripheral clock after prescaler. Any edge on PHASEA or PHASEB will reset this register to 0 and start counting. If the last edge timer count reaches 0xffff, the counting will stop in order to prevent an overflow.Counting will continue when an edge occurs on PHASEA or PHASEB.

Parameters

- base – EQDC peripheral base address.

Returns

The last edge time.

static inline uint16_t EQDC_GetHoldLastEdgeTime(EQDC_Type *base)

Get the hold last edge time.

The hold of last edge time(LASTEDGEH) is update to last edge time(LASTEDGE) when the position difference register register (POSD) is read.

Parameters

- base – EQDC peripheral base address.

Returns

Hold of last edge time.

static inline uint16_t EQDC_GetPositionDifferencePeriod(EQDC_Type *base)

Get the Position Difference Period counter value.

The Position Difference Period counter (POSDPER) counts up using the prescaled peripheral clock. When reading the position difference register(POSD), the last edge time (LASTEDGE) will be loaded to position difference period counter(POSDPER). If the POSDPER count reaches 0xffff, the counting will stop in order to prevent an overflow. Counting will continue when an edge occurs on PHASEA or PHASEB.

Parameters

- base – EQDC peripheral base address.

Returns
The position difference period counter value.

static inline uint16_t EQDC_GetBufferedPositionDifferencePeriod(EQDC_Type *base)

Get buffered Position Difference Period counter value.

The Bufferd Position Difference Period (POSDPERBFR) value is updated with the position difference period counter(POSDPER) when any edge occurs on PHASEA or PHASEB.

Parameters

- base – EQDC peripheral base address.

Returns
The buffered position difference period counter value.

static inline uint16_t EQDC_GetHoldPositionDifferencePeriod(EQDC_Type *base)

Get Hold Position Difference Period counter value.

The hold position difference period(POSDPERH) is updated with the value of buffered position difference period(POSDPERBFR) when the position difference(POSD) register is read.

Parameters

- base – EQDC peripheral base address.

Returns
The hold position difference period counter value.

enum _eqdc_status_flags

EQDC status flags, these flags indicate the counter's events. .

Values:

enumerator kEQDC_HomeEnableTransitionFlag
HOME/ENABLE signal transition occured.

enumerator kEQDC_IndexPresetPulseFlag
INDEX/PRESET pulse occured.

enumerator kEQDC_WatchdogTimeoutFlag
Watchdog timeout occured.

enumerator kEQDC_SimultPhaseChangeFlag
Simultaneous change of PHASEA and PHASEB occured.

enumerator kEQDC_CountDirectionChangeFlag
Count direction change interrupt enable.

enumerator kEQDC_PositionRollOverFlag
Position counter rolls over from 0xFFFFFFFF to 0, or from MOD value to INIT value.

enumerator kEQDC_PositionRollUnderFlag
Position register roll under from 0 to 0xFFFFFFFF, or from INIT value to MOD value.

enumerator kEQDC_PositionCompare0Flag
Position counter match the COMP0 value.

enumerator kEQDC_PositionCompare1Flag
Position counter match the COMP1 value.

enumerator kEQDC_PositionCompare2Flag
Position counter match the COMP2 value.

enumerator kEQDC_PositionCompare3Flag
Position counter match the COMP3 value.

enumerator kEQDC_StatusAllFlags

enum _eqdc_signal_status
Signal status, these flags indicate the raw and filtered input signal status. .

Values:

enumerator kEQDC_SignalStatusRawHomeEnable
Raw HOME/ENABLE input.

enumerator kEQDC_SignalStatusRawIndexPreset
Raw INDEX/PRESET input.

enumerator kEQDC_SignalStatusRawPhaseB
Raw PHASEB input.

enumerator kEQDC_SignalStatusRawPhaseA
Raw PHASEA input.

enumerator kEQDC_SignalStatusFilteredHomeEnable
The filtered HOME/ENABLE input.

enumerator kEQDC_SignalStatusFilteredIndexPreset
The filtered INDEX/PRESET input.

enumerator kEQDC_SignalStatusFilteredPhaseB
The filtered PHASEB input.

enumerator kEQDC_SignalStatusFilteredPhaseA
The filtered PHASEA input.

enumerator kEQDC_SignalStatusPositionCompare0Flag
Position Compare 0 Flag Output.

enumerator kEQDC_SignalStatusPositionCompare1Flag
Position Compare 1 Flag Output.

enumerator kEQDC_SignalStatusPositionCompare2Flag
Position Compare 2 Flag Output.

enumerator kEQDC_SignalStatusPositionCompare3Flag
Position Compare 3 Flag Output.

enumerator kEQDC_SignalStatusCountDirectionFlagHold
Count Direction Flag Hold.

enumerator kEQDC_SignalStatusCountDirectionFlag
Count Direction Flag Output.

enumerator kEQDC_SignalStatusAllFlags

enum _eqdc_interrupt_enable
Interrupt enable/disable mask. .

Values:

enumerator kEQDC_HomeEnableTransitionInterruptEnable
HOME/ENABLE signal transition interrupt enable.

enumerator kEQDC_IndexPresetPulseInterruptEnable
INDEX/PRESET pulse interrupt enable.

enumerator kEQDC_WatchdogTimeoutInterruptEnable
Watchdog timeout interrupt enable.

enumerator kEQDC_SimultPhaseChangeInterruptEnable
Simultaneous PHASEA and PHASEB change interrupt enable.

enumerator kEQDC_CountDirectionChangeInterruptEnable
Count direction change interrupt enable.

enumerator kEQDC_PositionRollOverInterruptEnable
Roll-over interrupt enable.

enumerator kEQDC_PositionRollUnderInterruptEnable
Roll-under interrupt enable.

enumerator kEQDC_AllInterruptEnable

enum _eqdc_home_enable_init_pos_counter_mode
Define HOME/ENABLE signal's trigger mode.

Values:

enumerator kEQDC_HomeInitPosCounterDisabled
Don't use HOME/ENABLE signal to initialize the position counter.

enumerator kEQDC_HomeInitPosCounterOnRisingEdge
Use positive going edge to trigger initialization of position counters.

enumerator kEQDC_HomeInitPosCounterOnFallingEdge
Use negative going edge to trigger initialization of position counters.

enum _eqdc_index_preset_init_pos_counter_mode
Define INDEX/PRESET signal's trigger mode.

Values:

enumerator kEQDC_IndexInitPosCounterDisabled
INDEX/PRESET pulse does not initialize the position counter.

enumerator kEQDC_IndexInitPosCounterOnRisingEdge
Use INDEX/PRESET pulse rising edge to initialize position counter.

enumerator kEQDC_IndexInitPosCounterOnFallingEdge
Use INDEX/PRESET pulse falling edge to initialize position counter.

enum _eqdc_operate_mode
Define type for decoder opertion mode.

The Quadrature Decoder operates in following 4 operation modes: 1.Quadrature Decode(QDC) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 0) In QDC operation mode, Module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 2.Quadrature Count(QCT) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 1) In QCT operation mode, Module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor. 3.Single Phase Decode(PH1DC) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 0) In PH1DC operation mode, the module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 4.Single Phase Count(PH1CT) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 1) In PH1CT operation mode, the module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor.

Values:

enumerator kEQDC_QuadratureDecodeOperationMode
Use standard quadrature decoder with PHASEA/PHASEB, INDEX/HOME.

enumerator kEQDC_QuadratureCountOperationMode
Use quadrature count operation mode with PHASEA/PHASEB, PRESET/ENABLE.

enumerator kEQDC_SinglePhaseDecodeOperationMode
Use single phase quadrature decoder with PHASEA/PHASEB, INDEX/HOME.

enumerator kEQDC_SinglePhaseCountOperationMode
Use single phase count decoder with PHASEA/PHASEB, PRESET/ENABLE.

enum _eqdc_count_mode
Define type for decoder count mode.

In decode mode, it uses the standard quadrature decoder with PHASEA and PHASEB, PHASEA = 0 and PHASEB = 0 mean reverse direction.

- If PHASEA leads PHASEB, then motion is in the positive direction.
- If PHASEA trails PHASEB,then motion is in the negative direction. In single phase mode, there are three count modes:
- In Signed Count mode (Single Edge). Both position counter (POS) and position difference counter (POSD) count on the input PHASEA rising edge while the input PHASEB provides the selected position counter direction (up/down). If CTRL[REV] is 1, then the position counter will count in the opposite direction.
- In Signed Count mode (double edge), both position counter (POS) and position difference counter (POSD) count the input PHASEA on both rising edge and falling edge while the input PHASEB provides the selected position counter direction (up/down).
- In UP/DOWN Pulse Count mode. Both position counter (POS) and position difference counter (POSD) count in the up direction when input PHASEA rising edge occurs. Both counters count in the down direction when input PHASEB rising edge occurs. If CTRL[REV] is 1, then the position counter will count in the opposite direction.

Values:

enumerator kEQDC_QuadratureX4
Active on kEQDC_QuadratureDecodeOperationMode/kEQDC_QuadratureCountOperationMode.

enumerator kEQDC_QuadratureX2
Active on kEQDC_QuadratureDecodeOperationMode/kEQDC_QuadratureCountOperationMode.

enumerator kEQDC_QuadratureX1
Active on kEQDC_QuadratureDecodeOperationMode/kEQDC_QuadratureCountOperationMode.

enumerator kEQDC_UpDownPulseCount
Active on kEQDC_SinglePhaseDecodeOperationMode/kEQDC_SinglePhaseCountOperationMode.

enumerator kEQDC_SignedCountDoubleEdge
Active on kEQDC_SinglePhaseDecodeOperationMode/kEQDC_SinglePhaseCountOperationMode.

enumerator kEQDC_SignedCountSingleEdge
Active on kEQDC_SinglePhaseDecodeOperationMode/kEQDC_SinglePhaseCountOperationMode.

enum _eqdc_output_pulse_mode
Define type for the condition of POSMATCH pulses.

Values:

enumerator kEQDC_OutputPulseOnCounterEqualCompare
POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (UCOMPx/LCOMPx)(x range is 0-3).

enumerator kEQDC_OutputPulseOnReadingPositionCounter

POSMATCH pulses when reading position counter(POS and LPOS), revolution counter(REV), position difference counter(POSD).

enum _eqdc_revolution_count_condition

Define type for determining how the revolution counter (REV) is incremented/decremented.

Values:

enumerator kEQDC_RevolutionCountOnIndexPulse

Use INDEX pulse to increment/decrement revolution counter.

enumerator kEQDC_RevolutionCountOnRollOverModulus

Use modulus counting roll-over/under to increment/decrement revolution counter.

enum _eqdc_filter_sample_count

Input Filter Sample Count.

The Input Filter Sample Count represents the number of consecutive samples that must agree, before the input filter accepts an input transition

Values:

enumerator kEQDC_Filter3Samples

3 samples.

enumerator kEQDC_Filter4Samples

4 samples.

enumerator kEQDC_Filter5Samples

5 samples.

enumerator kEQDC_Filter6Samples

6 samples.

enumerator kEQDC_Filter7Samples

7 samples.

enumerator kEQDC_Filter8Samples

8 samples.

enumerator kEQDC_Filter9Samples

9 samples.

enumerator kEQDC_Filter10Samples

10 samples.

enum _eqdc_count_direction_flag

Count direction.

Values:

enumerator kEQDC_CountDirectionDown

Last count was in down direction.

enumerator kEQDC_CountDirectionUp

Last count was in up direction.

enum _eqdc_prescaler

Prescaler used by Last Edge Time (LASTEDGE) and Position Difference Period Counter (POSDPER).

Values:

enumerator kEQDC_Prescaler1
Prescaler value 1.

enumerator kEQDC_Prescaler2
Prescaler value 2.

enumerator kEQDC_Prescaler4
Prescaler value 4.

enumerator kEQDC_Prescaler8
Prescaler value 8.

enumerator kEQDC_Prescaler16
Prescaler value 16.

enumerator kEQDC_Prescaler32
Prescaler value 32.

enumerator kEQDC_Prescaler64
Prescaler value 64.

enumerator kEQDC_Prescaler128
Prescaler value 128.

enumerator kEQDC_Prescaler256
Prescaler value 256.

enumerator kEQDC_Prescaler512
Prescaler value 512.

enumerator kEQDC_Prescaler1024
Prescaler value 1024.

enumerator kEQDC_Prescaler2048
Prescaler value 2048.

enumerator kEQDC_Prescaler4096
Prescaler value 4096.

enumerator kEQDC_Prescaler8192
Prescaler value 8192.

enumerator kEQDC_Prescaler16384
Prescaler value 16384.

enumerator kEQDC_Prescaler32768
Prescaler value 32768.

typedef enum *_eqdc_home_enable_init_pos_counter_mode*
eqdc_home_enable_init_pos_counter_mode_t

Define HOME/ENABLE signal's trigger mode.

typedef enum *_eqdc_index_preset_init_pos_counter_mode*
eqdc_index_preset_init_pos_counter_mode_t

Define INDEX/PRESET signal's trigger mode.

typedef enum *_eqdc_operate_mode* eqdc_operate_mode_t

Define type for decoder opertion mode.

The Quadrature Decoder operates in following 4 operation modes: 1.Quadrature Decode(QDC) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 0) In QDC operation mode, Module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the

PHASEA and PHASEB signals from Speed/Position sensor. 2.Quadrature Count(QCT) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 1) In QCT operation mode, Module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor. 3.Single Phase Decode(PH1DC) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 0) In PH1DC operation mode, the module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 4.Single Phase Count(PH1CT) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 1) In PH1CT operation mode, the module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor.

typedef enum *_eqdc_count_mode* eqdc_count_mode_t

Define type for decoder count mode.

In decode mode, it uses the standard quadrature decoder with PHASEA and PHASEB, PHASEA = 0 and PHASEB = 0 mean reverse direction.

- If PHASEA leads PHASEB, then motion is in the positive direction.
- If PHASEA trails PHASEB,then motion is in the negative direction. In single phase mode, there are three count modes:
- In Signed Count mode (Single Edge). Both position counter (POS) and position difference counter (POSD) count on the input PHASEA rising edge while the input PHASEB provides the selected position counter direction (up/down). If CTRL[REV] is 1, then the position counter will count in the opposite direction.
- In Signed Count mode (double edge), both position counter (POS) and position difference counter (POSD) count the input PHASEA on both rising edge and falling edge while the input PHASEB provides the selected position counter direction (up/down).
- In UP/DOWN Pulse Count mode. Both position counter (POS) and position difference counter (POSD) count in the up direction when input PHASEA rising edge occurs. Both counters count in the down direction when input PHASEB rising edge occurs. If CTRL[REV] is 1, then the position counter will count in the opposite direction.

typedef enum *_eqdc_output_pulse_mode* eqdc_output_pulse_mode_t

Define type for the condition of POSMATCH pulses.

typedef enum *_eqdc_revolution_count_condition* eqdc_revolution_count_condition_t

Define type for determining how the revolution counter (REV) is incremented/decremented.

typedef enum *_eqdc_filter_sample_count* eqdc_filter_sample_count_t

Input Filter Sample Count.

The Input Filter Sample Count represents the number of consecutive samples that must agree, before the input filter accepts an input transition

typedef enum *_eqdc_count_direction_flag* eqdc_count_direction_flag_t

Count direction.

typedef enum *_eqdc_prescaler* eqdc_prescaler_t

Prescaler used by Last Edge Time (LASTEDGE) and Position Difference Period Counter (POSDPER).

typedef struct *_eqdc_config* eqdc_config_t

Define user configuration structure for EQDC module.

FSL_EQDC_DRIVER_VERSION

EQDC_CTRL_W1C_FLAGS

W1C bits in EQDC CTRL registers.

EQDC_INTCTRL_W1C_FLAGS
W1C bits in EQDC INTCTRL registers.

EQDC_CTRL_INT_EN
Interrupt enable bits in EQDC CTRL registers.

EQDC_INTCTRL_INT_EN
Interrupt enable bits in EQDC INTCTRL registers.

EQDC_CTRL_INT_FLAGS
Interrupt flag bits in EQDC CTRL registers.

EQDC_INTCTRL_INT_FLAGS
Interrupt flag bits in EQDC INTCTRL registers.

struct _eqdc_config
#include <fsl_eqdc.h> Define user configuration structure for EQDC module.

Public Members

bool enableReverseDirection
Enable reverse direction counting.

bool countOnce
Selects modulo loop or one shot counting mode.

bool enableDma
Enable DMA for new written buffer values of COMPx/INIT/MOD(x range is 0-3)

bool bufferedRegisterLoadMode
selects the loading time point of the buffered compare registers UCOMPx/LCOMPx, x=0~3, initial register (UINIT/LINIT), and modulus register (UMOD/LMOD).

bool enableTriggerInitPositionCounter
Initialize position counter with initial register(UINIT, LINIT) value on TRIGGER's rising edge.

bool enableIndexInitPositionCounter
Enables the feature that the position counter to be initialized by Index Event Edge Mark.

This option works together with _eqdc_index_preset_init_pos_counter_mode and enableReverseDirection; If enabled, the behavior is like this:

When PHA leads PHB (Clockwise): If _eqdc_index_preset_init_pos_counter_mode is kEQDC_IndexInitPosCounterOnRisingEdge, then INDEX rising edge reset position counter. If _eqdc_index_preset_init_pos_counter_mode is kEQDC_IndexInitPosCounterOnFallingEdge, then INDEX falling edge reset position counter. If enableReverseDirection is false, then Reset position counter to initial value. If enableReverseDirection is true, then reset position counter to modulus value.

When PHA lags PHB (Counter Clockwise): If _eqdc_index_preset_init_pos_counter_mode is kEQDC_IndexInitPosCounterOnRisingEdge, then INDEX falling edge reset position counter. If _eqdc_index_preset_init_pos_counter_mode is kEQDC_IndexInitPosCounterOnFallingEdge, then INDEX rising edge reset position counter. If enableReverseDirection is false, then Reset position counter to modulus value. If enableReverseDirection is true, then reset position counter to initial value.

bool enableTriggerClearPositionRegisters
Clear position counter(POS), revolution counter(REV), position difference counter (POSD) on TRIGGER's rising edge.

bool enableTriggerHoldPositionRegisters

Load position counter(POS), revolution counter(REV), position difference counter (POSD) values to hold registers on TRIGGER's rising edge.

bool filterPhaseA

Filter operation on PHASEA input, when write 1, it means filter for PHASEA input is bypassed.

bool filterPhaseB

Filter operation on PHASEB input, when write 1, it means filter for PHASEB input is bypassed.

bool filterIndPre

Filter operation on INDEX/PRESET input, when write 1, it means filter for INDEX/PRESET input is bypassed.

bool filterHomEna

Filter operation on HOME/ENABLE input, when write 1, it means filter for HOME/ENABLE input is bypassed.

bool enableWatchdog

Enable the watchdog to detect if the target is moving or not.

uint16_t watchdogTimeoutValue

Watchdog timeout count value. It stores the timeout count for the quadrature decoder module watchdog timer.

eqdc_prescaler_t prescaler

Prescaler.

bool filterClockSourceselection

Filter Clock Source selection.

eqdc_filter_sample_count_t filterSampleCount

Input Filter Sample Count. This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition.

uint8_t filterSamplePeriod

Input Filter Sample Period. This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available range is 0 - 255.

eqdc_operate_mode_t operateMode

Selects operation mode.

eqdc_count_mode_t countMode

Selects count mode.

eqdc_home_enable_init_pos_counter_mode_t homeEnableInitPosCounterMode

Select how HOME/Enable signal used to initialize position counters.

eqdc_index_preset_init_pos_counter_mode_t indexPresetInitPosCounterMode

Select how INDEX/Preset signal used to initialize position counters.

eqdc_output_pulse_mode_t outputPulseMode

The condition of POSMATCH pulses.

uint32_t positionCompareValue[4]

Position compare 0 ~ 3 value. The available value is a 32-bit number.

eqdc_revolution_count_condition_t revolutionCountCondition

Revolution Counter Modulus Enable.

uint32_t positionModulusValue

Position modulus value. The available value is a 32-bit number.

uint32_t positionInitialValue

Position initial value. The available value is a 32-bit number.

uint32_t positionCounterValue

Position counter value. When Modulo mode enabled, the positionCounterValue should be in the range of positionInitialValue and positionModulusValue.

bool enablePeriodMeasurement

Enable period measurement. When enabled, the position difference hold register (POSDH) is only updated when position difference register (POSD) is read.

uint16_t enabledInterruptsMask

Mask of interrupts to be enabled, should be OR'ed value of _eqdc_interrupt_enable.

2.13 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const *ewm_config_t* *config)

Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

ewm_config_t

Parameters

- config – Pointer to the EWM configuration structure.

static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to disable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

_ewm_status_flags_t

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns
State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)
Services the EWM.

This function resets the EWM counter to zero.

Parameters

- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION
EWM driver version 2.0.4.

enum _ewm_lpo_clock_source
Describes EWM clock source.

Values:

enumerator kEWM_LpoClockSource0
EWM clock sourced from lpo_clk[0]

enumerator kEWM_LpoClockSource1
EWM clock sourced from lpo_clk[1]

enumerator kEWM_LpoClockSource2
EWM clock sourced from lpo_clk[2]

enumerator kEWM_LpoClockSource3
EWM clock sourced from lpo_clk[3]

enum _ewm_interrupt_enable_t
EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable
Enable the EWM to generate an interrupt

enum _ewm_status_flags_t
EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator kEWM_RunningFlag
Running flag, set when EWM is enabled

typedef enum *_ewm_lpo_clock_source* ewm_lpo_clock_source_t
Describes EWM clock source.

typedef struct *_ewm_config* ewm_config_t
Data structure for EWM configuration.

This structure is used to configure the EWM.

struct _ewm_config
#include <fsl_ewm.h> Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

bool enableEwm
: Enable EWM module

bool enableEwmInput
: Enable EWM_in input

bool setInputAssertLogic
: EWM_in signal assertion state

bool enableInterrupt
: Enable EWM interrupt

ewm_lpo_clock_source_t clockSource
: Clock source select

uint8_t prescaler
: Clock prescaler value

uint8_t compareLowValue
: Compare low-register value

uint8_t compareHighValue
: Compare high-register value

2.14 FGPIO Driver

2.15 FlexCAN: Flex Controller Area Network Driver

2.16 FlexCAN Driver

bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)
: Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN_FDInit to enable CAN FD mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
: return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)
: Get Mailbox offset number by dword.

  This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080.

For 32 bytes payload frame, the second mailbox is ((1/12)*512 + 1%12*40)/4 = 10, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – Mailbox index.

Returns
Mailbox address offset in word.

status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

uint32_t FLEXCAN_GetInstance(CAN_Type *base)

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns
FlexCAN instance.

bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns
TRUE if timing configuration found, FALSE if failed to find configuration.

void FLEXCAN_Init(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                 = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate                = 1000000U;
flexcanConfig.maxMbNum               = 16;
flexcanConfig.enableLoopBack         = false;
flexcanConfig.enableSelfWakeup       = false;
flexcanConfig.enableIndividMask      = false;
flexcanConfig.enableDoze             = false;
flexcanConfig.disableSelfReception   = false;
flexcanConfig.enableListenOnlyMode   = false;
flexcanConfig.timingConfig           = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 40000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t bitRateFD, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.
- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

void FLEXCAN_FDInit(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz, *flexcan_mb_size_t* dataSize, bool brs)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                 = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate                = 1000000U;
```

(continues on next page)

(continued from previous page)

```
flexcanConfig.bitRateFD          = 2000000U;
flexcanConfig.maxMbNum           = 16;
flexcanConfig.enableLoopBack     = false;
flexcanConfig.enableSelfWakeup   = false;
flexcanConfig.enableIndividMask  = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze         = false;
flexcanConfig.timingConfig       = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 80000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

void FLEXCAN_Deinit(CAN_Type *base)

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN_GetDefaultConfig(*flexcan_config_t* *pConfig)

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enablePretendedeNetworking = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian; flexcanConfig.timingConfig = timingConfig;

Parameters

- pConfig – Pointer to the FlexCAN configuration structure.

void FLEXCAN_SetTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetBitRate() instead.

Note: Calling FLEXCAN_SetTimingConfig() overrides the bit rate set in FLEXCAN_Init() or FLEXCAN_SetBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on FLEXCAN_CalculateImprovedTimingValues() API calculated timing values.

Note: Calling FLEXCAN_SetBitRate() overrides the bit rate set in FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRate_Bps – Bit rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only Nominal phase) successfully.

void FLEXCAN_SetFDTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetFDBitRate() to set both Nominal/Data bit Rate instead.

Note: Calling FLEXCAN_SetFDTimingConfig() overrides the data phase bit rate set in FLEXCAN_FDInit()/FLEXCAN_SetFDBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRateN_Bps, uint32_t bitRateD_Bps)

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on FLEXCAN_FDCalculateImprovedTimingValues() API calculated timing values.

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRateN_Bps – Nominal bit Rate in Bps.
- bitRateD_Bps – Data bit Rate in Bps.

Returns

kStatus_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN_SetTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's

ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbEmpty, kFLEXCAN_RxMbFull or kFLEXCAN_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pFrame)

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const *flexcan_rx_fifo_config_t* *pRxFifoConfig, bool enable)

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note: Legacy Rx FIFO only can receive classic CAN message.

Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Legacy Rx FIFO.
  - true: Enable Legacy Rx FIFO.
  - false: Disable Legacy Rx FIFO.

void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const *flexcan_enhanced_rx_fifo_config_t* *pConfig, bool enable)

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

Note: Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

- base – FlexCAN peripheral base address.

- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
  - true: Enable Enhanced Rx FIFO.
  - false: Disable Enhanced Rx FIFO.

void FLEXCAN_SetPNConfig(CAN_Type *base, const *flexcan_pn_config_t* *pConfig)

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators _flexcan_flags. To check the specific status, compare the return value with enumerators in _flexcan_flags.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the _flexcan_flags.

static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of _flexcan_flags.

static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.

- mask – The ORed FlexCAN Message Buffer mask.

Returns
The status of given Message Buffers.

static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns
The status of given Message Buffers.

static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base, *flexcan_memory_error_report_status_t* *errorStatus)

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

Parameters

- base – FlexCAN peripheral base address.

Returns
The number of received wake up msessages.

static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of _flexcan_interrupt_enable.

static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to disable. Logical OR of _flexcan_interrupt_enable.

static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- enable – true to enable, false to disable.

static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN Rx FIFO Head address.

static inline *status_t* FLEXCAN_Enable(CAN_Type *base, bool enable)

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

Parameters

- base – FlexCAN base pointer.
- enable – true to enable, false to disable.

Returns

kStatus_Success Enable FlexCAN module successful kStatus_Timeout Timeout when wait for Low-Power Mode Acknowledge

status_t FLEXCAN_WriteTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pTxFrame)

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.

status_t FLEXCAN_WriteFDTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_fd_frame_t* *pTxFrame)

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadFDRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.

status_t FLEXCAN_ReadRxFifo(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadEnhancedRxFifo(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadPNWakeUpMB(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.
- mbIdx – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

Return values

- kStatus_Success – - Read Message from Wake up Message Buffer successfully.
- kStatus_Fail – - Wake up Message Buffer has no valid content.

status_t FLEXCAN_TransferFDSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferFDReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferFDSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferFDReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.

- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

void FLEXCAN_TransferFDAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

void FLEXCAN_TransferFDAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

status_t FLEXCAN_TransferSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveFifoBlocking(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveEnhancedFifoBlocking(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

void FLEXCAN_TransferCreateHandle(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_transfer_callback_t* callback, void *userData)

Initializes the FlexCAN handle.

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t FLEXCAN_TransferSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the flexcan_fifo_transfer_t.

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

status_t FLEXCAN_TransferGetReceiveFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

status_t FLEXCAN_TransferReceiveEnhancedFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the ref flexcan_fifo_transfer_t.@

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

uint32_t FLEXCAN_GetTimeStamp(*flexcan_handle_t* *handle, uint8_t mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback

function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

static inline uint32_t FLEXCAN_GetHighResolutionTimeStamp(CAN_Type *base, uint8_t mbIdx)

void FLEXCAN_TransferAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferAbortReceiveEnhancedFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Enhanced Rx FIFO process.

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MbHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle, uint32_t startMbIdx, uint32_t endMbIdx)

FlexCAN Message Buffer IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

void FLEXCAN_EhancedRxFifoHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Ehanced Rx FIFO IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_BusoffErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Bus Off, Error and Warning IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_PNWakeUpHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Pretended Networking Wake-up IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MemoryErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Memory Error IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

FSL_FLEXCAN_DRIVER_VERSION

FlexCAN driver version.

FlexCAN transfer status.

Values:

enumerator kStatus_FLEXCAN_TxBusy

Tx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_TxIdle

Tx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_TxSwitchToRx

Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus_FLEXCAN_RxBusy
Rx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_RxIdle
Rx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_RxOverflow
Rx Message Buffer is Overflowed.

enumerator kStatus_FLEXCAN_RxFifoBusy
Rx Message FIFO is Busy.

enumerator kStatus_FLEXCAN_RxFifoIdle
Rx Message FIFO is Idle.

enumerator kStatus_FLEXCAN_RxFifoOverflow
Rx Message FIFO is overflowed.

enumerator kStatus_FLEXCAN_RxFifoWarning
Rx Message FIFO is almost overflowed.

enumerator kStatus_FLEXCAN_RxFifoDisabled
Rx Message FIFO is disabled during reading.

enumerator kStatus_FLEXCAN_ErrorStatus
FlexCAN Module Error and Status.

enumerator kStatus_FLEXCAN_WakeUp
FlexCAN is waken up from STOP mode.

enumerator kStatus_FLEXCAN_UnHandled
UnHadled Interrupt asserted.

enumerator kStatus_FLEXCAN_RxRemote
Rx Remote Message Received in Mail box.

enumerator kStatus_FLEXCAN_RxFifoUnderflow
Enhanced Rx Message FIFO is underflow.

enumerator kStatus_FLEXCAN_MemoryError
FlexCAN Memory Error.

enum _flexcan_frame_format
FlexCAN frame format.

Values:

enumerator kFLEXCAN_FrameFormatStandard
Standard frame format attribute.

enumerator kFLEXCAN_FrameFormatExtend
Extend frame format attribute.

enum _flexcan_frame_type
FlexCAN frame type.

Values:

enumerator kFLEXCAN_FrameTypeData
Data frame type attribute.

enumerator kFLEXCAN_FrameTypeRemote
Remote frame type attribute.

enum _flexcan_clock_source

FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Values:

enumerator kFLEXCAN_ClkSrcOsc

FlexCAN Protocol Engine clock from Oscillator.

enumerator kFLEXCAN_ClkSrcPeri

FlexCAN Protocol Engine clock from Peripheral Clock.

enumerator kFLEXCAN_ClkSrc0

FlexCAN Protocol Engine clock selected by user as SRC == 0.

enumerator kFLEXCAN_ClkSrc1

FlexCAN Protocol Engine clock selected by user as SRC == 1.

enum _flexcan_wake_up_source

FlexCAN wake up source.

Values:

enumerator kFLEXCAN_WakeupSrcUnfiltered

FlexCAN uses unfiltered Rx input to detect edge.

enumerator kFLEXCAN_WakeupSrcFiltered

FlexCAN uses filtered Rx input to detect edge.

enum _flexcan_endianness

FlexCAN payload endianness.

Values:

enumerator kFLEXCAN_bigEndian

Transmit frame with MSB first, receive frame with big-endian format.

enumerator kFLEXCAN_littleEndian

Transmit frame with LSB first, receive frame with little-endian format.

enum _flexcan_MB_timestamp_base

FlexCAN timebase used for capturing 16-bit TIME_STAMP field of message buffer.

Values:

enumerator kFLEXCAN_CANTimer

FlexCAN free-running timer.

enumerator kFLEXCAN_Lower16bitsHRTimer

Lower 16 bits of high-resolution on-chip timer.

enumerator kFLEXCAN_Upper16bitsHRTimer

Upper 16 bits of high-resolution on-chip timer.

enum _flexcan_capture_point

FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

Values:

enumerator kFLEXCAN_CANFrameID2ndBit

Second bit of identifier field of any frame is on the CAN bus. HR_TIME_STAMPn register will not capture 32-bit counter value.

enumerator kFLEXCAN_CANFrameEnd

End of the CAN frame.

enumerator kFLEXCAN_CANFrameStart

Start of the CAN frame.

enumerator kFLEXCAN_CANFDFrameRes

Start of frame for classical CAN frames; res bit for CAN FD frames.

enum _flexcan_rx_fifo_filter_type

FlexCAN Rx Fifo Filter type.

Values:

enumerator kFLEXCAN_RxFifoFilterTypeA

One full ID (standard and extended) per ID Filter element.

enumerator kFLEXCAN_RxFifoFilterTypeB

Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeC

Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeD

All frames rejected.

enum _flexcan_mb_size

FlexCAN Message Buffer Payload size.

Values:

enumerator kFLEXCAN_8BperMB

Selects 8 bytes per Message Buffer.

enumerator kFLEXCAN_16BperMB

Selects 16 bytes per Message Buffer.

enumerator kFLEXCAN_32BperMB

Selects 32 bytes per Message Buffer.

enumerator kFLEXCAN_64BperMB

Selects 64 bytes per Message Buffer.

enum _flexcan_fd_frame_length

FlexCAN CAN FD frame supporting data length (available DLC values).

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

Values:

enumerator kFLEXCAN_0BperFrame

Frame contains 0 valid data bytes.

enumerator kFLEXCAN_1BperFrame

Frame contains 1 valid data bytes.

enumerator kFLEXCAN_2BperFrame
Frame contains 2 valid data bytes.

enumerator kFLEXCAN_3BperFrame
Frame contains 3 valid data bytes.

enumerator kFLEXCAN_4BperFrame
Frame contains 4 valid data bytes.

enumerator kFLEXCAN_5BperFrame
Frame contains 5 valid data bytes.

enumerator kFLEXCAN_6BperFrame
Frame contains 6 valid data bytes.

enumerator kFLEXCAN_7BperFrame
Frame contains 7 valid data bytes.

enumerator kFLEXCAN_8BperFrame
Frame contains 8 valid data bytes.

enumerator kFLEXCAN_12BperFrame
Frame contains 12 valid data bytes.

enumerator kFLEXCAN_16BperFrame
Frame contains 16 valid data bytes.

enumerator kFLEXCAN_20BperFrame
Frame contains 20 valid data bytes.

enumerator kFLEXCAN_24BperFrame
Frame contains 24 valid data bytes.

enumerator kFLEXCAN_32BperFrame
Frame contains 32 valid data bytes.

enumerator kFLEXCAN_48BperFrame
Frame contains 48 valid data bytes.

enumerator kFLEXCAN_64BperFrame
Frame contains 64 valid data bytes.

enum _flexcan_efifo_dma_per_read_length
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

Values:

enumerator kFLEXCAN_1WordPerRead
Transfer 1 32-bit words (CS).

enumerator kFLEXCAN_2WordPerRead
Transfer 2 32-bit words (CS + ID).

enumerator kFLEXCAN_3WordPerRead
Transfer 3 32-bit words (CS + ID + 1~4 bytes data).

enumerator kFLEXCAN_4WordPerRead
Transfer 4 32-bit words (CS + ID + 5~8 bytes data).

enumerator kFLEXCAN_5WordPerRead
Transfer 5 32-bit words (CS + ID + 9~12 bytes data).

enumerator kFLEXCAN_6WordPerRead
Transfer 6 32-bit words (CS + ID + 13~16 bytes data).

enumerator kFLEXCAN_7WordPerRead
Transfer 7 32-bit words (CS + ID + 17~20 bytes data).

enumerator kFLEXCAN_8WordPerRead
Transfer 8 32-bit words (CS + ID + 21~24 bytes data).

enumerator kFLEXCAN_9WordPerRead
Transfer 9 32-bit words (CS + ID + 25~28 bytes data).

enumerator kFLEXCAN_10WordPerRead
Transfer 10 32-bit words (CS + ID + 29~32 bytes data).

enumerator kFLEXCAN_11WordPerRead
Transfer 11 32-bit words (CS + ID + 33~36 bytes data).

enumerator kFLEXCAN_12WordPerRead
Transfer 12 32-bit words (CS + ID + 37~40 bytes data).

enumerator kFLEXCAN_13WordPerRead
Transfer 13 32-bit words (CS + ID + 41~44 bytes data).

enumerator kFLEXCAN_14WordPerRead
Transfer 14 32-bit words (CS + ID + 45~48 bytes data).

enumerator kFLEXCAN_15WordPerRead
Transfer 15 32-bit words (CS + ID + 49~52 bytes data).

enumerator kFLEXCAN_16WordPerRead
Transfer 16 32-bit words (CS + ID + 53~56 bytes data).

enumerator kFLEXCAN_17WordPerRead
Transfer 17 32-bit words (CS + ID + 57~60 bytes data).

enumerator kFLEXCAN_18WordPerRead
Transfer 18 32-bit words (CS + ID + 61~64 bytes data).

enumerator kFLEXCAN_19WordPerRead
Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT).

enumerator kFLEXCAN_20WordPerRead
Transfer 20 32-bit words (CS + ID + 64 bytes data + ID HIT + HR timestamp).

enum _flexcan_rx_fifo_priority
FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

Values:

enumerator kFLEXCAN_RxFifoPrioLow
Matching process start from Rx Message Buffer first.

enumerator kFLEXCAN_RxFifoPrioHigh
Matching process start from Enhanced/Legacy Rx FIFO first.

enum _flexcan_interrupt_enable

FlexCAN interrupt enable enumerations.

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

Note: FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Values:

enumerator kFLEXCAN_BusOffInterruptEnable
Bus Off interrupt, use bit 15.

enumerator kFLEXCAN_ErrorInterruptEnable
CAN Error interrupt, use bit 14.

enumerator kFLEXCAN_TxWarningInterruptEnable
Tx Warning interrupt, use bit 11.

enumerator kFLEXCAN_RxWarningInterruptEnable
Rx Warning interrupt, use bit 10.

enumerator kFLEXCAN_FDErrorInterruptEnable
CAN FD Error interrupt, use bit 31.

enumerator kFLEXCAN_PNMatchWakeUpInterruptEnable
PN Match Wake Up interrupt, use high word bit 17.

enumerator kFLEXCAN_PNTimeoutWakeUpInterruptEnable
PN Timeout Wake Up interrupt, use high word bit 16. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

enumerator kFLEXCAN_ERxFifoUnderflowInterruptEnable
Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

enumerator kFLEXCAN_ERxFifoOverflowInterruptEnable
Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

enumerator kFLEXCAN_ERxFifoWatermarkInterruptEnable
Enhanced Rx FIFO Data Avilable interrupt, use high word bit 28.

enumerator kFLEXCAN_ERxFifoDataAvlInterruptEnable

enumerator kFLEXCAN_HostAccessNCErrorInterruptEnable
Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN_FlexCanAccessNCErrorInterruptEnable
FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN_HostOrFlexCanCErrorInterruptEnable
Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum _flexcan_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note: The CPU read action clears the bits corresponding to the FlEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using _flexcan_error_flags enumerations.

Values:

enumerator kFLEXCAN_ErrorOverrunFlag
Error Overrun Status.

enumerator kFLEXCAN_FDErrorIntFlag
CAN FD Error Interrupt Flag.

enumerator kFLEXCAN_BusoffDoneIntFlag
Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN_SynchFlag
CAN Synchronization Status.

enumerator kFLEXCAN_TxWarningIntFlag
Tx Warning Interrupt Flag.

enumerator kFLEXCAN_RxWarningIntFlag
Rx Warning Interrupt Flag.

enumerator kFLEXCAN_IdleFlag
FlexCAN In IDLE Status.

enumerator kFLEXCAN_FaultConfinementFlag
FlexCAN Fault Confinement State.

enumerator kFLEXCAN_TransmittingFlag
FlexCAN In Transmission Status.

enumerator kFLEXCAN_ReceivingFlag
FlexCAN In Reception Status.

enumerator kFLEXCAN_BusOffIntFlag
Bus Off Interrupt Flag.

enumerator kFLEXCAN_ErrorIntFlag
CAN Error Interrupt Flag.

enumerator kFLEXCAN_ErrorFlag

enumerator kFLEXCAN_PNMatchIntFlag
PN Matching Event Interrupt Flag.

enumerator kFLEXCAN_PNTimeoutIntFlag
PN Timeout Event Interrupt Flag.

enumerator kFLEXCAN_ERxFifoUnderflowIntFlag
Enhanced Rx FIFO underflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoOverflowIntFlag
Enhanced Rx FIFO overflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoWatermarkIntFlag
Enhanced Rx FIFO watermark Interrupt Flag.

enumerator kFLEXCAN_ERxFifoDataAvlIntFlag
Enhanced Rx FIFO data available Interrupt Flag.

enumerator kFLEXCAN_ERxFifoEmptyFlag
Enhanced Rx FIFO empty status.

enumerator kFLEXCAN_ERxFifoFullFlag
Enhanced Rx FIFO full status.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorIntFlag
Host Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag
FlexCAN Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_CorrectableErrorIntFlag
Correctable Error Interrupt Flag.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag
Host Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag
FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_CorrectableErrorOverrunFlag
Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_AllMemoryErrorIntFlag
All Memory Error Interrupt Flags.

enumerator kFLEXCAN_AllMemoryErrorFlag
All Memory Error Flags.

enum _flexcan_error_flags
FlexCAN error status flags.

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in _flexcan_flags enumerations to ditermine which error is generated.

Values:

enumerator kFLEXCAN_FDStuffingError
Stuffing Error.

enumerator kFLEXCAN_FDFormError
Form Error.

enumerator kFLEXCAN_FDCrcError
Cyclic Redundancy Check Error.

enumerator kFLEXCAN_FDBit0Error
Unable to send dominant bit.

enumerator kFLEXCAN_FDBit1Error
Unable to send recessive bit.

enumerator kFLEXCAN_TxErrorWarningFlag
Tx Error Warning Status.

enumerator kFLEXCAN_RxErrorWarningFlag
Rx Error Warning Status.

enumerator kFLEXCAN_StuffingError
Stuffing Error.

enumerator kFLEXCAN_FormError
Form Error.

enumerator kFLEXCAN_CrcError
Cyclic Redundancy Check Error.

enumerator kFLEXCAN_AckError
Received no ACK on transmission.

enumerator kFLEXCAN_Bit0Error
Unable to send dominant bit.

enumerator kFLEXCAN_Bit1Error
Unable to send recessive bit.

FlexCAN Legacy Rx FIFO status flags.

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Values:

enumerator kFLEXCAN_RxFifoOverflowFlag
Rx FIFO overflow flag.

enumerator kFLEXCAN_RxFifoWarningFlag
Rx FIFO almost full flag.

enumerator kFLEXCAN_RxFifoFrameAvlFlag
Frames available in Rx FIFO flag.

enum _flexcan_memory_error_type
FlexCAN Memory Error Type.

Values:

enumerator kFLEXCAN_CorrectableError
The memory error is correctable which means on bit error.

enumerator kFLEXCAN_NonCorrectableError
The memory error is non-correctable which means two bit errors.

enum _flexcan_memory_access_type
FlexCAN Memory Access Type.

Values:

enumerator kFLEXCAN_MoveOutFlexCanAccess
The memory error was detected during move-out FlexCAN access.

enumerator kFLEXCAN_MoveInAccess
The memory error was detected during move-in FlexCAN access.

enumerator kFLEXCAN_TxArbitrationAccess
The memory error was detected during Tx Arbitration FlexCAN access.

enumerator kFLEXCAN_RxMatchingAccess
The memory error was detected during Rx Matching FlexCAN access.

enumerator kFLEXCAN_MoveOutHostAccess
The memory error was detected during Rx Matching Host (CPU) access.

enum _flexcan_byte_error_syndrome
FlexCAN Memory Error Byte Syndrome.

Values:

enumerator kFLEXCAN_NoError
No bit error in this byte.

enumerator kFLEXCAN_ParityBits0Error
Parity bit 0 error in this byte.

enumerator kFLEXCAN_ParityBits1Error
Parity bit 1 error in this byte.

enumerator kFLEXCAN_ParityBits2Error
Parity bit 2 error in this byte.

enumerator kFLEXCAN_ParityBits3Error
Parity bit 3 error in this byte.

enumerator kFLEXCAN_ParityBits4Error
Parity bit 4 error in this byte.

enumerator kFLEXCAN_DataBits0Error
Data bit 0 error in this byte.

enumerator kFLEXCAN_DataBits1Error
Data bit 1 error in this byte.

enumerator kFLEXCAN_DataBits2Error
Data bit 2 error in this byte.

enumerator kFLEXCAN_DataBits3Error
Data bit 3 error in this byte.

enumerator kFLEXCAN_DataBits4Error
Data bit 4 error in this byte.

enumerator kFLEXCAN_DataBits5Error
Data bit 5 error in this byte.

enumerator kFLEXCAN_DataBits6Error
Data bit 6 error in this byte.

enumerator kFLEXCAN_DataBits7Error
Data bit 7 error in this byte.

enumerator kFLEXCAN_AllZeroError
All-zeros non-correctable error in this byte.

enumerator kFLEXCAN_AllOneError
All-ones non-correctable error in this byte.

enumerator kFLEXCAN_NonCorrectableErrors
Non-correctable error in this byte.

enum _flexcan_pn_match_source
FlexCAN Pretended Networking match source selection.

Values:

enumerator kFLEXCAN_PNMatSrcID
Message match with ID filtering.

enumerator kFLEXCAN_PNMatSrcIDAndData
Message match with ID filtering and payload filtering.

enum _flexcan_pn_match_mode

FlexCAN Pretended Networking mode match type.

Values:

enumerator kFLEXCAN_PNMatModeEqual

Match upon ID/Payload contents against an exact target value.

enumerator kFLEXCAN_PNMatModeGreater

Match upon an ID/Payload value greater than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeSmaller

Match upon an ID/Payload value smaller than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeRange

Match upon an ID/Payload value inside a range, greater than or equal to a specified lower limit, and smaller than or equal to a specified upper limit

typedef enum *_flexcan_frame_format* flexcan_frame_format_t

FlexCAN frame format.

typedef enum *_flexcan_frame_type* flexcan_frame_type_t

FlexCAN frame type.

typedef enum *_flexcan_clock_source* flexcan_clock_source_t

FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

typedef enum *_flexcan_wake_up_source* flexcan_wake_up_source_t

FlexCAN wake up source.

typedef enum *_flexcan_endianness* flexcan_endianness_t

FlexCAN payload endianness.

typedef enum *_flexcan_MB_timestamp_base* flexcan_MB_timestamp_base_t

FlexCAN timebase used for capturing 16-bit TIME_STAMP field of message buffer.

typedef enum *_flexcan_capture_point* flexcan_capture_point_t

FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

typedef enum *_flexcan_rx_fifo_filter_type* flexcan_rx_fifo_filter_type_t

FlexCAN Rx Fifo Filter type.

typedef enum *_flexcan_mb_size* flexcan_mb_size_t

FlexCAN Message Buffer Payload size.

typedef enum *_flexcan_efifo_dma_per_read_length* flexcan_efifo_dma_per_read_length_t

FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

typedef enum *_flexcan_rx_fifo_priority* flexcan_rx_fifo_priority_t

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

typedef enum *_flexcan_memory_error_type* flexcan_memory_error_type_t

FlexCAN Memory Error Type.

typedef enum *_flexcan_memory_access_type* flexcan_memory_access_type_t

FlexCAN Memory Access Type.

typedef enum *_flexcan_byte_error_syndrome* flexcan_byte_error_syndrome_t

FlexCAN Memory Error Byte Syndrome.

typedef struct *_flexcan_memory_error_report_status* flexcan_memory_error_report_status_t

FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN_GetMemoryErrorReportStatus() function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

typedef struct *_flexcan_frame* flexcan_frame_t

FlexCAN message frame structure.

typedef struct *_flexcan_fd_frame* flexcan_fd_frame_t

CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see _flexcan_fd_frame_length.

typedef struct *_flexcan_timing_config* flexcan_timing_config_t

FlexCAN protocol timing characteristic configuration structure.

typedef struct *_flexcan_config* flexcan_config_t

FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

typedef struct *_flexcan_rx_mb_config* flexcan_rx_mb_config_t

FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

typedef enum *_flexcan_pn_match_source* flexcan_pn_match_source_t

FlexCAN Pretended Networking match source selection.

typedef enum *_flexcan_pn_match_mode* flexcan_pn_match_mode_t

FlexCAN Pretended Networking mode match type.

typedef struct *_flexcan_pn_config* flexcan_pn_config_t

FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

typedef struct *_flexcan_rx_fifo_config* flexcan_rx_fifo_config_t

FlexCAN Legacy Rx FIFO configuration structure.

typedef struct *_flexcan_enhanced_rx_fifo_std_id_filter* flexcan_enhanced_rx_fifo_std_id_filter_t

FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

typedef struct *_flexcan_enhanced_rx_fifo_ext_id_filter* flexcan_enhanced_rx_fifo_ext_id_filter_t
FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

typedef struct *_flexcan_enhanced_rx_fifo_config* flexcan_enhanced_rx_fifo_config_t
FlexCAN Enhanced Rx FIFO configuration structure.

typedef struct *_flexcan_mb_transfer* flexcan_mb_transfer_t
FlexCAN Message Buffer transfer.

typedef struct *_flexcan_fifo_transfer* flexcan_fifo_transfer_t
FlexCAN Rx FIFO transfer.

typedef struct *_flexcan_handle* flexcan_handle_t
FlexCAN handle structure definition.

typedef void (*flexcan_transfer_callback_t)(CAN_Type *base, *flexcan_handle_t* *handle, *status_t* status, uint64_t result, void *userData)

FLEXCAN_WAIT_TIMEOUT

FLEXCAN_POLLING_TIMEOUT
Max loops to wait for polling transfer.

FLEXCAN_MODULE_TIMEOUT
Max loops to wait for FlexCAN register access complete.

DLC_LENGTH_DECODE(dlc)
FlexCAN frame length helper macro.

FLEXCAN_ID_STD(id)
FlexCAN Frame ID helper macro.

Standard Frame ID helper macro.

FLEXCAN_ID_EXT(id)
Extend Frame ID helper macro.

FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)
FlexCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)
Extend Rx Message Buffer Mask helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)
FlexCAN Legacy Rx FIFO Mask helper macro.

Standard Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)
Standard Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)
Standard Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id)
Standard Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id)
Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id)
Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)
Standard Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)
Extend Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)
Extend Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)
Extend Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)
Extend Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)
Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)
Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)
Extend Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)
FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)
Standard Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)
Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)
Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)
Standard Rx FIFO Filter helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)
Extend Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)
Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)
Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id)
Extend Rx FIFO Filter helper macro Type C lower part helper macro.

ENHANCED_RX_FIFO_FSCH(x)
FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.

RTR_STD_HIGH(x)

RTR_STD_LOW(x)

RTR_EXT(x)

ID_STD_LOW(id)

ID_STD_HIGH(id)

ID_EXT(id)

FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask)
Standard ID filter element with filter + mask scheme.

FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE(id_upper, rtr, id_lower, rtr_mask)
Standard ID filter element with filter range.

FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS(id1, rtr1, id2, rtr2)
Standard ID filter element with two filters without masks.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW(id, rtr)
Extended ID filter element with filter + mask scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH(id_mask, rtr_mask)
Extended ID filter element with filter + mask scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW(id_upper, rtr)
Extended ID filter element with range scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH(id_lower, rtr_mask)
Extended ID filter element with range scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW(id2, rtr2)
Extended ID filter element with two filters without masks low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH(id1, rtr1)
Extended ID filter element with two filters without masks high word.

FLEXCAN_PN_STD_MASK(id, rtr)
FlexCAN Pretended Networking ID Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_PN_EXT_MASK(id, rtr)
Extend Rx Message Buffer Mask helper macro.

FLEXCAN_PN_INT_MASK(x)
FlexCAN interrupt/status flag helper macro.

FLEXCAN_PN_INT_UNMASK(x)

FLEXCAN_PN_STATUS_MASK(x)

FLEXCAN_PN_STATUS_UNMASK(x)

FLEXCAN_EFIFO_INT_MASK(x)

FLEXCAN_EFIFO_INT_UNMASK(x)

FLEXCAN_EFIFO_STATUS_MASK(x)

FLEXCAN_EFIFO_STATUS_UNMASK(x)

FLEXCAN_MECR_INT_MASK(x)

FLEXCAN_MECR_INT_UNMASK(x)

FLEXCAN_MECR_STATUS_MASK(x)

FLEXCAN_MECR_STATUS_UNMASK(x)

FLEXCAN_ERROR_AND_STATUS_INT_FLAG

FLEXCAN_PNWAKE_UP_FLAG

FLEXCAN_WAKE_UP_FLAG

FLEXCAN_MEMORY_ERROR_INT_FLAG

FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG

FlexCAN Enhanced Rx FIFO base address helper macro.

E_RX_FIFO(base)

FLEXCAN_CALLBACK(x)

FlexCAN transfer callback function.

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_FLEXCAN_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

struct _flexcan_memory_error_report_status

#include <fsl_flexcan.h> FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN_GetMemoryErrorReportStatus() function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

Public Members

flexcan_memory_error_type_t errorType

The type of memory error that giving rise to the report.

flexcan_memory_access_type_t accessType

The type of memory access that giving rise to the memory error.

uint16_t accessAddress

The address where memory error detected.

uint32_t errorData

The raw data word read from memory with error.

struct _flexcan_frame

#include <fsl_flexcan.h> FlexCAN message frame structure.

struct _flexcan_fd_frame

#include <fsl_flexcan.h> CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see _flexcan_fd_frame_length.

Public Members

uint32_t idhit

Note: ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using FLEXCAN_FixEnhancedRxFifoFrameIdHit API is recommended to ensure this idhit value is correct. CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).

uint32_t hrtimestamp

Note: HR timestamp offset is changed dynamically according to data length code (DLC). External 32-bit on-chip timer high-resolution timestamp.

struct _flexcan_timing_config

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

uint32_t preDivider

Classic CAN or CAN FD nominal phase bit rate prescaler.

uint32_t rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint32_t phaseSeg1

Classic CAN or CAN FD nominal phase Segment 1.

uint32_t phaseSeg2

Classic CAN or CAN FD nominal phase Segment 2.

uint32_t propSeg

Classic CAN or CAN FD nominal phase Propagation Segment.

uint32_t fpreDivider

CAN FD data phase bit rate prescaler.

uint32_t frJumpwidth

CAN FD data phase Re-sync Jump Width.

uint32_t fphaseSeg1

CAN FD data phase Phase Segment 1.

uint32_t fphaseSeg2

CAN FD data phase Phase Segment 2.

uint32_t fpropSeg
: CAN FD data phase Propagation Segment.

struct _flexcan_config
: *#include <fsl_flexcan.h>* FlexCAN module configuration structure.

Deprecated:
: Do not use the baudRate. It has been superceded bitRate

  Do not use the baudRateFD. It has been superceded bitRateFD

Public Members

flexcan_clock_source_t clkSrc
: Clock source for FlexCAN Protocol Engine.

flexcan_wake_up_source_t wakeupSrc
: Wake up source selection.

uint8_t maxMbNum
: The maximum number of Message Buffers used by user.

bool enableLoopBack
: Enable or Disable Loop Back Self Test Mode.

bool enableTimerSync
: Enable or Disable Timer Synchronization.

bool enableIndividMask
: Enable or Disable Rx Individual Mask and Queue feature.

bool disableSelfReception
: Enable or Disable Self Reflection.

bool enableListenOnlyMode
: Enable or Disable Listen Only Mode.

bool enableDoze
: Enable or Disable Doze Mode.

bool enablePretendedeNetworking
: Enable or Disable the Pretended Networking mode.

bool enableMemoryErrorControl
: Enable or Disable the memory errors detection and correction mechanism.

bool enableNonCorrectableErrorEnterFreeze
: Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.

bool enableTransceiverDelayMeasure
: Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.

bool enableRemoteRequestFrameStored
: true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response Frame.

flexcan_endianness_t payloadEndianness

Selects the byte order for the payload of transmit and receive frames, see flexcan_endianness_t.

bool enableExternalTimeTick

true: External time tick clocks the free-running timer. false: FlexCAN bit clock clocks the free-running timer.

flexcan_MB_timestamp_base_t captureTimeBase

Timebase of message buffer 16-bit TIME_STAMP field.

flexcan_capture_point_t capturePoint

Point in time when 32-bit timebase is captured during CAN frame.

struct __flexcan__rx__mb__config

#include <fsl_flexcan.h> FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

Public Members

uint32_t id

CAN Message Buffer Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

flexcan_frame_format_t format

CAN Frame Identifier format(Standard of Extend).

flexcan_frame_type_t type

CAN Frame Type(Data or Remote for classical CAN only).

struct __flexcan__pn__config

#include <fsl_flexcan.h> FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

Public Members

bool enableTimeout

Enable or Disable timeout event trigger wakeup.

uint16_t timeoutValue

The timeout value that generates a wakeup event, the counter timer is incremented based on 64 times the CAN Bit Time unit.

bool enableMatch

Enable or Disable match event trigger wakeup.

flexcan_pn_match_source_t matchSrc

Selects the match source (ID and/or data match) to trigger wakeup.

uint8_t matchNum

The number of times a given message must match the predefined ID and/or data before generating a wakeup event, range in 0x1 ~ 0xFF.

flexcan_pn_match_mode_t idMatchMode
The ID match type.

flexcan_pn_match_mode_t dataMatchMode
The data match type.

uint32_t idLower
The ID target values 1 which used either for ID match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in ID match “range detection”.

uint32_t idUpper
The ID target values 2 which used only as the upper limit value in ID match “range detection” or used to store the ID mask in “equal to”.

uint8_t lengthLower
The lower limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

uint8_t lengthUpper
The upper limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

struct __flexcan__rx__fifo__config
#include <fsl_flexcan.h> FlexCAN Legacy Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable
Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

uint8_t idFilterNum
The FlexCAN Legacy Rx FIFO Filter elements quantity.

flexcan_rx_fifo_filter_type_t idFilterType
The FlexCAN Legacy Rx FIFO Filter type.

flexcan_rx_fifo_priority_t priority
The FlexCAN Legacy Rx FIFO receive priority.

struct __flexcan__enhanced__rx__fifo__std__id__filter
#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

Public Members

uint32_t filterType
FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1
CAN FD frame data length code (DLC), range see _flexcan_fd_frame_length, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use DLC_LENGTH_DECODE(length) macro to get the number of valid data bytes.

uint32_t std1
CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct __flexcan_enhanced_rx_fifo_ext_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see _flexcan_fd_frame_length, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use DLC_LENGTH_DECODE(length) macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct __flexcan_enhanced_rx_fifo_config

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to idFilterNum. There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupie 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

uint8_t idFilterPairNum

idFilterPairNum is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

uint8_t extendIdFilterNum

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, extendIdFilterNum need less than or equal to idFilterPairNum.

uint8_t fifoWatermark

(fifoWatermark + 1) is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

flexcan_efifo_dma_per_read_length_t dmaPerReadLength

Define the length of each read of the Enhanced RX FIFO element by the DAM, see _flexcan_fd_frame_length.

flexcan_rx_fifo_priority_t priority
The FlexCAN Enhanced Rx FIFO receive priority.

struct _flexcan_mb_transfer
#include <fsl_flexcan.h> FlexCAN Message Buffer transfer.

Public Members

flexcan_frame_t *frame
The buffer of CAN Message to be transfer.

uint8_t mbIdx
The index of Message buffer used to transfer Message.

struct _flexcan_fifo_transfer
#include <fsl_flexcan.h> FlexCAN Rx FIFO transfer.

Public Members

flexcan_fd_frame_t *framefd
The buffer of CAN Message to be received from Enhanced Rx FIFO.

flexcan_frame_t *frame
The buffer of CAN Message to be received from Legacy Rx FIFO.

size_t frameNum
Number of CAN Message need to be received from Legacy or Ehanced Rx FIFO.

struct _flexcan_handle
#include <fsl_flexcan.h> FlexCAN handle structure.

Public Members

flexcan_transfer_callback_t callback
Callback function.

void *userData
FlexCAN callback function parameter.

flexcan_frame_t *volatile mbFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN data from Message Buffers.

flexcan_fd_frame_t *volatile mbFDFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN FD data from Message Buffers.

flexcan_frame_t *volatile rxFifoFrameBuf
The buffer for received CAN data from Legacy Rx FIFO.

flexcan_fd_frame_t *volatile rxFifoFDFrameBuf
The buffer for received CAN FD data from Ehanced Rx FIFO.

size_t rxFifoFrameNum
The number of CAN messages remaining to be received from Legacy or Ehanced Rx FIFO.

size_t rxFifoTransferTotalNum
Total CAN Message number need to be received from Legacy or Ehanced Rx FIFO.

volatile uint8_t mbState[CAN_WORD1_COUNT]
Message Buffer transfer state.

volatile uint8_t rxFifoState
Rx FIFO transfer state.

volatile uint32_t timestamp[CAN_WORD1_COUNT]
Mailbox transfer timestamp.

struct byteStatus

Public Members

bool byteIsRead
The byte n (0~3) was read or not. The type of error and which bit in byte (n) is affected by the error.

struct ___unnamed21___

Public Members

uint32_t timestamp
FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length
CAN frame data length in bytes (Range: 0~8).

uint32_t type
CAN Frame Type(DATA or REMOTE).

uint32_t format
CAN Frame Identifier(STD or EXT format).

uint32_t ___pad0___
Reserved.

uint32_t idhit
CAN Rx FIFO filter hit id(This value is only used in Rx FIFO receive mode).

struct ___unnamed23___

Public Members

uint32_t id
CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

uint32_t ___pad0___
Reserved.

union ___unnamed25___

Public Members

struct __flexcan_frame

struct __flexcan_frame

struct ___unnamed27___

Public Members

uint32_t dataWord0
: CAN Frame payload word0.

uint32_t dataWord1
: CAN Frame payload word1.

struct ___unnamed29___

Public Members

uint8_t dataByte3
: CAN Frame payload byte3.

uint8_t dataByte2
: CAN Frame payload byte2.

uint8_t dataByte1
: CAN Frame payload byte1.

uint8_t dataByte0
: CAN Frame payload byte0.

uint8_t dataByte7
: CAN Frame payload byte7.

uint8_t dataByte6
: CAN Frame payload byte6.

uint8_t dataByte5
: CAN Frame payload byte5.

uint8_t dataByte4
: CAN Frame payload byte4.

struct ___unnamed31___

Public Members

uint32_t timestamp
: FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length
: CAN FD frame data length code (DLC), range see _flexcan_fd_frame_length, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use DLC_LENGTH_DECODE(length) macro to get the number of valid data bytes.

uint32_t type
: CAN Frame Type(DATA only).

uint32_t format
: CAN Frame Identifier(STD or EXT format).

uint32_t srr
: Substitute Remote request.

uint32_t esi
: Error State Indicator.

uint32_t brs
: Bit Rate Switch.

uint32_t edl
: Extended Data Length.

struct ___unnamed33___

Public Members

uint32_t id
: CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

uint32_t ___pad0___
: Reserved.

union ___unnamed35___

Public Members

struct _flexcan_fd_frame

struct _flexcan_fd_frame

struct ___unnamed37___

Public Members

uint32_t dataWord[16]
: CAN FD Frame payload, 16 double word maximum.

struct ___unnamed39___

Public Members

uint8_t dataByte3
: CAN Frame payload byte3.

uint8_t dataByte2
: CAN Frame payload byte2.

uint8_t dataByte1
: CAN Frame payload byte1.

uint8_t dataByte0
: CAN Frame payload byte0.

uint8_t dataByte7
: CAN Frame payload byte7.

uint8_t dataByte6
: CAN Frame payload byte6.

uint8_t dataByte5
: CAN Frame payload byte5.

uint8_t dataByte4
CAN Frame payload byte4.

union ___unnamed41___

Public Members

struct __flexcan__config

struct __flexcan__config

struct ___unnamed43___

Public Members

uint32_t baudRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t baudRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

struct ___unnamed45___

Public Members

uint32_t bitRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t bitRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

union ___unnamed47___

Public Members

struct __flexcan__pn__config

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range

detection”.

struct __flexcan__pn__config

struct ___unnamed51___

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range

detection”.

Public Members

uint32_t lowerWord0
CAN Frame payload word0.

uint32_t lowerWord1
CAN Frame payload word1.

struct ___unnamed53___

Public Members

uint8_t lowerByte3
: CAN Frame payload byte3.

uint8_t lowerByte2
: CAN Frame payload byte2.

uint8_t lowerByte1
: CAN Frame payload byte1.

uint8_t lowerByte0
: CAN Frame payload byte0.

uint8_t lowerByte7
: CAN Frame payload byte7.

uint8_t lowerByte6
: CAN Frame payload byte6.

uint8_t lowerByte5
: CAN Frame payload byte5.

uint8_t lowerByte4
: CAN Frame payload byte4.

union ___unnamed49___

Public Members

struct __flexcan__pn__config

< The data target values 2 which used only as the upper limit value in data match “range

detection” or used to store the data mask in “equal to”.

struct __flexcan__pn__config

struct ___unnamed55___

< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

Public Members

uint32_t upperWord0
: CAN Frame payload word0.

uint32_t upperWord1
: CAN Frame payload word1.

struct ___unnamed57___

Public Members

uint8_t upperByte3
: CAN Frame payload byte3.

uint8_t upperByte2
: CAN Frame payload byte2.

uint8_t upperByte1
: CAN Frame payload byte1.

uint8_t upperByte0
: CAN Frame payload byte0.

uint8_t upperByte7
: CAN Frame payload byte7.

uint8_t upperByte6
: CAN Frame payload byte6.

uint8_t upperByte5
: CAN Frame payload byte5.

uint8_t upperByte4
: CAN Frame payload byte4.

2.17 FlexCAN eDMA Driver

void FLEXCAN_TransferCreateHandleEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *rxFifoEdmaHandle)

Initializes the FlexCAN handle, which is used in transactional functions.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

void FLEXCAN_PrepareTransfConfiguration(CAN_Type *base, *flexcan_fifo_transfer_t* *pFifoXfer, *edma_transfer_config_t* *pEdmaConfig)

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

status_t FLEXCAN_StartTransferDatafromRxFIFO(CAN_Type *base, *flexcan_edma_handle_t* *handle, *edma_transfer_config_t* *pEdmaConfig)

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer confiugration and Start eDMA transfer

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferGetReceiveFifoCountEMDA(CAN_Type *base, *flexcan_edma_handle_t* *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXCAN_TransferAbortReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle)

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.

status_t FLEXCAN_TransferReceiveEnhancedFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives the CAN FD Message from the Enhanced Rx FIFO using eDMA.

This function receives the CAN FD Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA(CAN_Type *base, *flexcan_edma_handle_t* *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

FSL_FLEXCAN_EDMA_DRIVER_VERSION

FlexCAN EDMA driver version.

typedef struct *_flexcan_edma_handle* flexcan_edma_handle_t

typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, *flexcan_edma_handle_t* *handle, *status_t* status, void *userData)

FlexCAN transfer callback function.

struct _flexcan_edma_handle

#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.

Public Members

flexcan_edma_transfer_callback_t callback

Callback function.

void *userData

FlexCAN callback function parameter.

edma_handle_t *rxFifoEdmaHandle

The EDMA handler for Rx FIFO.

volatile uint8_t rxFifoState
: Rx FIFO transfer state.

size_t frameNum
: The number of messages that need to be received.

flexcan_fd_frame_t *framefd
: Point to the buffer of CAN Message to be received from Enhanced Rx FIFO.

2.18 FlexIO: FlexIO Driver

2.19 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)
: Gets the default configuration to configure the FlexIO module. The configuration can 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

2.20 FlexIO eDMA I2S Driver

void FLEXIO_I2S_TransferTxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.

- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferRxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferSetFormatEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

status_t FLEXIO_I2S_TransferSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S transfer using DMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA send successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_TxBusy – FlexIO I2S is busy sending data.

status_t FLEXIO_I2S_TransferReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S receive using eDMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA receive successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_RxBusy – FlexIO I2S is busy receiving data.

void FLEXIO_I2S_TransferAbortSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S transfer using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

void FLEXIO_I2S_TransferAbortReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S receive using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

status_t FLEXIO_I2S_TransferGetSendCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Gets the remaining bytes to be sent.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

Return values

- kStatus_Success – Succeed get the transfer count.

- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t FLEXIO_I2S_TransferGetReceiveCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Get the remaining bytes to be received.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION

FlexIO I2S EDMA driver version 2.1.9.

typedef struct *_flexio_i2s_edma_handle* flexio_i2s_edma_handle_t

typedef void (*flexio_i2s_edma_callback_t)(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2S eDMA transfer callback function for finish and error.

struct _flexio_i2s_edma_handle

#include <fsl_flexio_i2s_edma.h> FlexIO I2S DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for FlexIO I2S send

uint8_t bytesPerFrame

Bytes in a frame

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint32_t state

Internal state for FlexIO I2S eDMA transfer

flexio_i2s_edma_callback_t callback

Callback for users while transfer finish or error occurred

void *userData

User callback parameter

edma_tcd_t tcd[(4U) + 1U]

TCD pool for eDMA transfer.

flexio_i2s_transfer_t queue[(4U)]

Transfer queue storing queued transfer.

size_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.21 FlexIO eDMA SPI Driver

status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txHandle, *edma_handle_t* *rxHandle)

Initializes the FlexIO SPI master eDMA handle.

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_SPI_MasterTransferEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

status_t FLEXIO_SPI_MasterTransferGetCountEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *flexio_spi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txHandle, *edma_handle_t* *rxHandle)

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

status_t FLEXIO_SPI_SlaveTransferEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCountEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION

FlexIO SPI EDMA driver version.

typedef struct *_flexio_spi_master_edma_handle* flexio_spi_master_edma_handle_t

typedef for flexio_spi_master_edma_handle_t in advance.

typedef *flexio_spi_master_edma_handle_t* flexio_spi_slave_edma_handle_t

Slave handle is the same with master handle.

typedef void (*flexio_spi_master_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI slave callback for finished transmit.

struct _flexio_spi_master_edma_handle

#include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch the content of the handle.

Public Members

size_t transferSize
: Total bytes to be transferred.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

bool txInProgress
: Send transfer in progress

bool rxInProgress
: Receive transfer in progress

edma_handle_t *txHandle
: DMA handler for SPI send

edma_handle_t *rxHandle
: DMA handler for SPI receive

flexio_spi_master_edma_transfer_callback_t callback
: Callback for SPI DMA transfer

void *userData
: User Data for SPI DMA callback

2.22 FlexIO eDMA UART Driver

status_t FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_UART_Type.
- handle – Pointer to flexio_uart_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

status_t FLEXIO_UART_TransferReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

void FLEXIO_UART_TransferAbortSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

void FLEXIO_UART_TransferAbortReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

status_t FLEXIO_UART_TransferGetSendCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferGetReceiveCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

FSL_FLEXIO_UART_EDMA_DRIVER_VERSION

FlexIO UART EDMA driver version.

typedef struct *_flexio_uart_edma_handle* flexio_uart_edma_handle_t

typedef void (*flexio_uart_edma_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct _flexio_uart_edma_handle

#include <fsl_flexio_uart_edma.h> UART eDMA handle.

Public Members

flexio_uart_edma_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

size_t txDataSizeAll

Total bytes to be sent.

size_t rxDataSizeAll

Total bytes to be received.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.23 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)
Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)
De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND _flexio_i2c_master_status_flags can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_I2C_RxFullFlag
  - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t* direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
  - kFLEXIO_I2C_Write: Transmit
  - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

status_t FLEXIO_I2C_MasterTransferGetCount(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_I2C_MasterTransferAbort(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy
I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle
I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak
NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout
Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable
Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable
Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag
Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag
Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag
Receive NAK flag.

enum _flexio_i2c_direction

Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write
Master send to slave.

enumerator kFLEXIO_I2C_Read
Master receive from slave.

typedef enum *_flexio_i2c_direction* flexio_i2c_direction_t

Direction of master transfer.

typedef struct *_flexio_i2c_type* FLEXIO_I2C_Type
Define FlexIO I2C master access structure typedef.

typedef struct *_flexio_i2c_master_config* flexio_i2c_master_config_t
Define FlexIO I2C master user configuration structure.

typedef struct *_flexio_i2c_master_transfer* flexio_i2c_master_transfer_t
Define FlexIO I2C master transfer structure.

typedef struct *_flexio_i2c_master_handle* flexio_i2c_master_handle_t
FlexIO I2C master handle typedef.

typedef void (*flexio_i2c_master_transfer_callback_t)(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *status_t* status, void *userData)
FlexIO I2C master transfer callback typedef.

I2C_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_i2c_type
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t SDAPinIndex
Pin select for I2C SDA.

uint8_t SCLPinIndex
Pin select for I2C SCL.

uint8_t shifterIndex[2]
Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]
Timer index used in FlexIO I2C.

uint32_t baudrate
Master transfer baudrate, used to calculate delay time.

struct _flexio_i2c_master_config
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster
Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze
Enable/disable FlexIO operation in doze mode.

bool enableInDebug
Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
Baud rate in Bps.

struct __flexio_i2c_master_transfer
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags
Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress
7-bit slave address.

flexio_i2c_direction_t direction
Transfer direction, read or write.

uint32_t subaddress
Sub address. Transferred MSB first.

uint8_t subaddressSize
Size of sub address.

uint8_t volatile *data
Transfer buffer.

volatile size_t dataSize
Transfer size.

struct __flexio_i2c_master_handle
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer
FlexIO I2C master transfer copy.

size_t transferSize
Total bytes to be transferred.

uint8_t state
Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback
Callback function called at transfer event. Callback function called at transfer event.

void *userData
Callback parameter passed to callback function.

bool needRestart
Whether master needs to send re-start signal.

2.24 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO_I2S_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in FLEXIO_I2S_Init(). Users may use the initialized structure unchanged in FLEXIO_I2S_Init() or modify some fields of the structure before calling FLEXIO_I2S_Init().

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the 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

2.25 FlexIO SPI Driver

void FLEXIO_SPI_MasterInit(*FLEXIO_SPI_Type* *base, *flexio_spi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 3*2=6. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – status flag The parameter can be any combination of the following values:
  - kFLEXIO_SPI_TxEmptyFlag
  - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_DisableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_EnableDMA(*FLEXIO_SPI_Type* *base, uint32_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

static inline void FLEXIO_SPI_Enable(*FLEXIO_SPI_Type* *base, bool enable)

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.
- enable – True to enable, false does not have any effect.

void FLEXIO_SPI_MasterSetBaudRate(*FLEXIO_SPI_Type* *base, uint32_t baudRate_Bps, uint32_t srcClockHz)

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

static inline void FLEXIO_SPI_WriteData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint32_t data)

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

static inline uint32_t FLEXIO_SPI_ReadData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

status_t FLEXIO_SPI_WriteBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, const uint8_t *buffer, size_t size)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint8_t *buffer, size_t size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.

- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle)

Aborts the master data transfer, which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.

- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_SlaveTransferCreateHandle(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle,
flexio_spi_slave_transfer_callback_t callback,
void *userData)

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_SlaveTransferNonBlocking(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle,
flexio_spi_transfer_t *xfer)

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbort(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle)

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCount(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle,
size_t *count)

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI slave IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

FSL_FLEXIO_SPI_DRIVER_VERSION

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator kStatus_FLEXIO_SPI_Busy

FlexIO SPI is busy.

enumerator kStatus_FLEXIO_SPI_Idle

SPI is idle

enumerator kStatus_FLEXIO_SPI_Error

FlexIO SPI error.

enumerator kStatus_FLEXIO_SPI_Timeout

FlexIO SPI timeout polling status flags.

enum _flexio_spi_clock_phase

FlexIO SPI clock phase configuration.

Values:

enumerator kFLEXIO_SPI_ClockPhaseFirstEdge

First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge

First edge on SPSCK occurs at the start of the first cycle of a data transfer.

enum _flexio_spi_shift_direction

FlexIO SPI data shifter direction options.

Values:

enumerator kFLEXIO_SPI_MsbFirst

Data transfers start with most significant bit.

enumerator kFLEXIO_SPI_LsbFirst

Data transfers start with least significant bit.

enum _flexio_spi_data_bitcount_mode

FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode
8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags
Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag
Receive buffer full flag.

enum _flexio_spi_dma_enable
Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable
Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable
Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable
All DMA request source

enum _flexio_spi_transfer_flags
Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb
FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb
FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb
FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb
FlexIO SPI 16-bit LSB first

enumerator kFLEXIO_SPI_32bitMsb
FlexIO SPI 32-bit MSB first

enumerator kFLEXIO_SPI_32bitLsb
FlexIO SPI 32-bit LSB first

enumerator kFLEXIO_SPI_csContinuous
Enable the CS signal continuous mode

typedef enum *_flexio_spi_clock_phase* flexio_spi_clock_phase_t
FlexIO SPI clock phase configuration.

typedef enum *_flexio_spi_shift_direction* flexio_spi_shift_direction_t
FlexIO SPI data shifter direction options.

typedef enum *_flexio_spi_data_bitcount_mode* flexio_spi_data_bitcount_mode_t
FlexIO SPI data length mode options.

typedef struct *_flexio_spi_type* FLEXIO_SPI_Type
Define FlexIO SPI access structure typedef.

typedef struct *_flexio_spi_master_config* flexio_spi_master_config_t
Define FlexIO SPI master configuration structure.

typedef struct *_flexio_spi_slave_config* flexio_spi_slave_config_t
Define FlexIO SPI slave configuration structure.

typedef struct *_flexio_spi_transfer* flexio_spi_transfer_t
Define FlexIO SPI transfer structure.

typedef struct *_flexio_spi_master_handle* flexio_spi_master_handle_t
typedef for flexio_spi_master_handle_t in advance.

typedef *flexio_spi_master_handle_t* flexio_spi_slave_handle_t
Slave handle is the same with master handle.

typedef void (*flexio_spi_master_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *status_t* status, void *userData)
FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *status_t* status, void *userData)
FlexIO SPI slave callback for finished transmit.

FLEXIO_SPI_DUMMYDATA
FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

SPI_RETRY_TIMES
Retry times for waiting flag.

FLEXIO_SPI_XFER_DATA_FORMAT(flag)
Get the transfer data format of width and bit order.

struct _flexio_spi_type
#include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

Public Members

FLEXIO_Type *flexioBase
: FlexIO base pointer.

uint8_t SDOPinIndex
: Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex
: Pin select for data input.

uint8_t SCKPinIndex
: Pin select for clock.

uint8_t CSnPinIndex
: Pin select for enable.

uint8_t shifterIndex[2]
: Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]
: Timer index used in FlexIO SPI.

struct __flexio_spi_master_config
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster
: Enable/disable FlexIO SPI master after configuration.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode.

bool enableInDebug
: Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

flexio_spi_clock_phase_t phase
: Clock phase.

flexio_spi_data_bitcount_mode_t dataMode
: 8bit or 16bit mode.

struct __flexio_spi_slave_config
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave
: Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode.

bool enableInDebug
: Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase
: Clock phase.

flexio_spi_data_bitcount_mode_t dataMode
: 8bit or 16bit mode.

struct __flexio_spi_transfer
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData
: Send buffer.

uint8_t *rxData
: Receive buffer.

size_t dataSize
: Transfer bytes.

uint8_t flags
: FlexIO SPI control flag, MSB first or LSB first.

struct __flexio_spi_master_handle
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData
: Transfer buffer.

uint8_t *rxData
: Receive buffer.

size_t transferSize
: Total bytes to be transferred.

volatile size_t txRemainingBytes
: Send data remaining in bytes.

volatile size_t rxRemainingBytes
: Receive data remaining in bytes.

volatile uint32_t state
: FlexIO SPI internal state.

uint8_t bytePerFrame
: SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction
Shift direction.

flexio_spi_master_transfer_callback_t callback
FlexIO SPI callback.

void *userData
Callback parameter.

bool isCsContinuous
Is current transfer using CS continuous mode.

uint32_t timer1Cfg
TIMER1 TIMCFG regiser value backup.

2.26 FlexIO UART Driver

status_t FLEXIO_UART_Init(*FLEXIO_UART_Type* *base, const *flexio_uart_config_t* *userConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_GetDefaultConfig().

Example

```
FLEXIO_UART_Type base = {
.flexioBase = FLEXIO,
.TxPinIndex = 0,
.RxPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_uart_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the 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 ___unnamed249___

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

2.27 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32_t FLEXSPI_GetInstance(FLEXSPI_Type *base)
Get the instance number for FLEXSPI.

Parameters

- base – FLEXSPI base pointer.

status_t FLEXSPI_CheckAndClearError(FLEXSPI_Type *base, uint32_t status)
Check and clear IP command execution errors.

Parameters

- base – FLEXSPI base pointer.
- status – interrupt status.

void FLEXSPI_Init(FLEXSPI_Type *base, const *flexspi_config_t* *config)
Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI_GetDefaultConfig(*flexspi_config_t* *config)

Gets default settings for FLEXSPI.

Parameters

- config – FLEXSPI configuration structure.

void FLEXSPI_Deinit(FLEXSPI_Type *base)

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

- base – FLEXSPI peripheral base address.

void FLEXSPI_UpdateDllValue(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SetFlashConfig(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SoftwareReset(FLEXSPI_Type *base)

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – FLEXSPI peripheral base address.

static inline void FLEXSPI_Enable(FLEXSPI_Type *base, bool enable)

Enables or disables the FLEXSPI module.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

void FLEXSPI_UpdateAhbBuffersSettings(FLEXSPI_Type *base, *flexspi_ahbBuffers_ctrl_t* *ptrAhbBufferCtrl)

Update all AHB buffers' settings, including buffer size, master ID.

Parameters

- base – FLEXSPI peripheral base address.

- ptrAhbBufferCtrl – Pointer to structure flexspi_ahbBuffers_ctrl_t which store all AHB buffers' settings.

static inline void FLEXSPI_EnableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Enables the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Disable the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP tx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – tx fifo address.

static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP rx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – rx fifo address.

static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)

Clears the FLEXSPI IP FIFO logic.

Parameters

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)

Gets the valid data entries in the FLEXSPI FIFOs.

Parameters

- base – FLEXSPI peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.

Return values

interrupt – status flag, use status flag to AND flexspi_flags_t could get the related status.

static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline *flexspi_arb_command_source_t* FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type *base)

Gets the trigger source of current command sequence granted by arbitrator.

Parameters

- base – FLEXSPI peripheral base address.

Return values

trigger – source of current command sequence.

static inline *flexspi_ip_error_code_t* FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when IP command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when IP command error detected.

static inline *flexspi_ahb_error_code_t* FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when AHB command error detected.

Parameters

- base – FLEXSPI peripheral base address.

- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when AHB command error detected.

static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)

Returns whether the bus is idle.

Parameters

- base – FLEXSPI peripheral base address.

Return values

- true – Bus is idle.
- false – Bus is busy.

void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, *flexspi_read_sample_clock_t* clockSource)

Update read sample clock source.

Parameters

- base – FLEXSPI peripheral base address.
- clockSource – clockSource of type flexspi_read_sample_clock_t

void FLEXSPI_UpdateLUT(FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t count)

Updates the LUT table.

Parameters

- base – FLEXSPI peripheral base address.
- index – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory.
- cmd – Command sequence array.
- count – Number of sequences.

static inline void FLEXSPI_WriteData(FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)

Writes data into FIFO.

Parameters

- base – FLEXSPI peripheral base address
- data – The data bytes to send
- fifoIndex – Destination fifo index.

static inline uint32_t FLEXSPI_ReadData(FLEXSPI_Type *base, uint8_t fifoIndex)

Receives data from data FIFO.

Parameters

- base – FLEXSPI peripheral base address
- fifoIndex – Source fifo index.

Returns

The data in the FIFO.

status_t FLEXSPI_WriteBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to send

Return values

- kStatus_Success – write success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_ReadBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to receive

Return values

- kStatus_Success – read success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequencen error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_TransferBlocking(FLEXSPI_Type *base, *flexspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Parameters

- base – FLEXSPI peripheral base address
- xfer – pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error

- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_callback_t* callback, void *userData)

Initializes the FLEXSPI handle which is used in transactional functions.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_t* *xfer)

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXSPI_Busy, the transfer is finished. For FLEXSPI_Read, the dataSize should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state.
- xfer – pointer to flexspi_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXSPI_Busy – Previous transmission still not finished.

status_t FLEXSPI_TransferGetCount(FLEXSPI_Type *base, *flexspi_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXSPI_TransferAbort(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state

void FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Master interrupt handler.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure.

FSL_FLEXSPI_DRIVER_VERSION

FLEXSPI driver version.

Status structure of FLEXSPI.

Values:

enumerator kStatus_FLEXSPI_Busy

FLEXSPI is busy

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandSequenceError

IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandGrantTimeout

IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, _flexspi_command.

Values:

enumerator kFLEXSPI_Command_STOP

Stop execution, deassert CS.

enumerator kFLEXSPI_Command_SDR

Transmit Command code to Flash, using SDR mode.

enumerator kFLEXSPI_Command_RADDR_SDR

Transmit Row Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_CADDR_SDR

Transmit Column Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE1_SDR

Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE2_SDR
Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE4_SDR
Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE8_SDR
Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_WRITE_SDR
Transmit Programming Data to Flash, using SDR mode.

enumerator kFLEXSPI_Command_READ_SDR
Receive Read Data from Flash, using SDR mode.

enumerator kFLEXSPI_Command_LEARN_SDR
Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator kFLEXSPI_Command_DATSZ_SDR
Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator kFLEXSPI_Command_DUMMY_SDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_SDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_DDR
Transmit Command code to Flash, using DDR mode.

enumerator kFLEXSPI_Command_RADDR_DDR
Transmit Row Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_CADDR_DDR
Transmit Column Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE1_DDR
Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE2_DDR
Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE4_DDR
Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE8_DDR
Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_WRITE_DDR
Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI_Command_READ_DDR
Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI_Command_LEARN_DDR
Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI_Command_DATSZ_DDR
Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI_Command_DUMMY_DDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_DDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_JUMP_ON_CS
Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum _flexspi_pad
pad definition of FLEXSPI, use to form LUT instruction.

Values:

enumerator kFLEXSPI_1PAD
Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI_2PAD
Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI_4PAD
Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI_8PAD
Transmit command/address and transmit/receive data only through DATA[7:0].

enum _flexspi_flags
FLEXSPI interrupt status flags.

Values:

enumerator kFLEXSPI_SequenceExecutionTimeoutFlag
Sequence execution timeout.

enumerator kFLEXSPI_AhbBusErrorFlag
AHB Bus error flag.

enumerator kFLEXSPI_SckStoppedBecauseTxEmptyFlag
SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI_SckStoppedBecauseRxFullFlag
SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI_IpTxFifoWatermarkEmptyFlag
IP TX FIFO WaterMark empty.

enumerator kFLEXSPI_IpRxFifoWatermarkAvailableFlag
IP RX FIFO WaterMark available.

enumerator kFLEXSPI_AhbCommandSequenceErrorFlag
AHB triggered Command Sequences Error.

enumerator kFLEXSPI_IpCommandSequenceErrorFlag
IP triggered Command Sequences Error.

enumerator kFLEXSPI_AhbCommandGrantTimeoutFlag
AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandGrantTimeoutFlag
IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandExecutionDoneFlag
IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI_AllInterruptFlags
All flags.

enum _flexspi_read_sample_clock

FLEXSPI sample clock source selection for Flash Reading.

Values:

enumerator kFLEXSPI_ReadSampleClkLoopbackInternally

Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromDqsPad

Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromSckPad

SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI_ReadSampleClkExternalInputFromDqsPad

Flash provided Read strobe and input from DQS pad.

enum _flexspi_cs_interval_cycle_unit

FLEXSPI interval unit for flash device select.

Values:

enumerator kFLEXSPI_CsIntervalUnit1SckCycle

Chip selection interval: CSINTERVAL * 1 serial clock cycle.

enumerator kFLEXSPI_CsIntervalUnit256SckCycle

Chip selection interval: CSINTERVAL * 256 serial clock cycle.

enum _flexspi_ahb_write_wait_unit

FLEXSPI AHB wait interval unit for writing.

Values:

enumerator kFLEXSPI_AhbWriteWaitUnit2AhbCycle

AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8AhbCycle

AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32AhbCycle

AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit128AhbCycle

AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit512AhbCycle

AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit2048AhbCycle

AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8192AhbCycle

AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32768AhbCycle

AWRWAIT unit is 32768 ahb clock cycle.

enum _flexspi_ip_error_code

Error Code when IP command Error detected.

Values:

enumerator kFLEXSPI_IpCmdErrorNoError

No error.

enumerator kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd
IP command with JMP_ON_CS instruction used.

enumerator kFLEXSPI_IpCmdErrorUnknownOpCode
Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence
Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence
Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_IpCmdErrorInvalidAddress
Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI_IpCmdErrorSequenceExecutionTimeout
Sequence execution timeout.

enumerator kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss
Flash boundary crossed.

enum _flexspi_ahb_error_code
Error Code when AHB command Error detected.

Values:

enumerator kFLEXSPI_AhbCmdErrorNoError
No error.

enumerator kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd
AHB Write command with JMP_ON_CS instruction used in the sequence.

enumerator kFLEXSPI_AhbCmdErrorUnknownOpCode
Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence
Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence
Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_AhbCmdSequenceExecutionTimeout
Sequence execution timeout.

enum _flexspi_port
FLEXSPI operation port select.

Values:

enumerator kFLEXSPI_PortA1
Access flash on A1 port.

enumerator kFLEXSPI_PortA2
Access flash on A2 port.

enumerator kFLEXSPI_PortB1
Access flash on B1 port.

enumerator kFLEXSPI_PortB2
Access flash on B2 port.

enumerator kFLEXSPI_PortCount

enum _flexspi_arb_command_source
Trigger source of current command sequence granted by arbitrator.

Values:

enumerator kFLEXSPI_AhbReadCommand

enumerator kFLEXSPI_AhbWriteCommand

enumerator kFLEXSPI_IpCommand

enumerator kFLEXSPI_SuspendedCommand

enum _flexspi_command_type
Command type.

Values:

enumerator kFLEXSPI_Command
FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI_Config
FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI_Read

enumerator kFLEXSPI_Write

typedef enum *_flexspi_pad* flexspi_pad_t
pad definition of FLEXSPI, use to form LUT instruction.

typedef enum *_flexspi_flags* flexspi_flags_t
FLEXSPI interrupt status flags.

typedef enum *_flexspi_read_sample_clock* flexspi_read_sample_clock_t
FLEXSPI sample clock source selection for Flash Reading.

typedef enum *_flexspi_cs_interval_cycle_unit* flexspi_cs_interval_cycle_unit_t
FLEXSPI interval unit for flash device select.

typedef enum *_flexspi_ahb_write_wait_unit* flexspi_ahb_write_wait_unit_t
FLEXSPI AHB wait interval unit for writing.

typedef enum *_flexspi_ip_error_code* flexspi_ip_error_code_t
Error Code when IP command Error detected.

typedef enum *_flexspi_ahb_error_code* flexspi_ahb_error_code_t
Error Code when AHB command Error detected.

typedef enum *_flexspi_port* flexspi_port_t
FLEXSPI operation port select.

typedef enum *_flexspi_arb_command_source* flexspi_arb_command_source_t
Trigger source of current command sequence granted by arbitrator.

typedef enum *_flexspi_command_type* flexspi_command_type_t
Command type.

typedef struct *_flexspi_ahbBuffer_config* flexspi_ahbBuffer_config_t

typedef struct *_flexspi_ahbBuffers_ctrl* flexspi_ahbBuffers_ctrl_t
Structure to control all AHB buffers.

typedef struct *_flexspi_config* flexspi_config_t
FLEXSPI configuration structure.

typedef struct *_flexspi_device_config* flexspi_device_config_t
External device configuration items.

typedef struct *_flexspi_transfer* flexspi_transfer_t
Transfer structure for FLEXSPI.

typedef struct *_flexspi_handle* flexspi_handle_t

typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *status_t* status, void *userData)
FLEXSPI transfer callback function.

typedef struct *_flexspi_addr_map_config* flexspi_addr_map_config_t
Address mapping configuration structure.

FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT

FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
Formula to form FLEXSPI instructions in LUT table.

struct _flexspi_ahbBuffer_config
#include <fsl_flexspi.h>

Public Members

uint8_t priority
This priority for AHB Master Read which this AHB RX Buffer is assigned.

uint8_t masterIndex
AHB Master ID the AHB RX Buffer is assigned.

uint16_t bufferSize
AHB buffer size in byte.

bool enablePrefetch
AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows prefetch disable/enable separately for each master.

struct _flexspi_ahbBuffers_ctrl
#include <fsl_flexspi.h> Structure to control all AHB buffers.

Public Members

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]
Configurations of all AHB buffers.

struct _flexspi_config
#include <fsl_flexspi.h> FLEXSPI configuration structure.

Public Members

flexspi_read_sample_clock_t rxSampleClock
Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning
Enable/disable SCK output free-running.

bool enableDoze
: Enable/disable doze mode support.

bool enableHalfSpeedAccess
: Enable/disable divide by 2 of the clock for half speed commands.

bool enableSameConfigForAll
: Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16_t seqTimeoutCycle
: Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCyle*1024 serial root clock cycles.

uint8_t ipGrantTimeoutCycle
: Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle*1024 AHB clock cycles.

uint8_t txWatermark
: FLEXSPI IP transmit watermark value.

uint8_t rxWatermark
: FLEXSPI receive watermark value.

struct __flexspi_device_config
: *#include <fsl_flexspi.h>* External device configuration items.

Public Members

uint32_t flexspiRootClk
: FLEXSPI serial root clock.

bool isSck2Enabled
: FLEXSPI use SCK2.

uint32_t flashSize
: Flash size in KByte.

bool addressShift
: Address shift.

flexspi_cs_interval_cycle_unit_t CSIntervalUnit
: CS interval unit, 1 or 256 cycle.

uint16_t CSInterval
: CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

uint8_t CSHoldTime
: CS line hold time.

uint8_t CSSetupTime
: CS line setup time.

uint8_t dataValidTime
: Data valid time for external device.

uint8_t columnspace
: Column space size.

bool enableWordAddress
: If enable word address.

uint8_t AWRSeqIndex
Sequence ID for AHB write command.

uint8_t AWRSeqNumber
Sequence number for AHB write command.

uint8_t ARDSeqIndex
Sequence ID for AHB read command.

uint8_t ARDSeqNumber
Sequence number for AHB read command.

flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit
AHB write wait unit.

uint16_t AHBWriteWaitInterval
AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask
Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

struct _flexspi_transfer
#include <fsl_flexspi.h> Transfer structure for FLEXSPI.

Public Members

uint32_t deviceAddress
Operation device address.

flexspi_port_t port
Operation port.

flexspi_command_type_t cmdType
Execution command type.

uint8_t seqIndex
Sequence ID for command.

uint8_t SeqNumber
Sequence number for command.

uint32_t *data
Data buffer.

size_t dataSize
Data size in bytes.

struct _flexspi_handle
#include <fsl_flexspi.h> Transfer handle structure for FLEXSPI.

Public Members

uint32_t state
Internal state for FLEXSPI transfer

uint8_t *data
Data buffer.

size_t dataSize
Remaining Data size in bytes.

size_t transferTotalSize
Total Data size in bytes.

flexspi_transfer_callback_t completionCallback
Callback for users while transfer finish or error occurred

void *userData
FLEXSPI callback function parameter.

struct _flexspi_addr_map_config
#include <fsl_flexspi.h> Address mapping configuration structure.

Public Members

uint32_t addrStart
Remapping start address.

uint32_t addrEnd
Remapping end address.

uint32_t addrOffset
Address offset.

bool remapEnable
Enable address remapping.

struct ahbConfig

Public Members

uint8_t ahbGrantTimeoutCycle
Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCyle*1024 AHB clock cycles.

uint16_t ahbBusTimeoutCycle
Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle*1024 AHB clock cycles.

uint8_t resumeWaitCycle
Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]
AHB buffer size.

bool enableClearAHBBufferOpt
Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt
Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch
Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

2.28 FLEXSPI eDMA Driver

void FLEXSPI_TransferCreateHandleEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *flexspi_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle, *edma_handle_t* *rxDmaHandle)

Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_edma_handle_t structure
- callback – FLEXSPI callback, NULL means no callback.
- userData – User callback function data.
- txDmaHandle – User requested DMA handle for TX DMA transfer.
- rxDmaHandle – User requested DMA handle for RX DMA transfer.

void FLEXSPI_TransferUpdateSizeEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *flexspi_edma_transfer_nsize_t* nsize)

Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

See also:

flexspi_edma_transfer_nsize_t .

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_edma_handle_t structure
- nsize – FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_EDMAnSize1Bytes(one byte).

status_t FLEXSPI_TransferEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *flexspi_transfer_t* *xfer)

Transfers FLEXSPI data using an eDMA non-blocking method.

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure
- xfer – FLEXSPI transfer structure.

Return values

- kStatus_FLEXSPI_Busy – FLEXSPI is busy transfer.
- kStatus_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer.
- kStatus_Success – FLEXSPI successfully start edma transfer.

void FLEXSPI_TransferAbortEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle)

Aborts the transfer data using eDMA.

This function aborts the transfer data using eDMA.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure

status_t FLEXSPI_TransferGetTransferCountEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, size_t *count)

Gets the transferred counts of transfer.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure.
- count – Bytes transfer.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL_FLEXSPI_EDMA_DRIVER_VERSION

FLEXSPI EDMA driver version.

enum _flexspi_edma_ntransfer_size

eDMA transfer configuration

Values:

enumerator kFLEXPSI_EDMAnSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kFLEXPSI_EDMAnSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kFLEXPSI_EDMAnSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kFLEXPSI_EDMAnSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kFLEXPSI_EDMAnSize32Bytes

Source/Destination data transfer size is 32 bytes every time

typedef struct *_flexspi_edma_handle* flexspi_edma_handle_t

typedef void (*flexspi_edma_callback_t)(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *status_t* status, void *userData)

FLEXSPI eDMA transfer callback function for finish and error.

typedef enum *_flexspi_edma_ntransfer_size* flexspi_edma_transfer_nsize_t

eDMA transfer configuration

struct _flexspi_edma_handle

#include <fsl_flexspi_edma.h> FLEXSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *txDmaHandle

eDMA handler for FLEXSPI Tx.

edma_handle_t *rxDmaHandle

eDMA handler for FLEXSPI Rx.

size_t transferSize

Bytes need to transfer.

flexspi_edma_transfer_nsize_t nsize

eDMA SSIZE/DSIZE in each transfer.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t count

The transfer data count in a DMA request.

uint32_t state

Internal state for FLEXSPI eDMA transfer.

flexspi_edma_callback_t completionCallback

A callback function called after the eDMA transfer is finished.

void *userData

User callback parameter

2.29 FLEXSPI_FLR: Flexible Serial Peripheral Interface Follower Driver

IO mode enumeration of FLEXSPI FOLLOWER.

Values:

enumerator kFLEXSPI_SLV_IOMODE_SDRx4

enumerator kFLEXSPI_SLV_IOMODE_SDRx8

enumerator kFLEXSPI_SLV_IOMODE_DDRx4

enumerator kFLEXSPI_SLV_IOMODE_DDRx8

The read fetch size enumeration of FLEXSPI FOLLOWER.

Values:

enumerator kFLEXSPI_SLV_Read_Fetch_256Bytes

enumerator kFLEXSPI_SLV_Read_Fetch_512Bytes

enumerator kFLEXSPI_SLV_Read_Fetch_1KBytes

enumerator kFLEXSPI_SLV_Read_Fetch_2KBytes

The write watermark enumeration of FLEXSPI FOLLOWER.

Values:

enumerator kFLEXSPI_SLV_Write_Watermark_32Bytes

enumerator kFLEXSPI_SLV_Write_Watermark_64Bytes

enumerator kFLEXSPI_SLV_Write_Watermark_128Bytes

enumerator kFLEXSPI_SLV_Write_Watermark_256Bytes

Interrupt status flags of FLEXSPI FOLLOWER.

Values:

enumerator kFLEXSPI_SLV_WriteOverflowFlag
An IO RX FIFO overflow occurred during command/address/write data phase.

enumerator kFLEXSPI_SLV_ReadUnderflowFlag
IO TX FIFO underflow has occurred during a read command.

enumerator kFLEXSPI_SLV_ErrorCommandFlag
An unknown command has been received from the SPI bus.

enumerator kFLEXSPI_SLV_MailInterruptFlag
Mailbox interrupt.

enumerator kFLEXSPI_SLV_AllInterruptFlags
All flags.

typedef struct *_flexspi_slv_config* flexspi_slv_config_t
FLEXSPI FOLLOWER configuration structure.

typedef struct *_flexspi_slv_handle* flexspi_slv_handle_t

typedef void (*flexspi_slv_callback_t)(FLEXSPI_SLV_Type *base, *flexspi_slv_handle_t* *handle)
FLEXSPI FOLLOWER interrupt callback function.

FSL_FLEXSPI_SLV_DRIVER_VERSION
FLEXSPI FOLLOWER driver version.

FLEXSPI_SLV_CMD_DDR(**x**)

uint32_t FLEXSPI_SLV_GetInstance(FLEXSPI_SLV_Type *base)
Get the instance number for FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER base pointer.

void FLEXSPI_SLV_Init(FLEXSPI_SLV_Type *base, const *flexspi_slv_config_t* *config)
Initializes the FLEXSPI FOLLOWER module and internal state.

This function enables the clock for FLEXSPI FOLLOWER and also configures the FLEXSPI FOLLOWER with the input configure parameters. Users should call this function before any FLEXSPI FOLLOWER operations.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- config – FLEXSPI FOLLOWER configure structure.

void FLEXSPI_SLV_GetDefaultConfig(*flexspi_slv_config_t* *config)

Gets default settings for FLEXSPI FOLLOWER.

Parameters

- config – FLEXSPI FOLLOWER configuration structure.

void FLEXSPI_SLV_Deinit(FLEXSPI_SLV_Type *base)

Deinitializes the FLEXSPI FOLLOWER module.

Clears the FLEXSPI FOLLOWER state and FLEXSPI FOLLOWER module registers.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

static inline void FLEXSPI_SLV_SoftwareReset(FLEXSPI_SLV_Type *base)

Software reset for the FLEXSPI FOLLOWER logic.

This function does software reset for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

static inline void FLEXSPI_SLV_SetIOMode(FLEXSPI_SLV_Type *base, uint32_t ioMode)

Set IO mode for the FLEXSPI FOLLOWER module.

This function sets the IO mode flags for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- ioMode – Set IO Mode for FLEXSPI FOLLOWER

static inline void FLEXSPI_SLV_UpdateRWCmdBaseRange(FLEXSPI_SLV_Type *base)

Update RW CMD base address and range value for the FLEXSPI FOLLOWER module.

This function updates RW CMD base address and range value for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

static inline void FLEXSPI_SLV_SetRWCmdBaseAddr(FLEXSPI_SLV_Type *base, uint32_t addr1, uint32_t addr2)

Set RW command base address1 for the FLEXSPI FOLLOWER module.

This function sets the RW command base address1 for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- addr1 – The high 16-bit base address of the RW command0.
- addr2 – The high 16-bit base address of the RW command1.

static inline void FLEXSPI_SLV_SetAddrRange(FLEXSPI_SLV_Type *base, uint32_t index, uint32_t val)

Set address1/2 range for the FLEXSPI FOLLOWER module.

This function sets the address1/2 range for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- index – The index of RW command, 0 or 1.
- val – The size of the memory range in 1KB units.

static inline void FLEXSPI_SLV_SetReadWatermark(FLEXSPI_SLV_Type *base, uint32_t rxWatermark, bool enable)

Set read water mark level for the FLEXSPI FOLLOWER module.

This function sets read water mark level for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- val – Read watermark level in bytes

static inline void FLEXSPI_SLV_SetReadFetchSize(FLEXSPI_SLV_Type *base, uint32_t rxFetchSize)

Sets the maximum read size triggered by a single read command.

This function sets the maximum read size for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- rxFetchSize – The maximum read size triggered by a single read command.

static inline void FLEXSPI_SLV_SetWriteWatermark(FLEXSPI_SLV_Type *base, uint32_t txWatermark)

Set write water mark level for the FLEXSPI FOLLOWER module.

This function sets write water mark level for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- txWatermark – Write watermark level

static inline void FLEXSPI_SLV_MaskChipSelect(FLEXSPI_SLV_Type *base, uint32_t mask)

Set CS mask value for the FLEXSPI FOLLOWER module.

This function sets CS mask value for the FLEXSPI FOLLOWER.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- mask – 0 - Not masked, 1 - Masked.

static inline void FLEXSPI_SLV_EnableInterrupts(FLEXSPI_SLV_Type *base, uint32_t mask)

Enables the FLEXSPI FOLLOWER interrupts.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- mask – FLEXSPI FOLLOWER interrupt source.

static inline void FLEXSPI_SLV_DisableInterrupts(FLEXSPI_SLV_Type *base, uint32_t mask)

Disable the FLEXSPI FOLLOWER interrupts.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- mask – FLEXSPI FOLLOWER interrupt source.

static inline uint32_t FLEXSPI_SLV_GetEnabledInterrupts(FLEXSPI_SLV_Type *base)

Get the FLEXSPI FOLLOWER enabled interrupts.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

static inline void FLEXSPI_SLV_EnableMailInterrupt(FLEXSPI_SLV_Type *base, bool enable)

Enable the FLEXSPI FOLLOWER mailbox interrupts.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- enable – Whether enable the mailbox interrupt.

static inline bool FLEXSPI_SLV_GetEnabledMailInterrupt(FLEXSPI_SLV_Type *base)

Return whether the FLEXSPI FOLLOWER enables the mailbox interrupt.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

static inline void FLEXSPI_SLV_GetOutOfRangeCounts(FLEXSPI_SLV_Type *base, size_t *rdCount, size_t *wrCount)

Gets the SPI leader read/write out-of-allowed-range count.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- rdCount – **[out]** Pointer through which the current number in the read out-of-allowed-range counter is returned. Pass NULL if this value is not required.
- wrCount – **[out]** Pointer through which the current number in the write out-of-allowed-range counter is returned Pass NULL if this value is not required.

static inline uint32_t FLEXSPI_SLV_GetInterruptStatusFlags(FLEXSPI_SLV_Type *base)

Get the FLEXSPI FOLLOWER interrupt status flags.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

Return values

Interrupt – status flag, use status flag to AND the bit mask could get the related status.

static inline uint32_t FLEXSPI_SLV_GetMailboxData(FLEXSPI_SLV_Type *base, uint32_t index)

Get the FLEXSPI FOLLOWER mailbox data.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- index – The index of the mail interrupt register

Returns

Return the FLEXSPI FOLLOWER mailbox data

static inline void FLEXSPI_SLV_ClearInterruptStatusFlags(FLEXSPI_SLV_Type *base, uint32_t mask)

Clear the FLEXSPI FOLLOWER interrupt status flags.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- mask – FLEXSPI FOLLOWER interrupt source.

static inline void FLEXSPI_SLV_ClearMailInterruptFlag(FLEXSPI_SLV_Type *base)

Clear the FLEXSPI FOLLOWER mailbox interrupt flag.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

static inline bool FLEXSPI_SLV_GetAXIWriteBusyStatus(FLEXSPI_SLV_Type *base)

Returns whether the current AXI write leader is busy with a write command.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

Return values

- true – The current AXI write leader is busy.
- false – The current AXI write leader is not busy.

static inline bool FLEXSPI_SLV_GetAXIReadIdleStatus(FLEXSPI_SLV_Type *base)

Returns whether the AXI read leader is busy with a read request or else idle with no pending AXI read request.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

Return values

- true – The current AXI read leader is idle.
- false – The current AXI read leader is busy.

static inline bool FLEXSPI_SLV_GetRegReadWriteIdleStatus(FLEXSPI_SLV_Type *base)

Returns whether the SPI to read/write register queue is idle.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

Return values

- true – The SPI to read/write register queue is idle.
- false – The SPI to read/write register queue is busy.

static inline bool FLEXSPI_SLV_GetSEQIdleStatus(FLEXSPI_SLV_Type *base)

Returns whether the SEQ control logic is idle or else busy with an ongoing SPI request.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

Return values

- true – The SEQ control logic is idle.
- false – The SEQ control logic is busy.

static inline bool FLEXSPI_SLV_GetModuleBusyStatus(FLEXSPI_SLV_Type *base)

Returns whether the FLEXSPI FOLLOWER module is busy.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.

Return values

- true – The SEQ control logic is busy.
- false – The SEQ control logic is idle.

static inline void FLEXSPI_SLV_SetReadMemCommand(FLEXSPI_SLV_Type *base, uint32_t index, uint16_t command, uint16_t dummyCycle)

Sets the read memory command.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- index – The read command setting register index.
- command – The read command value.
- dummyCycle – The dummy cycle value of the read command.

static inline void FLEXSPI_SLV_SetWriteMemCommand(FLEXSPI_SLV_Type *base, uint32_t index, uint32_t command)

Sets the write memory command.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- index – The write command setting register index.
- command – The write command value.

static inline void FLEXSPI_SLV_SetReadRegCommand(FLEXSPI_SLV_Type *base, uint16_t command, uint16_t dummyCycle)

Sets the read register command.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address
- command – The read command value.
- dummyCycle – The dummy cycle value of the read command.

static inline void FLEXSPI_SLV_SetWriteRegCommand(FLEXSPI_SLV_Type *base, uint32_t command)

Sets the write register command.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- command – The write register command value.

void FLEXSPI_SLV_InterruptCreateHandle(FLEXSPI_SLV_Type *base, *flexspi_slv_handle_t* *handle, *flexspi_slv_callback_t* callback, uint32_t interruptMask)

Initializes the FLEXSPI FOLLOWER handle which is used in transactional functions.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- handle – Pointer to flexspi_slv_handle_t structure to store the interrupt state.
- callback – Pointer to user callback function.
- interruptMask – Interrupt mask to enable during handle creation. Use enumeration values ORed.

void FLEXSPI_SLV_HandleIRQ(FLEXSPI_SLV_Type *base, *flexspi_slv_handle_t* *handle)

Master interrupt handler.

Parameters

- base – FLEXSPI FOLLOWER peripheral base address.
- handle – Pointer to flexspi_slv_handle_t structure.

uint32_t baseAddr1
: Read/Write CMD1 Base Address.

uint32_t baseAddr2
: Read/Write CMD2 Base Address.

uint32_t addrRange1
: Read/Write CMD1 Addr Range.

uint32_t addrRange2
: Read/Write CMD2 Addr Range.

uint8_t ioMode
: IO mode control - SDRx4, SDRx8, DDRx4, DDRx8.

uint8_t rxFetchSize
: Specifies the maximum read size triggered by a single read command.

uint8_t rxWatermark
: Triggers a new AXI read to fetch more data through the IP AXI header.

uint8_t txWatermark
: Specifies the watermark value. During the write command, if pending write data equals or exceeds the watermark level, it triggers a new AXI write.

uint16_t readRegCmd
: Read register command.

uint16_t readRegDummyCycle
: Read register dymmy cycle.

uint16_t writeRegCmd
: Write register command.

uint16_t readMemCmd1
: Read memory command1.

uint16_t readMemDummyCycle1
: Read memory dymmy cycle1.

uint16_t readMemCmd2
: Read memory command2.

uint16_t readMemDummyCycle2
: Read memory dymmy cycle2.

uint16_t writeMemCmd1
: Write memory command1.

uint16_t writeMemCmd2
: Write memory command2.

uint32_t intrMask
: Interrupt state for FLEXSPI FOLLOWER.

flexspi_slv_callback_t callback
: Callback for users while mailbox received or error occurred.

struct __flexspi__slv__config
: *#include <fsl_flexspi_flr.h>* FLEXSPI FOLLOWER configuration structure.

struct __flexspi__slv__handle
: *#include <fsl_flexspi_flr.h>* Interrupt handle structure for FLEXSPI FOLLOWER.

2.30 GPT: General Purpose Timer

void GPT_Init(GPT_Type *base, const *gpt_config_t* *initConfig)

Initialize GPT to reset state and initialize running mode.

Parameters

- base – GPT peripheral base address.
- initConfig – GPT mode setting configuration.

void GPT_Deinit(GPT_Type *base)

Disables the module and gates the GPT clock.

Parameters

- base – GPT peripheral base address.

void GPT_GetDefaultConfig(*gpt_config_t* *config)

Fills in the GPT configuration structure with default settings.

The default values are:

```
config->clockSource = kGPT_ClockSource_Periph;
config->divider = 1U;
config->enableRunInStop = true;
config->enableRunInWait = true;
config->enableRunInDoze = false;
config->enableRunInDbg = false;
config->enableFreeRun = false;
config->enableMode  = true;
```

Parameters

- config – Pointer to the user configuration structure.

static inline void GPT_SoftwareReset(GPT_Type *base)

Software reset of GPT module.

Parameters

- base – GPT peripheral base address.

static inline void GPT_SetClockSource(GPT_Type *base, *gpt_clock_source_t* gptClkSource)

Set clock source of GPT.

Parameters

- base – GPT peripheral base address.
- gptClkSource – Clock source (see gpt_clock_source_t typedef enumeration).

static inline *gpt_clock_source_t* GPT_GetClockSource(GPT_Type *base)

Get clock source of GPT.

Parameters

- base – GPT peripheral base address.

Returns

clock source (see gpt_clock_source_t typedef enumeration).

static inline void GPT_SetClockDivider(GPT_Type *base, uint32_t divider)

Set pre scaler of GPT.

Parameters

- base – GPT peripheral base address.

- divider – Divider of GPT (1-4096).

static inline uint32_t GPT_GetClockDivider(GPT_Type *base)

Get clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

clock divider in GPT module (1-4096).

static inline void GPT_SetOscClockDivider(GPT_Type *base, uint32_t divider)

OSC 24M pre-scaler before selected by clock source.

Parameters

- base – GPT peripheral base address.
- divider – OSC Divider(1-16).

static inline uint32_t GPT_GetOscClockDivider(GPT_Type *base)

Get OSC 24M clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

OSC clock divider in GPT module (1-16).

static inline void GPT_StartTimer(GPT_Type *base)

Start GPT timer.

Parameters

- base – GPT peripheral base address.

static inline void GPT_StopTimer(GPT_Type *base)

Stop GPT timer.

Parameters

- base – GPT peripheral base address.

static inline uint32_t GPT_GetCurrentTimerCount(GPT_Type *base)

Reads the current GPT counting value.

Parameters

- base – GPT peripheral base address.

Returns

Current GPT counter value.

static inline void GPT_SetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel, *gpt_input_operation_mode_t* mode)

Set GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).
- mode – GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

static inline *gpt_input_operation_mode_t* GPT_GetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

static inline uint32_t GPT_GetInputCaptureValue(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT input capture value of certain channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

static inline void GPT_SetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel, *gpt_output_operation_mode_t* mode)

Set GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
- mode – GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

static inline *gpt_output_operation_mode_t* GPT_GetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

static inline void GPT_SetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel, uint32_t value)

Set GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.

- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
- value – GPT output compare value.

static inline uint32_t GPT_GetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output compare value.

static inline void GPT_ForceOutput(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Force GPT output action on output compare channel, ignoring comparator.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

static inline void GPT_EnableInterrupts(GPT_Type *base, uint32_t mask)

Enables the selected GPT interrupts.

Parameters

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

static inline void GPT_DisableInterrupts(GPT_Type *base, uint32_t mask)

Disables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration gpt_interrupt_enable_t

static inline uint32_t GPT_GetEnabledInterrupts(GPT_Type *base)

Gets the enabled GPT interrupts.

Parameters

- base – GPT peripheral base address

Returns

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

static inline uint32_t GPT_GetStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Get GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see gpt_status_flag_t for bit definition).

Returns
GPT status, each bit represents one status flag.

static inline void GPT_ClearStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)
Clears the GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see gpt_status_flag_t for bit definition).

FSL_GPT_DRIVER_VERSION

enum _gpt_clock_source
List of clock sources.

Note: Actual number of clock sources is SoC dependent

Values:

enumerator kGPT_ClockSource_Off
GPT Clock Source Off.

enumerator kGPT_ClockSource_Periph
GPT Clock Source from Peripheral Clock.

enumerator kGPT_ClockSource_HighFreq
GPT Clock Source from High Frequency Reference Clock.

enumerator kGPT_ClockSource_Ext
GPT Clock Source from external pin.

enumerator kGPT_ClockSource_LowFreq
GPT Clock Source from Low Frequency Reference Clock.

enumerator kGPT_ClockSource_Osc
GPT Clock Source from Crystal oscillator.

enum _gpt_input_capture_channel
List of input capture channel number.

Values:

enumerator kGPT_InputCapture_Channel1
GPT Input Capture Channel1.

enumerator kGPT_InputCapture_Channel2
GPT Input Capture Channel2.

enum _gpt_input_operation_mode
List of input capture operation mode.

Values:

enumerator kGPT_InputOperation_Disabled
Don't capture.

enumerator kGPT_InputOperation_RiseEdge
Capture on rising edge of input pin.

enumerator kGPT_InputOperation_FallEdge
Capture on falling edge of input pin.

enumerator kGPT_InputOperation_BothEdge
Capture on both edges of input pin.

enum _gpt_output_compare_channel
List of output compare channel number.

Values:

enumerator kGPT_OutputCompare_Channel1
Output Compare Channel1.

enumerator kGPT_OutputCompare_Channel2
Output Compare Channel2.

enumerator kGPT_OutputCompare_Channel3
Output Compare Channel3.

enum _gpt_output_operation_mode
List of output compare operation mode.

Values:

enumerator kGPT_OutputOperation_Disconnected
Don't change output pin.

enumerator kGPT_OutputOperation_Toggle
Toggle output pin.

enumerator kGPT_OutputOperation_Clear
Set output pin low.

enumerator kGPT_OutputOperation_Set
Set output pin high.

enumerator kGPT_OutputOperation_Activelow
Generate a active low pulse on output pin.

enum _gpt_interrupt_enable
List of GPT interrupts.

Values:

enumerator kGPT_OutputCompare1InterruptEnable
Output Compare Channel1 interrupt enable

enumerator kGPT_OutputCompare2InterruptEnable
Output Compare Channel2 interrupt enable

enumerator kGPT_OutputCompare3InterruptEnable
Output Compare Channel3 interrupt enable

enumerator kGPT_InputCapture1InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_InputCapture2InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_RollOverFlagInterruptEnable
Counter rolled over interrupt enable

enum _gpt_status_flag
Status flag.

Values:

enumerator kGPT_OutputCompare1Flag
Output compare channel 1 event.

enumerator kGPT_OutputCompare2Flag
Output compare channel 2 event.

enumerator kGPT_OutputCompare3Flag
Output compare channel 3 event.

enumerator kGPT_InputCapture1Flag
Input Capture channel 1 event.

enumerator kGPT_InputCapture2Flag
Input Capture channel 2 event.

enumerator kGPT_RollOverFlag
Counter reaches maximum value and rolled over to 0 event.

typedef enum *_gpt_clock_source* gpt_clock_source_t
List of clock sources.

Note: Actual number of clock sources is SoC dependent

typedef enum *_gpt_input_capture_channel* gpt_input_capture_channel_t
List of input capture channel number.

typedef enum *_gpt_input_operation_mode* gpt_input_operation_mode_t
List of input capture operation mode.

typedef enum *_gpt_output_compare_channel* gpt_output_compare_channel_t
List of output compare channel number.

typedef enum *_gpt_output_operation_mode* gpt_output_operation_mode_t
List of output compare operation mode.

typedef enum *_gpt_interrupt_enable* gpt_interrupt_enable_t
List of GPT interrupts.

typedef enum *_gpt_status_flag* gpt_status_flag_t
Status flag.

typedef struct *_gpt_init_config* gpt_config_t
Structure to configure the running mode.

struct _gpt_init_config
#include <fsl_gpt.h> Structure to configure the running mode.

Public Members

gpt_clock_source_t clockSource
clock source for GPT module.

uint32_t divider
clock divider (prescaler+1) from clock source to counter.

bool enableFreeRun
true: FreeRun mode, false: Restart mode.

bool enableRunInWait
GPT enabled in wait mode.

bool enableRunInStop
GPT enabled in stop mode.

bool enableRunInDoze
GPT enabled in doze mode.

bool enableRunInDbg
GPT enabled in debug mode.

bool enableMode
true: counter reset to 0 when enabled; false: counter retain its value when enabled.

2.31 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.

2.32 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;
```

(continues on next page)

(continued from previous page)

```
config->ignoreS0S1Error     = false;
config->offline             = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C_Init().

Parameters

- config – **[out]** User provided configuration structure for default values. Refer to i3c_config_t.

void I3C_Init(I3C_Type *base, const *i3c_config_t* *config, uint32_t sourceClock_Hz)

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- config – User provided peripheral configuration. Use I3C_GetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

struct _i3c_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay

I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay
I3C SCL delay after Repeated START.

uint8_t masterDynamicAddress
Main master dynamic address configuration.

uint32_t maxWriteLength
Maximum write length.

uint32_t maxReadLength
Maximum read length.

bool enableSlave
Whether to enable slave.

uint8_t staticAddr
Static address.

uint16_t vendorID
Device vendor ID(manufacture ID).

uint32_t partNumber
Device part number info

uint8_t dcr
Device characteristics register information.

uint8_t bcr
Bus characteristics register information.

uint8_t hdrMode
Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest
Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error
Whether to ignore S0/S1 error in SDR mode.

bool offline
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop
Whether to assert start/stop status only the time slave is addressed.

2.33 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.

i3c_start_scl_delay_t startSclDelay
I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay
I3C SCL delay after Repeated START.

struct __i3c_master_transfer_callback
#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)
Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)
IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_handle_t* *handle, *status_t* completionStatus, void *userData)
Transfer complete callback

struct __i3c_master_transfer
#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
Bit mask of options for the transfer. See enumeration _i3c_master_transfer_flags for available options. Set to 0 or kI3C_TransferDefaultFlag for normal transfers.

uint8_t slaveAddress
The 7-bit slave address.

i3c_direction_t direction
Either kI3C_Read or kI3C_Write.

uint32_t subaddress
Sub address. Transferred MSB first.

size_t subaddressSize
Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
Pointer to data to transfer.

size_t dataSize
Number of bytes to transfer.

i3c_bus_type_t busType
bus type.

i3c_ibi_response_t ibiResponse
ibi response during transfer.

struct _i3c_master_handle

#include <fsl_i3c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint32_t remainingBytes

Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps

Read termination operation.

i3c_master_transfer_t transfer

Copy of the current transfer info.

uint8_t ibiAddress

Slave address which request IBI.

uint8_t *ibiBuff

Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize

IBI payload size.

i3c_ibi_type_t ibiType

IBI type.

i3c_master_transfer_callback_t callback

Callback functions pointer.

void *userData

Application data passed to callback.

2.34 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.

2.35 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

2.36 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.

2.37 IEE: Inline Encryption Engine

FSL_IEE_DRIVER_VERSION
IEE driver version. Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0

- Initial version

- Version 2.1.0
  - Add region lock function IEE_LockRegionConfig() and driver clock control
- Version 2.1.1
  - Fixed MISRA issues.
- Version 2.2.0
  - Add ELE (EdgeLock Enclave) key provisioning feature.

enum _iee_region

IEE region.

Values:

enumerator kIEE_Region0

IEE region 0

enumerator kIEE_Region1

IEE region 1

enumerator kIEE_Region2

IEE region 2

enumerator kIEE_Region3

IEE region 3

enumerator kIEE_Region4

IEE region 4

enumerator kIEE_Region5

IEE region 5

enumerator kIEE_Region6

IEE region 6

enumerator kIEE_Region7

IEE region 7

enum _iee_aes_bypass

IEE AES enablement/bypass.

Values:

enumerator kIEE_AesUseMdField

AES encryption/decryption enabled

enumerator kIEE_AesBypass

AES encryption/decryption bypass

enum _iee_aes_mode

IEE AES mode.

Values:

enumerator kIEE_ModeNone

AES NONE mode

enumerator kIEE_ModeAesXTS

AES XTS mode

enumerator kIEE_ModeAesCTRWAddress
AES CTR w address binding mode

enumerator kIEE_ModeAesCTRWOAddress
AES CTR w/o address binding mode

enumerator kIEE_ModeAesCTRkeystream
AES CTR keystream only

enum _iee_aes_key_size
IEE AES key size.

Values:

enumerator kIEE_AesCTR128XTS256
AES 128 bits (CTR), 256 bits (XTS)

enumerator kIEE_AesCTR256XTS512
AES 256 bits (CTR), 512 bits (XTS)

enum _iee_aes_key_num
IEE AES key number.

Values:

enumerator kIEE_AesKey1
AES Key 1

enumerator kIEE_AesKey2
AES Key 2

typedef enum *_iee_region* iee_region_t
IEE region.

typedef enum *_iee_aes_bypass* iee_aes_bypass_t
IEE AES enablement/bypass.

typedef enum *_iee_aes_mode* iee_aes_mode_t
IEE AES mode.

typedef enum *_iee_aes_key_size* iee_aes_key_size_t
IEE AES key size.

typedef enum *_iee_aes_key_num* iee_aes_key_num_t
IEE AES key number.

typedef struct *_iee_config* iee_config_t
IEE configuration structure.

void IEE_Init(IEE_Type *base)
Resets IEE module to factory default values.

This function performs hardware reset of IEE module. Attributes and keys of all regions are cleared.

Parameters

- base – IEER peripheral address.

void IEE_GetDefaultConfig(*iee_config_t* *config)
Loads default values to the IEE configuration structure.

Loads default values to the IEE region configuration structure. The default values are as follows.

```
config->bypass = kIEE_AesUseMdField;
config->mode = kIEE_ModeNone;
config->keySize = kIEE_AesCTR128XTS256;
config->pageOffset = 0U;
```

Parameters

- config – Configuration for the selected IEE region.

void IEE_SetRegionConfig(IEE_Type *base, *iee_region_t* region, *iee_config_t* *config)

Sets the IEE module according to the configuration structure.

This function configures IEE region according to configuration structure.

Parameters

- base – IEE peripheral address.
- region – Selection of the IEE region to be configured.
- config – Configuration for the selected IEE region.

status_t IEE_SetRegionKey(IEE_Type *base, *iee_region_t* region, *iee_aes_key_num_t* keyNum, const uint8_t *key, size_t keySize)

Sets the IEE module key.

This function sets specified AES key for the given region.

Parameters

- base – IEE peripheral address.
- region – Selection of the IEE region to be configured.
- keyNum – Selection of AES KEY1 or KEY2.
- key – AES key.
- keySize – Size of AES key.

static inline uint32_t IEE_GetOffset(uint32_t addressIee, uint32_t addressMemory)

Computes IEE offset to be set for specifed memory location.

This function calculates offset that must be set for IEE region to access physical memory location.

Parameters

- addressIee – Address of IEE peripheral.
- addressMemory – Address of physical memory location.

void IEE_LockRegionConfig(IEE_Type *base, *iee_region_t* region)

Lock the IEE region configuration.

This function locks IEE region registers for Key, Offset and Attribute. Only system reset can clear the Lock bit.

Parameters

- base – IEE peripheral address.
- region – Selection of the IEE region to be locked.

struct _iee_config

#include <fsl_iee.h> IEE configuration structure.

Public Members

iee_aes_bypass_t bypass
AES encryption/decryption bypass

iee_aes_mode_t mode
AES mode

iee_aes_key_size_t keySize
size of AES key

uint32_t pageOffset
Offset to physical memory location from IEE start address

2.38 Ieer

FSL_IEE_APC_DRIVER_VERSION
IEE_APC driver version. Version 2.0.2.

Current version: 2.0.2

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Fixed MISRA issues.
- Version 2.0.2
  - Update to newer version of implementation in HW.

enum _iee_apc_region
APC IEE regions.

Values:

enumerator kIEE_APC_Region0
APC IEE region 0

enumerator kIEE_APC_Region1
APC IEE region 1

enumerator kIEE_APC_Region2
APC IEE region 2

enumerator kIEE_APC_Region3
APC IEE region 3

enumerator kIEE_APC_Region4
APC IEE region 4

enumerator kIEE_APC_Region5
APC IEE region 5

enumerator kIEE_APC_Region6
APC IEE region 6

enumerator kIEE_APC_Region7
APC IEE region 7

enum _apc_iee_domain

APC IEE domains.

Values:

enumerator kIEE_APC_Domain0

APC IEE region 0

enumerator kIEE_APC_Domain1

APC IEE region 1

typedef enum *_iee_apc_region* iee_apc_region_t

APC IEE regions.

typedef enum *_apc_iee_domain* iee_apc_domain_t

APC IEE domains.

void IEE_APC_GlobalEnable(IEE_APC_Type *base)

Enable the APC IEE Region setting.

This function enables IOMUXC LPSR GPR and APC IEE for setting the region.

Parameters

- base – APC IEE peripheral address.

void IEE_APC_GlobalDisable(IEE_APC_Type *base)

Disables the APC IEE Region setting.

This function disables IOMUXC LPSR GPR and APC IEE for setting the region.

Parameters

- base – APC IEE peripheral address.

status_t IEE_APC_SetRegionConfig(IEE_APC_Type *base, *iee_apc_region_t* region, uint32_t startAddr, uint32_t endAddr)

Sets the APC IEE Memory Region Descriptors.

This function configures APC IEE Memory Region Descriptor according to region configuration structure.

Parameters

- base – APC IEE peripheral address.
- region – Selection of the APC IEE region to be configured.
- startAddr – Start encryption adress for the selected APC IEE region.
- endAddr – End encryption adress for the selected APC IEE region.

status_t IEE_APC_LockRegionConfig(IEE_APC_Type *base, *iee_apc_region_t* region, *iee_apc_domain_t* domain)

Lock the LPSR GPR and APC IEE configuration.

This function locks writting to IOMUXC LPSR GPR and APC IEE encryption region setting registers. Only system reset can clear the LPSR GPR and APC IEE-RDC_D0/1 Lock bit

Parameters

- base – APC IEE peripheral address.
- region – Selection of the APC IEE region to be locked.
- domain – Core domain ID

void IEE_APC_RegionEnable(IEE_APC_Type *base, *iee_apc_region_t* region)

Enable the IEE encryption/decryption and can lock this setting.

This function enables encryption/decryption by writting to IOMUXC LPSR GPR.

Parameters

- base – APC IEE peripheral address.
- region – Selection of the APC IEE region to be enabled.

2.39 KPP: KeyPad Port Driver

void KPP_Init(KPP_Type *base, *kpp_config_t* *configure)

KPP initialize. This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

- base – KPP peripheral base address.
- configure – The KPP configuration structure pointer.

void KPP_Deinit(KPP_Type *base)

Deinitializes the KPP module and gates the clock. This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

- base – KPP peripheral base address.

static inline void KPP_EnableInterrupts(KPP_Type *base, uint16_t mask)

Enable the interrupt.

Parameters

- base – KPP peripheral base address.
- mask – KPP interrupts to enable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

static inline void KPP_DisableInterrupts(KPP_Type *base, uint16_t mask)

Disable the interrupt.

Parameters

- base – KPP peripheral base address.
- mask – KPP interrupts to disable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

static inline uint16_t KPP_GetStatusFlag(KPP_Type *base)

Gets the KPP interrupt event status.

Parameters

- base – KPP peripheral base address.

Returns

The status of the KPP. Application can use the enum type in the "kpp_interrupt_enable_t" to get the right status of the related event.

static inline void KPP_ClearStatusFlag(KPP_Type *base, uint16_t mask)

Clears KPP status flag.

Parameters

- base – KPP peripheral base address.
- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

static inline void KPP_SetSynchronizeChain(KPP_Type *base, uint16_t mask)

Set KPP synchronization chain.

Parameters

- base – KPP peripheral base address.
- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_sync_operation_t.

status_t KPP_keyPressScanning(KPP_Type *base, uint8_t *data, uint32_t clockSrc_Hz)

Keypad press scanning.

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

Parameters

- base – KPP peripheral base address.
- data – KPP key press scanning data. The data buffer should be prepared with length at least equal to KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX. the data pointer is recommended to be a array like uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]. for example the data[2] = 4, that means in column 1 row 2 has a key press event.
- clockSrc_Hz – Source clock.

Return values

kStatus_Success – kpp press scan succeed.

FSL_KPP_DRIVER_VERSION

KPP driver version.

enum _kpp_interrupt_enable

List of interrupts supported by the peripheral. This enumeration uses one-bot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Values:

enumerator kKPP_keyDepressInterrupt

Keypad depress interrupt source

enumerator kKPP_keyReleaseInterrupt

Keypad release interrupt source

enum _kpp_sync_operation

Lists of KPP synchronize chain operation.

Values:

enumerator kKPP_ClearKeyDepressSyncChain

Keypad depress interrupt status.

enumerator kKPP_SetKeyReleasesSyncChain

Keypad release interrupt status.

typedef enum *_kpp_interrupt_enable* kpp_interrupt_enable_t

List of interrupts supported by the peripheral. This enumeration uses one-bot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

typedef enum *_kpp_sync_operation* kpp_sync_operation_t
Lists of KPP synchronize chain operation.

typedef struct *_kpp_config* kpp_config_t
Lists of KPP status.

KPP_KEYPAD_COLUMNNUM_MAX

KPP_KEYPAD_ROWNUM_MAX

struct _kpp_config
#include <fsl_kpp.h> Lists of KPP status.

Public Members

uint8_t activeRow
The row number: bit 7 ~ 0 represents the row 7 ~ 0.

uint8_t activeColumn
The column number: bit 7 ~ 0 represents the column 7 ~ 0.

uint16_t interrupt
KPP interrupt source. A logical OR of "kpp_interrupt_enable_t".

2.40 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.

2.41 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.

enum _lpadc_tune_value

Define enumeration of tune value.

Values:

enumerator kLPADC_TuneValue0

Tune value 0.

enumerator kLPADC_TuneValue1

Tune value 1.

enumerator kLPADC_TuneValue2

Tune value 2.

enumerator kLPADC_TuneValue3

Tune value 3.

typedef enum *_lpadc_sample_scale_mode* lpadc_sample_scale_mode_t

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

typedef enum *_lpadc_sample_channel_mode* lpadc_sample_channel_mode_t

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

typedef enum *_lpadc_hardware_average_mode* lpadc_hardware_average_mode_t

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

typedef enum *_lpadc_sample_time_mode* lpadc_sample_time_mode_t

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

typedef enum *_lpadc_hardware_compare_mode* lpadc_hardware_compare_mode_t

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

typedef enum *_lpadc_conversion_resolution_mode* lpadc_conversion_resolution_mode_t

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc_sample_channel_mode_t

typedef enum *_lpadc_conversion_average_mode* lpadc_conversion_average_mode_t

Define enumeration of conversion averages mode.

Configure the 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 enum *_lpadc_tune_value* lpadc_tune_value_t

Define enumeration of tune value.

typedef struct *_lpadc_calibration_value* lpadc_calibration_value_t

A structure of calibration value.

LPADC_CONVERSION_COMPLETE_TIMEOUT

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

LPADC_CALIBRATION_READY_TIMEOUT

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

LPADC_GAIN_CAL_READY_TIMEOUT

Max loops to wait for LPADC gain calibration GAIN_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN_CAL ready. This parameter defines how many loops to check the gain calibration GAIN_CAL ready. If defined as 0, driver will wait forever until ready.

ADC_OFSTRIM_OFSTRIM_MAX

ADC_OFSTRIM_OFSTRIM_SIGN

LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

void LPADC_Init(ADC_Type *base, const *lpadc_config_t* *config)

Initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See "lpadc_config_t".

void LPADC_GetDefaultConfig(*lpadc_config_t* *config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode        = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay            = 0x80;
config->referenceVoltageSource  = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode          = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause         = false;
config->convPauseDelay          = 0U;
config->FIFOWatermark           = 0U;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_Deinit(ADC_Type *base)

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_Enable(ADC_Type *base, bool enable)

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC_DoResetFIFO(ADC_Type *base)

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_DoResetConfig(ADC_Type *base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns
status flags' mask. See to _lpadc_status_flags.

static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clear status flags.

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to _lpadc_status_flags.

static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Parameters

- base – LPADC peripheral base address.

Returns
The OR'ed value of _lpadc_trigger_status_flags.

static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of _lpadc_trigger_status_flags.

static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to _lpadc_interrupt_enable.

static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to _lpadc_interrupt_enable.

static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns
The count of result kept in conversion FIFO.

bool LPADC_GetConvResult(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns
Status whether FIFO entry is valid.

void LPADC_GetConvResultBlocking(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const *lpadc_conv_trigger_config_t* *config)

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to lpadc_conv_trigger_config_t.

void LPADC_GetDefaultConvTriggerConfig(*lpadc_conv_trigger_config_t* *config)

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId       = 0U;
config->delayPower            = 0U;
config->priority              = 0U;
config->channelAFIFOSelect    = 0U;
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_SetOffsetValue(ADC_Type *base, int16_t value)

Set trim value for offset.

Note: For 16-bit conversions, each increment is 1/2 LSB resulting in a programmable offset range of -256 LSB to 255.5 LSB; For 12-bit conversions, each increment is 1/32 LSB resulting in a programmable offset range of -16 LSB to 15.96875 LSB.

Parameters

- base – LPADC peripheral base address.
- value – Offset trim value, is a 10-bit signed value between -512 and 511.

static inline void LPADC_GetOffsetValue(ADC_Type *base, int16_t *pValue)

Get trim value of offset.

Parameters

- base – LPADC peripheral base address.
- pValue – Pointer to the variable in type of int16_t to store offset value.

static inline void LPADC_EnableOffsetCalibration(ADC_Type *base, bool enable)

Enable the offset calibration function.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetCalibrationMode(ADC_Type *base, *lpadc_offset_calibration_mode_t* mode)

Set offset calibration mode.

Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to lpadc_offset_calibration_mode_t .

status_t LPADC_DoOffsetCalibration(ADC_Type *base)

Do offset calibration.

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_PrepareAutoCalibration(ADC_Type *base)

Prepare auto calibration, LPADC_FinishAutoCalibration has to be called before using the LPADC. LPADC_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

- base – LPADC peripheral base address.

status_t LPADC_FinishAutoCalibration(ADC_Type *base)

Finish auto calibration start with LPADC_PrepareAutoCalibration.

Note: This feature is used for LPADC with CTRL[CALOFSMODE].

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_GetCalibrationValue(ADC_Type *base, *lpadc_calibration_value_t* *ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

Note: Please note the ADC will be disabled temporary.

Note: This function should be used after finish calibration.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure, this memory block should be always powered on even in low power modes.

status_t LPADC_SetCalibrationValue(ADC_Type *base, const *lpadc_calibration_value_t* *ptrCalibrationValue)

Set calibration value into ADC calibration registers.

Note: Please note the ADC will be disabled temporary.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure which contains ADC's calibration value.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_RequestHighSpeedModeTrim(ADC_Type *base)

Request high speed mode trim calculation.

Parameters

- base – LPADC peripheral base address.

static inline int8_t LPADC_GetHighSpeedTrimValue(ADC_Type *base)

Get high speed mode trim value, the result is a 5-bit signed value between -16 and 15.

Note: The high speed mode trim value is used to minimize offset for high speed conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The calculated high speed mode trim value.

static inline void LPADC_SetHighSpeedTrimValue(ADC_Type *base, int8_t trimValue)

Set high speed mode trim value.

Note: If is possible to set the trim value manually, but it is recommended to use the LPADC_RequestHighSpeedModeTrim.

Parameters

- base – LPADC peripheral base address.
- trimValue – The trim value to be set.

static inline void LPADC_EnableHighSpeedConversionMode(ADC_Type *base, bool enable)

Enable/disable high speed conversion mode, if enabled conversions complete 2 or 3 ADCK cycles sooner compared to conversion cycle counts when high speed mode is disabled.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable high speed conversion mode:
  - **true** Enable high speed conversion mode;
  - **false** Disable high speed conversion mode.

static inline void LPADC_EnableExtraCycle(ADC_Type *base, bool enable)

Enable/disable an additional ADCK cycle to conversion.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable an additional ADCK cycle to conversion:
  - **true** Enable an additional ADCK cycle to conversion;
  - **false** Disable an additional ADCK cycle to conversion.

static inline void LPADC_SetTuneValue(ADC_Type *base, *lpadc_tune_value_t* tuneValue)

Set tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.
- tuneValue – The tune value to be set, please refer to lpadc_tune_value_t.

static inline *lpadc_tune_value_t* LPADC_GetTuneValue(ADC_Type *base)

Get tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The tune value, please refer to lpadc_tune_value_t.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.9.4.

struct lpadc_config_t

#include <fsl_lpadc.h> LPADC global configuration.

This structure would used to keep the settings for initialization.

Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay * 4) ADCK cycles must result in a longer delay than the analog startup time.

lpadc_reference_voltage_source_t referenceVoltageSource

Selects the voltage reference high used for conversions.

lpadc_power_level_mode_t powerLevelMode

Power Configuration Selection.

lpadc_trigger_priority_policy_t triggerPriorityPolicy

Control how higher priority triggers are handled, see to lpadc_trigger_priority_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.

2.42 LPI2C: Low Power Inter-Integrated Circuit Driver

void LPI2C_DriverIRQHandler(uint32_t instance)

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

Parameters

- instance – LPI2C instance.

FSL_LPI2C_DRIVER_VERSION

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy

The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle

The slave driver is idle.

enumerator kStatus_LPI2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError
Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost
Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout
SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress
Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail
DMA request failed.

enumerator kStatus_LPI2C_Timeout
Timeout polling status flags.

IRQn_Type const kLpi2cMasterIrqs[]
Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

IRQn_Type const kLpi2cSlaveIrqs[]

lpi2c_master_isr_t s_lpi2cMasterIsr
Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

void *s_lpi2cMasterHandle[]
Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

uint32_t LPI2C_GetInstance(LPI2C_Type *base)
Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- base – The LPI2C peripheral base address.

Returns
LPI2C instance number starting from 0.

I2C_RETRY_TIMES
Retry times for waiting flag.

2.43 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;
```

(continues on next page)

(continued from previous page)

```
masterConfig->pinLowTimeout_ns      = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable    = false;
masterConfig->hostRequest.source    = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity  = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to lpi2c_master_config_t.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

Convert provided flags to status code, and clear any errors if present.

Parameters

- base – The LPI2C peripheral base address.
- status – Current status flags value that will be checked.

Return values

- kStatus_Success –
- kStatus_LPI2C_PinLowTimeout –

- kStatus_LPI2C_ArbitrationLost –
- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag

- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags().

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.

- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from

other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].

- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_LPI2C_Nak.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.

- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag
Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag
Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag
End Packet flag

enumerator kLPI2C_MasterStopDetectFlag
Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag
NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag
Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag
FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag
Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag
Data match flag

enumerator kLPI2C_MasterBusyFlag
Master busy flag

enumerator kLPI2C_MasterBusBusyFlag
Bus busy flag

enumerator kLPI2C_MasterClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
Errors to check for.

enum _lpi2c_direction
Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write
Master transmit.

enumerator kLPI2C_Read
Master receive.

enum _lpi2c_master_pin_config
LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain
LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Values:

enumerator kLPI2C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag
Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t
Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t
LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t
LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t
LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t

LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t

LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

LPI2C master descriptor of the transfer.

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

LPI2C master handle of the transfer.

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base
The LPI2C peripheral base address.

Param handle
Pointer to the LPI2C master driver handle.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct __lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable
: Enable transfers to continue when halted in debug mode.

bool ignoreAck
: Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig
: The pin configuration option.

uint32_t baudRate_Hz
: Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns
: Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns
: Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest
: Host request options.

struct __lpi2c_match_config
: *#include <fsl_lpi2c.h>* LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
: Data match configuration setting.

bool rxDataMatchOnly
: When set to true, received data is ignored until a successful match.

uint32_t match0
: Match value 0.

uint32_t match1
: Match value 1.

struct __lpi2c_master_transfer
: *#include <fsl_lpi2c.h>* Non-blocking transfer descriptor structure.

  This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
: Bit mask of options for the transfer. See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefaultFlag for normal transfers.

uint16_t slaveAddress
: The 7-bit slave address.

lpi2c_direction_t direction
: Either kLPI2C_Read or kLPI2C_Write.

uint32_t subaddress
: Sub address. Transferred MSB first.

size_t subaddressSize
: Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
: Pointer to data to transfer.

size_t dataSize
: Number of bytes to transfer.

struct __lpi2c_master_handle
: *#include <fsl_lpi2c.h>* Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
: Transfer state machine current state.

uint16_t remainingBytes
: Remaining byte count in current state.

uint8_t *buf
: Buffer pointer for current state.

uint16_t commandBuffer[6]
: LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer
: Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback
: Callback function pointer.

void *userData
: Application data passed to callback.

struct hostRequest

Public Members

bool enable
: Enable host request.

lpi2c_host_request_source_t source
: Host request source.

lpi2c_host_request_polarity_t polarity
: Host request pin polarity.

2.44 LPI2C Master DMA Driver

void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle, *lpi2c_master_edma_transfer_callback_t* callback, void *userData)

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –

- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

typedef struct *_lpi2c_master_edma_handle* lpi2c_master_edma_handle_t

LPI2C master EDMA handle of the transfer.

typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *status_t* completionStatus, void *userData)

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Param base
The LPI2C peripheral base address.

Param handle
Handle associated with the completed transfer.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

struct _lpi2c_master_edma_handle

#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base

LPI2C base pointer.

bool isBusy

Transfer state machine current state.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[20]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

edma_handle_t *rx

Handle for receive DMA channel.

edma_handle_t *tx

Handle for transmit DMA channel.

edma_tcd_t tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.45 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave              = true;
slaveConfig->address0                 = 0U;
slaveConfig->address1                 = 0U;
slaveConfig->addressMatchMode         = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable         = true;
slaveConfig->filterEnable             = true;
slaveConfig->enableGeneralCall        = false;
slaveConfig->sclStall.enableAck       = false;
slaveConfig->sclStall.enableTx        = true;
slaveConfig->sclStall.enableRx        = true;
slaveConfig->sclStall.enableAddress   = true;
slaveConfig->ignoreAck                = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns  = 0;
slaveConfig->sclGlitchFilterWidth_ns  = 0;
slaveConfig->dataValidDelay_ns        = 0;
slaveConfig->clockHoldTime_ns         = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t.

void LPI2C_SlaveInit(LPI2C_Type *base, const *lpi2c_slave_config_t* *slaveConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use LPI2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_slave_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_slave_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags().

static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the kLPI2C_SlaveTransmitAckFlag is asserted. This only happens if you enable the sclStall.enableAck field of the lpi2c_slave_config_t configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.
- ackOrNack – Pass true for an ACK or false for a NAK.

static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL,false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns
Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns
Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of lpi2c_slave_transfer_event_t enumerators for the events you wish to receive. The kLPI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kLPI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together lpi2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kLPI2C_SlaveAllEvents to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_LPI2C_Busy – Slave transfers have already been started on this handle.

status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag
Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag
Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag
Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag
Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag
Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag
Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag
Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag
FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag
Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag
Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag
General call flag

enumerator kLPI2C_SlaveBusyFlag
Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag
Bus busy flag

enumerator kLPI2C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags
Errors to check for.

enum _lpi2c_slave_address_match
LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0
Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1
Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1
Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent
Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent
Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent
Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent
Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C_SlaveRepeatedStartEvent
A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent
A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents
Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t
LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t
Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

LPI2C slave handle structure.

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the LPI2C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct __lpi2c_slave_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Enable slave mode.

uint8_t address0

Slave's 7-bit address.

uint8_t address1

Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode

Address matching options.

bool filterDozeEnable

Enable digital glitch filter in doze mode.

bool filterEnable
: Enable digital glitch filter.

bool enableGeneralCall
: Enable general call address matching.

struct *_lpi2c_slave_config* sclStall
: SCL stall enable options.

bool ignoreAck
: Continue transfers after a NACK is detected.

bool enableReceivedAddressRead
: Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns
: Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns
: Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns
: Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns
: Width in nanoseconds of the clock hold time.

struct __lpi2c_slave_transfer
: *#include <fsl_lpi2c.h>* LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event
: Reason the callback is being invoked.

uint8_t receivedAddress
: Matching address send by master.

uint8_t *data
: Transfer buffer

size_t dataSize
: Transfer size

status_t completionStatus
: Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount
: Number of bytes actually transferred since start or last repeated start.

struct __lpi2c_slave_handle
: *#include <fsl_lpi2c.h>* LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer
: LPI2C slave transfer copy.

bool isBusy
: Whether transfer is busy.

bool wasTransmit
: Whether the last transfer was a transmit.

uint32_t eventMask
: Mask of enabled events.

uint32_t transferredCount
: Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback
: Callback function called at transfer event.

void *userData
: Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck
: Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

bool enableTx
: Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx
: Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress
: Enables SCL clock stretching when the address valid flag is asserted.

2.46 LPIT: Low-Power Interrupt Timer

void LPIT_Init(LPIT_Type *base, const *lpit_config_t* *config)
: Ungates the LPIT clock and configures the peripheral for a basic operation.

  This function issues a software reset to reset all channels and registers except the Module Control register.

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

  Parameters

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

void LPIT_Deinit(LPIT_Type *base)

Disables the module and gates the LPIT clock.

Parameters

- base – LPIT peripheral base address.

void LPIT_GetDefaultConfig(*lpit_config_t* *config)

Fills in the LPIT configuration structure with default settings.

The default values are:

```
config->enableRunInDebug = false;
config->enableRunInDoze = false;
```

Parameters

- config – Pointer to the user configuration structure.

status_t LPIT_SetupChannel(LPIT_Type *base, *lpit_chnl_t* channel, const *lpit_chnl_params_t* *chnlSetup)

Sets up an LPIT channel based on the user's preference.

This function sets up the operation mode to one of the options available in the enumeration lpit_timer_modes_t. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

- base – LPIT peripheral base address.
- channel – Channel that is being configured.
- chnlSetup – Configuration parameters.

static inline void LPIT_EnableInterrupts(LPIT_Type *base, uint32_t mask)

Enables the selected PIT interrupts.

Parameters

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

static inline void LPIT_DisableInterrupts(LPIT_Type *base, uint32_t mask)

Disables the selected PIT interrupts.

Parameters

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

static inline uint32_t LPIT_GetEnabledInterrupts(LPIT_Type *base)

Gets the enabled LPIT interrupts.

Parameters

- base – LPIT peripheral base address.

Returns
The enabled interrupts. This is the logical OR of members of the enumeration lpit_interrupt_enable_t

static inline uint32_t LPIT_GetStatusFlags(LPIT_Type *base)

Gets the LPIT status flags.

Parameters

- base – LPIT peripheral base address.

Returns
The status flags. This is the logical OR of members of the enumeration lpit_status_flags_t

static inline void LPIT_ClearStatusFlags(LPIT_Type *base, uint32_t mask)

Clears the LPIT status flags.

Parameters

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

static inline void LPIT_SetTimerPeriod(LPIT_Type *base, *lpit_chnl_t* channel, uint32_t ticks)

Sets the timer period in units of count.

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.
- ticks – Timer period in units of ticks.

static inline void LPIT_SetTimerValue(LPIT_Type *base, *lpit_chnl_t* channel, uint32_t ticks)

Sets the timer period in units of count.

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

Note: Set TVAL register to 0 or 1 is invalid in compare mode.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.
- ticks – Timer period in units of ticks.

static inline uint32_t LPIT_GetCurrentTimerCount(LPIT_Type *base, *lpit_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 microseconds or milliseconds.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

Returns

Current timer counting value in ticks.

static inline void LPIT_StartTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Starts the timer counting.

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

static inline void LPIT_StopTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Stops the timer counting.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

FSL_LPIT_DRIVER_VERSION

Version 2.1.3

enum _lpit_chnl

List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

Values:

enumerator kLPIT_Chnl_0

LPIT channel number 0

enumerator kLPIT_Chnl_1

LPIT channel number 1

enumerator kLPIT_Chnl_2

LPIT channel number 2

enumerator kLPIT_Chnl_3

LPIT channel number 3

enum _lpit_timer_modes

Mode options available for the LPIT timer.

Values:

enumerator kLPIT_PeriodicCounter
Use the all 32-bits, counter loads and decrements to zero

enumerator kLPIT_DualPeriodicCounter
Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement

enumerator kLPIT_TriggerAccumulator
Counter loads on first trigger and decrements on each trigger

enumerator kLPIT_InputCapture
Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

enum _lpit_trigger_select
Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Values:

enumerator kLPIT_Trigger_TimerChn0
Channel 0 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn1
Channel 1 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn2
Channel 2 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn3
Channel 3 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn4
Channel 4 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn5
Channel 5 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn6
Channel 6 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn7
Channel 7 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn8
Channel 8 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn9
Channel 9 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn10
Channel 10 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn11
Channel 11 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn12
Channel 12 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn13
Channel 13 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn14
Channel 14 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn15
Channel 15 is selected as a trigger source

enum _lpit_trigger_source
Trigger source options available.

Values:

enumerator kLPIT_TriggerSource_External
Use external trigger input

enumerator kLPIT_TriggerSource_Internal
Use internal trigger

enum _lpit_interrupt_enable
List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerInterruptEnable
Channel 0 Timer interrupt

enumerator kLPIT_Channel1TimerInterruptEnable
Channel 1 Timer interrupt

enumerator kLPIT_Channel2TimerInterruptEnable
Channel 2 Timer interrupt

enumerator kLPIT_Channel3TimerInterruptEnable
Channel 3 Timer interrupt

enum _lpit_status_flags
List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerFlag
Channel 0 Timer interrupt flag

enumerator kLPIT_Channel1TimerFlag
Channel 1 Timer interrupt flag

enumerator kLPIT_Channel2TimerFlag
Channel 2 Timer interrupt flag

enumerator kLPIT_Channel3TimerFlag
Channel 3 Timer interrupt flag

typedef enum *_lpit_chnl* lpit_chnl_t
List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

typedef enum *_lpit_timer_modes* lpit_timer_modes_t

Mode options available for the LPIT timer.

typedef enum *_lpit_trigger_select* lpit_trigger_select_t

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

typedef enum *_lpit_trigger_source* lpit_trigger_source_t

Trigger source options available.

typedef enum *_lpit_interrupt_enable* lpit_interrupt_enable_t

List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

typedef enum *_lpit_status_flags* lpit_status_flags_t

List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

typedef struct *_lpit_chnl_params* lpit_chnl_params_t

Structure to configure the channel timer.

typedef struct *_lpit_config* lpit_config_t

LPIT configuration structure.

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

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

static void LPIT_ResetStateDelay(void)

Short wait for LPIT state reset.

After clear or set LPIT_EN, there should be delay longer than 4 LPIT functional clock.

static inline void LPIT_Reset(LPIT_Type *base)

Performs a software reset on the LPIT module.

This resets all channels and registers except the Module Control Register.

Parameters

- base – LPIT peripheral base address.

LPIT_RESET_STATE_DELAY

Delay used in LPIT_Reset.

The macro value should be larger than 4 * core clock / LPIT peripheral clock.

struct _lpit_chnl_params

#include <fsl_lpit.h> Structure to configure the channel timer.

Public Members

bool chainChannel

true: Timer chained to previous timer; false: Timer not chained

lpit_timer_modes_t timerMode
Timers mode of operation.

lpit_trigger_select_t triggerSelect
Trigger selection for the timer

lpit_trigger_source_t triggerSource
Decides if we use external or internal trigger.

bool enableReloadOnTrigger
true: Timer reloads when a trigger is detected; false: No effect

bool enableStopOnTimeout
true: Timer will stop after timeout; false: does not stop after timeout

bool enableStartOnTrigger
true: Timer starts when a trigger is detected; false: decrement immediately

struct __lpit__config
#include <fsl_lpit.h> LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the LPIT_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 enableRunInDebug
true: Timers run in debug mode; false: Timers stop in debug mode

bool enableRunInDoze
true: Timers run in doze mode; false: Timers stop in doze mode

2.47 LPSPI: Low Power Serial Peripheral Interface

2.48 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)
Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure lpspi_master_config_t.
- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)
Sets the lpspi_master_config_t structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to lpspi_master_config_t structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi_slave_config_t.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the lpspi_slave_config_t structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to lpspi_slave_config_t structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type _lpspi_flags.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

bool LPSPI_CheckTransferArgument(LPSPI_Type *base, *lpspi_transfer_t* *transfer, bool isEdma)

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, *lpspi_master_slave_mode_t* mode)

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, *lpspi_which_pcs_t* select)

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI_SetPCSContinous(LPSPI_Type *base, bool IsContinous)

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, *lpspi_delay_type_t* whichDelay)

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).

- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.

uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, *lpspi_delay_type_t* whichDelay, uint32_t srcClock_Hz)

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler).

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nano-seconds.
- whichDelay – The desired delay to configuration, which must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.
- data – The data word to be sent.

static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The data read from the data buffer.

void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_master_transfer_callback_t* callback, void *userData)

Initializes the LPSPI master handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_master_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns
true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag

Data Match flag

enumerator kLPSPI_ModuleBusyFlag

Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable
LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable
Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0
PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity
LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity
LPSPI clock polarity configuration.

Values:

enumerator kLPSPI_ClockPolarityActiveHigh
CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow
CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase
LPSPI clock phase configuration.

Values:

enumerator kLPSPI_ClockPhaseFirstEdge
CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge
CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction

LPSPI data shifter direction options.

Values:

enumerator kLPSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kLPSPI_LsbFirst

Data transfers start with least significant bit.

enum _lpspi_host_request_select

LPSPI Host Request select configuration.

Values:

enumerator kLPSPI_HostReqExtPin

Host Request is an ext pin.

enumerator kLPSPI_HostReqInternalTrigger

Host Request is an internal trigger.

enum _lpspi_match_config

LPSPI Match configuration options.

Values:

enumerator kLPSI_MatchDisabled

LPSPI Match Disabled.

enumerator kLPSI_1stWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordEqualsM0and2ndWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0andNxtWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordAndM1EqualsM0andM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1

LPSPI Match Enabled.

enum _lpspi_pin_config

LPSPI pin (SDO and SDI) configuration.

Values:

enumerator kLPSPI_SdiInSdoOut

LPSPI SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut

LPSPI SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut

LPSPI SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut

LPSPI SDO input, SDI output.

enum _lpspi_data_out_config

LPSPI data output configuration.

Values:

enumerator kLpspiDataOutRetained

Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate

Data out is tristated when chip select is de-asserted

enum _lpspi_transfer_width

LPSPI transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer

1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer

2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer

4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type

LPSPI delay type selection.

Values:

enumerator kLPSPI_PcsToSck

PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs

Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer

Delay between transfers.

enum _lpspi_transfer_config_flag_for_master

Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPI slave transfer configFlags.

Values:

enumerator kLPSPI_SlavePcs0

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS0 signal

enumerator kLPSPI_SlavePcs1

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS1 signal

enumerator kLPSPI_SlavePcs2

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS2 signal

enumerator kLPSPI_SlavePcs3

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS3 signal

enumerator kLPSPI_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

enum _lpspi_transfer_state

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator kLPSPI_Idle

Nothing in the transmitter/receiver.

enumerator kLPSPI_Busy

Transfer queue is not finished.

enumerator kLPSPI_Error

Transfer error.

typedef enum *_lpspi_master_slave_mode* lpspi_master_slave_mode_t

LPSPI master or slave mode configuration.

typedef enum *_lpspi_which_pcs_config* lpspi_which_pcs_t

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum *_lpspi_pcs_polarity_config* lpspi_pcs_polarity_config_t

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *_lpspi_clock_polarity* lpspi_clock_polarity_t
LPSPI clock polarity configuration.

typedef enum *_lpspi_clock_phase* lpspi_clock_phase_t
LPSPI clock phase configuration.

typedef enum *_lpspi_shift_direction* lpspi_shift_direction_t
LPSPI data shifter direction options.

typedef enum *_lpspi_host_request_select* lpspi_host_request_select_t
LPSPI Host Request select configuration.

typedef enum *_lpspi_match_config* lpspi_match_config_t
LPSPI Match configuration options.

typedef enum *_lpspi_pin_config* lpspi_pin_config_t
LPSPI pin (SDO and SDI) configuration.

typedef enum *_lpspi_data_out_config* lpspi_data_out_config_t
LPSPI data output configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t
LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t
LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t
LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t
LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t
Forward declaration of the _lpspi_master_handle typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t
Forward declaration of the _lpspi_slave_handle typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)
Master completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)
Slave completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_lpspi_transfer* lpspi_transfer_t
LPSPI master/slave transfer structure.

volatile uint8_t g_lpspiDummyData[]
Global variable for dummy data value setting.

LPSPI_DUMMY_DATA
LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

SPI_RETRY_TIMES
Retry times for waiting flag.

LPSPI_MASTER_PCS_SHIFT
LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_PCS_MASK
LPSPI master PCS shift macro , internal used.

LPSPI_SLAVE_PCS_SHIFT
LPSPI slave PCS shift macro , internal used.

LPSPI_SLAVE_PCS_MASK
LPSPI slave PCS shift macro , internal used.

struct _lpspi_master_config
#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate
Baud Rate for LPSPI.

uint32_t bitsPerFrame
Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
Clock polarity.

lpspi_clock_phase_t cpha
Clock phase.

lpspi_shift_direction_t direction
MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec
PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec
Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec
: After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs
: Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
: Desired PCS active high or low

lpspi_pin_config_t pinCfg
: Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
: Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay
: Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi_slave_config
: *#include <fsl_lpspi.h>* LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame
: Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
: Clock polarity.

lpspi_clock_phase_t cpha
: Clock phase.

lpspi_shift_direction_t direction
: MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs
: Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
: Desired PCS active high or low

lpspi_pin_config_t pinCfg
: Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
: Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct __lpspi_transfer
: *#include <fsl_lpspi.h>* LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData
: Send buffer.

uint8_t *rxData
: Receive buffer.

volatile size_t dataSize
: Transfer bytes.

uint32_t configFlags
: Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

struct __lpspi_master_handle
#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
: Is PCS continuous in transfer.

volatile bool writeTcrInIsr
: A flag that whether should write TCR in ISR.

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile bool isTxMask
: A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame
: Number of bytes in each frame

volatile uint16_t frameSize
: Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

uint32_t txBuffIfNull
: Used if the txData is NULL.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

struct __lpspi_slave_handle
: *#include <fsl_lpspi.h>* LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
: Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

2.49 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION
LPSPI EDMA driver version.

DMA_MAX_TRANSFER_COUNT
DMA max transfer size.

typedef struct *_lpspi_master_edma_handle* lpspi_master_edma_handle_t
Forward declaration of the _lpspi_master_edma_handle typedefs.

typedef struct *_lpspi_slave_edma_handle* lpspi_slave_edma_handle_t
Forward declaration of the _lpspi_slave_edma_handle typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)
Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)
Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)
Initializes the LPSPI master eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- configFlags – transfer configuration flags. _lpspi_transfer_config_flag_for_master.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

Returns
Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure, config field is not uesed.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.
- kStatus_InvalidArgument – The transfer structure is invalid.

Returns
Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, size_t *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_slave_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle)

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.

- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.

status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, size_t *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns

status of status_t.

struct __lpspi_master_edma_handle

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

volatile uint8_t bytesLastRead

Bytes for last read RDR.

volatile bool isThereExtraRxBytes

Is there extra RX byte.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

edma_tcd_t *lastTimeTCD
: Pointer to the lastTime TCD

bool isMultiDMATransmit
: Is there multi DMA transmit

volatile uint8_t dmaTransmitTime
: DMA Transfer times.

uint32_t lastTimeDataBytes
: DMA transmit last Time data Bytes

uint32_t dataBytesEveryTime
: Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

edma_transfer_config_t transferConfigRx
: Config of DMA rx channel.

edma_transfer_config_t transferConfigTx
: Config of DMA tx channel.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

uint32_t transmitCommand
: Used to write TCR for DMA purpose.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
: SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
: *#include <fsl_lpspi_edma.h>* LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

volatile uint8_t bytesLastRead
: Bytes for last read RDR.

volatile bool isThereExtraRxBytes
: Is there extra RX byte.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

volatile uint8_t state
: LPSPI transfer state.

uint32_t errorCount
: Error count for slave transfer.

lpspi_slave_edma_transfer_callback_t callback
: Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg

edma_tcd_t lpspiSoftwareTCD[2]
SoftwareTCD, internal used

2.50 LPTMR: Low-Power Timer

void LPTMR_Init(LPTMR_Type *base, const *lptmr_config_t* *config)
Ungates the LPTMR clock and configures the peripheral for a basic operation.

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

Parameters

- base – LPTMR peripheral base address
- config – A pointer to the LPTMR configuration structure.

void LPTMR_Deinit(LPTMR_Type *base)
Gates the LPTMR clock.

Parameters

- base – LPTMR peripheral base address

void LPTMR_GetDefaultConfig(*lptmr_config_t* *config)
Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

Parameters

- config – A pointer to the LPTMR configuration structure.

static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)
Enables the selected LPTMR interrupts.

Parameters

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

static inline void LPTMR_DisableInterrupts(LPTMR_Type *base, uint32_t mask)

Disables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration lptmr_interrupt_enable_t.

static inline uint32_t LPTMR_GetEnabledInterrupts(LPTMR_Type *base)

Gets the enabled LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address

Returns

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

static inline uint32_t LPTMR_GetStatusFlags(LPTMR_Type *base)

Gets the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address

Returns

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

static inline void LPTMR_ClearStatusFlags(LPTMR_Type *base, uint32_t mask)

Clears the LPTMR status flags.

Parameters

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

static inline void LPTMR_SetTimerPeriod(LPTMR_Type *base, uint32_t ticks)

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note:

a. The TCF flag is set with the CNR equals the count provided here and then increments.

b. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

- base – LPTMR peripheral base address
- ticks – A timer period in units of ticks

static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – LPTMR peripheral base address

Returns

The current counter value in ticks

static inline void LPTMR_StartTimer(LPTMR_Type *base)

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

- base – LPTMR peripheral base address

static inline void LPTMR_StopTimer(LPTMR_Type *base)

Stops the timer.

This function stops the timer and resets the timer's counter register.

Parameters

- base – LPTMR peripheral base address

FSL_LPTMR_DRIVER_VERSION

Driver Version

enum _lptmr_pin_select

LPTMR pin selection used in pulse counter mode.

Values:

enumerator kLPTMR_PinSelectInput_0

Pulse counter input 0 is selected

enumerator kLPTMR_PinSelectInput_1

Pulse counter input 1 is selected

enumerator kLPTMR_PinSelectInput_2

Pulse counter input 2 is selected

enumerator kLPTMR_PinSelectInput_3

Pulse counter input 3 is selected

enum _lptmr_pin_polarity

LPTMR pin polarity used in pulse counter mode.

Values:

enumerator kLPTMR_PinPolarityActiveHigh

Pulse Counter input source is active-high

enumerator kLPTMR_PinPolarityActiveLow

Pulse Counter input source is active-low

enum _lptmr_timer_mode

LPTMR timer mode selection.

Values:

enumerator kLPTMR_TimerModeTimeCounter

Time Counter mode

enumerator kLPTMR_TimerModePulseCounter
Pulse Counter mode

enum _lptmr_prescaler_glitch_value
LPTMR prescaler/glitch filter values.

Values:

enumerator kLPTMR_Prescale_Glitch_0
Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR_Prescale_Glitch_1
Prescaler divide 4, glitch filter 2

enumerator kLPTMR_Prescale_Glitch_2
Prescaler divide 8, glitch filter 4

enumerator kLPTMR_Prescale_Glitch_3
Prescaler divide 16, glitch filter 8

enumerator kLPTMR_Prescale_Glitch_4
Prescaler divide 32, glitch filter 16

enumerator kLPTMR_Prescale_Glitch_5
Prescaler divide 64, glitch filter 32

enumerator kLPTMR_Prescale_Glitch_6
Prescaler divide 128, glitch filter 64

enumerator kLPTMR_Prescale_Glitch_7
Prescaler divide 256, glitch filter 128

enumerator kLPTMR_Prescale_Glitch_8
Prescaler divide 512, glitch filter 256

enumerator kLPTMR_Prescale_Glitch_9
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR_Prescale_Glitch_10
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR_Prescale_Glitch_11
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR_Prescale_Glitch_12
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR_Prescale_Glitch_13
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR_Prescale_Glitch_14
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR_Prescale_Glitch_15
Prescaler divide 65536, glitch filter 32768

enum _lptmr_prescaler_clock_select
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

Values:

enum _lptmr_interrupt_enable

List of the LPTMR interrupts.

Values:

enumerator kLPTMR_TimerInterruptEnable

Timer interrupt enable

enum _lptmr_status_flags

List of the LPTMR status flags.

Values:

enumerator kLPTMR_TimerCompareFlag

Timer compare flag

typedef enum *_lptmr_pin_select* lptmr_pin_select_t

LPTMR pin selection used in pulse counter mode.

typedef enum *_lptmr_pin_polarity* lptmr_pin_polarity_t

LPTMR pin polarity used in pulse counter mode.

typedef enum *_lptmr_timer_mode* lptmr_timer_mode_t

LPTMR timer mode selection.

typedef enum *_lptmr_prescaler_glitch_value* lptmr_prescaler_glitch_value_t

LPTMR prescaler/glitch filter values.

typedef enum *_lptmr_prescaler_clock_select* lptmr_prescaler_clock_select_t

LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

---

typedef enum *_lptmr_interrupt_enable* lptmr_interrupt_enable_t

List of the LPTMR interrupts.

typedef enum *_lptmr_status_flags* lptmr_status_flags_t

List of the LPTMR status flags.

typedef struct *_lptmr_config* lptmr_config_t

LPTMR config structure.

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

The configuration struct can be made constant so it resides in flash.

static inline void LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)

Enable or disable timer DMA request.

Parameters

- base – base LPTMR peripheral base address
- enable – Switcher of timer DMA feature. “true” means to enable, “false” means to disable.

struct _lptmr_config

#include <fsl_lptmr.h> LPTMR config structure.

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

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode

Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler

True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource

LPTMR clock source

lptmr_prescaler_glitch_value_t value

Prescaler or glitch filter value

2.51 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.52 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;
```

(continues on next page)

(continued from previous page)

```
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

status_t LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

Return values

- kStatus_Success – Deinit is success.
- kStatus_LPUART_Timeout – Timeout during deinit.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, flase to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in lpuart_handle_t.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators _lpuart_flags. To check for a specific status, compare the return value with enumerators in the _lpuart_flags. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of _lpuart_interrupt_enable.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.

- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferStartRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART_TransferStopRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, *lpuart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.

- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_LPUART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.

- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_DriverIRQHandler(uint32_t instance)

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- instance – LPUART instance.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually
Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select
LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag
Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
LPUART transmit CTS configure.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *_lpuart_config* lpuart_config_t
LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_config
#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
: RX RTS enable

bool enableTxCTS
: TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
: TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
: TX CTS configure

uint8_t rtsWatermark
: RTS watermark

lpuart_idle_type_select_t rxIdleType
: RX IDLE type.

lpuart_idle_config_t rxIdleConfig
: RX IDLE configuration.

bool enableTx
: Enable TX

bool enableRx
: Enable RX

bool swapTxdRxd
: Swap TXD and RXD pins

struct __lpuart__transfer
: *#include <fsl_lpuart.h>* LPUART transfer structure.

Public Members

size_t dataSize
: The byte count to be transfer.

struct __lpuart__handle
: *#include <fsl_lpuart.h>* LPUART handle structure.

Public Members

volatile size_t txDataSize
: Size of the remaining data to send.

size_t txDataSizeAll
: Size of the data to send out.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t rxDataSizeAll
: Size of the data to receive.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
Callback function.

void *userData
LPUART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state.

bool isSevenDataBits
Seven data bits flag.

bool is16bitData
16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed72___

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 ___unnamed74___

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 ___unnamed76___

Public Members

uint8_t *volatile rxData
Address of remaining data to receive.

uint16_t *volatile rxData16
Address of remaining data to receive.

union ___unnamed78___

Public Members

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16

Start address of the receiver ring buffer.

2.53 LPUART eDMA Driver

void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

status_t LPUART_SendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- base – LPUART peripheral base address.
- lpuartEdmaHandle – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

typedef struct *_lpuart_edma_handle* lpuart_edma_handle_t

typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

struct __lpuart_edma_handle

#include <fsl_lpuart_edma.h> LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback

Callback function.

void *userData

LPUART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.54 MECC: internal error correction code

void MECC_Init(MECC_Type *base, *mecc_config_t* *config)

MECC module initialization function.

Parameters

- base – MECC base address.
- config – pointer to the MECC configuration structure.

void MECC_Deinit(MECC_Type *base)

Deinitializes the MECC.

Parameters

- base – MECC base address.

void MECC_GetDefaultConfig(*mecc_config_t* *config)

Sets the MECC configuration structure to default values.

Parameters

- config – pointer to the MECC configuration structure.

static inline uint32_t MECC_GetStatusFlags(MECC_Type *base)

Gets MECC status flags.

Parameters

- base – MECC peripheral base address.

Returns

MECC status flags.

static inline void MECC_ClearStatusFlags(MECC_Type *base, uint32_t mask)

MECC module clear interrupt status.

Parameters

- base – MECC base address.
- mask – status to clear.

static inline void MECC_EnableInterruptStatus(MECC_Type *base, uint32_t mask)

MECC module enable interrupt status.

Parameters

- base – MECC base address.
- mask – status to enable.

static inline void MECC_DisableInterruptStatus(MECC_Type *base, uint32_t mask)

MECC module disable interrupt status.

Parameters

- base – MECC base address.
- mask – status to disable.

static inline void MECC_EnableInterrupts(MECC_Type *base, uint32_t mask)

MECC module enable interrupt.

Parameters

- base – MECC base address.
- mask – The interrupts to enable.

static inline void MECC_DisableInterrupts(MECC_Type *base, uint32_t mask)

MECC module disable interrupt.

Parameters

- base – MECC base address.
- mask – The interrupts to disable.

status_t MECC_ErrorInjection(MECC_Type *base, uint32_t lowerrordata, uint32_t higherrordata, uint8_t eccdata, uint8_t banknumber)

MECC module error injection.

Bank0: ocram_base_address+0x20*i Bank1: ocram_base_address+0x20*i+0x8 Bank2: ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i = 0,1,2,3,4.....

Parameters

- base – MECC base address.
- lowerrordata – low 32 bits data.
- higherrordata – high 32 bits data.
- eccdata – ecc code.
- banknumber – ocram bank number.

Return values

kStatus_Success. –

status_t MECC_GetSingleErrorInfo(MECC_Type *base, *mecc_single_error_info_t* *info, uint8_t banknumber)

MECC module get single error information.

Bank0: ocram_base_address+0x20*i Bank1: ocram_base_address+0x20*i+0x8 Bank2: ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i = 0,1,2,3,4.....

Parameters

- base – MECC base address.
- info – single error information.
- banknumber – ocram bank number.

Return values

- kStatus_Success. –
- kStatus_MECC_BankMiss. –

status_t MECC_GetMultiErrorInfo(MECC_Type *base, *mecc_multi_error_info_t* *info, uint8_t banknumber)

MECC module get multiple error information.

Bank0: ocram_base_address+0x20*i Bank1: ocram_base_address+0x20*i+0x8 Bank2: ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i = 0,1,2,3,4.....

Parameters

- base – MECC base address.

- info – multiple error information.
- banknumber – ocram bank number.

Return values

- kStatus_Success. –
- kStatus_MECC_BankMiss. –

static inline uint32_t MECC_GetPendingFlags(MECC_Type *base)

Get pending flags for OCRAM wait and pipeline enable.

Parameters

- base – MECC base address.

Returns

Pending flags, should be the OR'ed value of mecc_pending_flag_t.

FSL_MECC_DRIVER_VERSION

Driver version 2.1.0.

Error codes for the MECC driver.

Values:

enumerator kStatus_MECC_BankMiss

Ocram bank miss

MECC interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the MECC interrupt configurations.

Values:

enumerator kMECC_SingleError0InterruptEnable

Single Bit Error On Ocram Bank0 interrupt enable.

enumerator kMECC_SingleError1InterruptEnable

Single Bit Error On Ocram Bank1 interrupt enable

enumerator kMECC_SingleError2InterruptEnable

Single Bit Error On Ocram Bank2 interrupt enable

enumerator kMECC_SingleError3InterruptEnable

Single Bit Error On Ocram Bank3 interrupt enable

enumerator kMECC_MultiError0InterruptEnable

Multiple Bits Error On Ocram Bank0 interrupt enable

enumerator kMECC_MultiError1InterruptEnable

Multiple Bits Error On Ocram Bank1 interrupt enable

enumerator kMECC_MultiError2InterruptEnable

Multiple Bits Error On Ocram Bank2 interrupt enable

enumerator kMECC_MultiError3InterruptEnable

Multiple Bits Error On Ocram Bank3 interrupt enable

enumerator kMECC_StrobeError0InterruptEnable

AXI Strobe Error On Ocram Bank0 interrupt enable

enumerator kMECC_StrobeError1InterruptEnable

AXI Strobe Error On Ocram Bank1 interrupt enable

enumerator kMECC_StrobeError2InterruptEnable
AXI Strobe Error On Ocram Bank2 interrupt enable

enumerator kMECC_StrobeError3InterruptEnable
AXI Strobe Error On Ocram Bank3 interrupt enable

enumerator kMECC_AccessError0InterruptEnable
Ocram Access Error On Bank0 interrupt enable

enumerator kMECC_AccessError1InterruptEnable
Ocram Access Error On Bank1 interrupt enable

enumerator kMECC_AccessError2InterruptEnable
Ocram Access Error On Bank2 interrupt enable

enumerator kMECC_AccessError3InterruptEnable
Ocram Access Error On Bank3 interrupt enable

enumerator kMECC_AllInterruptsEnable
all interrupts enable

MECC interrupt status configuration structure, default settings all disabled.

This structure contains the settings for all of the MECC interrupt status configurations.

Values:

enumerator kMECC_SingleError0InterruptStatusEnable
Single Bit Error On Ocram Bank0 interrupt status enable.

enumerator kMECC_SingleError1InterruptStatusEnable
Single Bit Error On Ocram Bank1 interrupt status enable

enumerator kMECC_SingleError2InterruptStatusEnable
Single Bit Error On Ocram Bank2 interrupt status enable

enumerator kMECC_SingleError3InterruptStatusEnable
Single Bit Error On Ocram Bank3 interrupt status enable

enumerator kMECC_MultiError0InterruptStatusEnable
Multiple Bits Error On Ocram Bank0 interrupt status enable

enumerator kMECC_MultiError1InterruptStatusEnable
Multiple Bits Error On Ocram Bank1 interrupt status enable

enumerator kMECC_MultiError2InterruptStatusEnable
Multiple Bits Error On Ocram Bank2 interrupt status enable

enumerator kMECC_MultiError3InterruptStatusEnable
Multiple Bits Error On Ocram Bank3 interrupt status enable

enumerator kMECC_StrobeError0InterruptStatusEnable
AXI Strobe Error On Ocram Bank0 interrupt status enable

enumerator kMECC_StrobeError1InterruptStatusEnable
AXI Strobe Error On Ocram Bank1 interrupt status enable

enumerator kMECC_StrobeError2InterruptStatusEnable
AXI Strobe Error On Ocram Bank2 interrupt status enable

enumerator kMECC_StrobeError3InterruptStatusEnable
AXI Strobe Error On Ocram Bank3 interrupt status enable

enumerator kMECC_AccessError0InterruptStatusEnable
Ocram Access Error On Bank0 interrupt status enable

enumerator kMECC_AccessError1InterruptStatusEnable
Ocram Access Error On Bank1 interrupt status enable

enumerator kMECC_AccessError2InterruptStatusEnable
Ocram Access Error On Bank2 interrupt status enable

enumerator kMECC_AccessError3InterruptStatusEnable
Ocram Access Error On Bank3 interrupt status enable

enumerator kMECC_AllInterruptsStatusEnable
all interrupts enable

MECC status flags.

This provides constants for the MECC status flags for use in the MECC functions.

Values:

enumerator kMECC_SingleError0InterruptFlag
Single Bit Error On Ocram Bank0 interrupt flag

enumerator kMECC_SingleError1InterruptFlag
Single Bit Error On Ocram Bank1 interrupt flag

enumerator kMECC_SingleError2InterruptFlag
Single Bit Error On Ocram Bank2 interrupt flag

enumerator kMECC_SingleError3InterruptFlag
Single Bit Error On Ocram Bank3 interrupt flag

enumerator kMECC_MultiError0InterruptFlag
Multiple Bits Error On Ocram Bank0 interrupt flag

enumerator kMECC_MultiError1InterruptFlag
Multiple Bits Error On Ocram Bank1 interrupt flag

enumerator kMECC_MultiError2InterruptFlag
Multiple Bits Error On Ocram Bank2 interrupt flag

enumerator kMECC_MultiError3InterruptFlag
Multiple Bits Error On Ocram Bank3 interrupt flag

enumerator kMECC_StrobeError0InterruptFlag
AXI Strobe Error On Ocram Bank0 interrupt flag

enumerator kMECC_StrobeError1InterruptFlag
AXI Strobe Error On Ocram Bank1 interrupt flag

enumerator kMECC_StrobeError2InterruptFlag
AXI Strobe Error On Ocram Bank2 interrupt flag

enumerator kMECC_StrobeError3InterruptFlag
AXI Strobe Error On Ocram Bank3 interrupt flag

enumerator kMECC_AccessError0InterruptFlag
Ocram Access Error On Bank0 interrupt flag

enumerator kMECC_AccessError1InterruptFlag
Ocram Access Error On Bank1 interrupt flag

enumerator kMECC_AccessError2InterruptFlag
Ocram Access Error On Bank2 interrupt flag

enumerator kMECC_AccessError3InterruptFlag
Ocram Access Error On Bank3 interrupt flag

enumerator kMECC_AllInterruptsFlag
all interrupts interrupt flag

MECC ocram bank number.

Values:

enumerator kMECC_OcramBank0
ocram bank number 0: ocram_base_address+0x20*i

enumerator kMECC_OcramBank1
ocram bank number 1: ocram_base_address+0x20*i+0x8

enumerator kMECC_OcramBank2
ocram bank number 2: ocram_base_address+0x20*i+0x10

enumerator kMECC_OcramBank3
ocram bank number 3: ocram_base_address+0x20*i+0x18

enum _mecc_pending_flag
Pending flags for OCRAM wait and pipeline enable. .

Values:

enumerator kMECC_ReadDataWaitPendingFlag
Indicate an update pending status for read data wait.

enumerator kMECC_ReadAddrPipelinePendingFlag
Indicate an update pending status for read address pipeline.

enumerator kMECC_WriteDataPipelinePendingFlag
Indicate an update pending status for write data pipeline.

enumerator kMECC_WriteAddrPipelinePendingFlag
Indicate an update pending status for write address pipeline.

enumerator kMECC_AllPendingFlags
Indicate all pending status flags.

typedef struct *_mecc_config* mecc_config_t
MECC user configuration.

Note: Ocram1StartAddress, Ocram1EndAddress, Ocram2StartAddress, Ocram2EndAddress are removed since 2.1.0 version; This changes will cause compile error for applications which use MECC driver before 2.1.0 version; To resolve compile error please use *startAddress* and *endAddress* as instead.

---

typedef struct *_mecc_single_error_info* mecc_single_error_info_t
MECC ocram single error information, including single error address, ECC code, error data and error bit position.

typedef struct *_mecc_multi_error_info* mecc_multi_error_info_t
MECC ocram multiple error information, including multiple error address, ECC code, error data.

struct __mecc__config

#include <fsl_mecc.h> MECC user configuration.

Note: Ocram1StartAddress, Ocram1EndAddress, Ocram2StartAddress, Ocram2EndAddress are removed since 2.1.0 version; This changes will cause compile error for applications which use MECC driver before 2.1.0 version; To resolve compile error please use *startAddress* and *endAddress* as instead.

Public Members

bool enableMecc

Enable the MECC function.

uint32_t startAddress

Start address of corresponding OCRAM memory region to enable ECC.

uint32_t endAddress

end address of corresponding OCRAM memory region to enable ECC.

bool enableReadDataWait

uint32_t Ocram1StartAddress; Ocram 1 start address, deprecated since 2.1.0

uint32_t Ocram1EndAddress; Ocram 1 end address, deprecated since 2.1.0

uint32_t Ocram2StartAddress; Ocram 2 start address, deprecated since 2.1.0.

uint32_t Ocram2EndAddress; Ocram 2 end address, deprecated since 2.1.0 If enabled, 1-cycle wait state will be inserted into each read access:

- **true** Enable read data wait;
- **false** Disable read data wait.

bool enableReadAddrPipeline

If enabled, the read address will be registered before can be applied to memory cell:

- **true** Enable Read address pipeline;
- **false** Disable Read address pipeline.

bool enableWriteDataPipeline

If enabled, the write data will be registered before can be applied to memory cell:

- **true** Enable write data pipeline;
- **false** Disable write data pipeline.

bool enableWriteAddrPipeline

If enabled, write address will be registered before can be applied to memory cell:

- **true** Enable write address pipeline;
- **false** Disable write address pipeline.

struct __mecc__single__error__info

#include <fsl_mecc.h> MECC ocram single error information, including single error address, ECC code, error data and error bit position.

Public Members

uint32_t singleErrorAddress

Single error address on Ocram bank n

uint32_t singleErrorDataLow
: Single error low 32 bits uncorrected read data on Ocram bank n

uint32_t singleErrorDataHigh
: Single error high 32 bits uncorrected read data on Ocram bank n

uint32_t singleErrorPosLow
: Single error bit postion of low 32 bits read data on Ocram bank n

uint32_t singleErrorPosHigh
: Single error bit postion of high 32 bits read data on Ocram bank n

uint8_t singleErrorEccCode
: Single error ECC code on Ocram bank n

struct __mecc_multi_error_info
: *#include <fsl_mecc.h>* MECC ocram multiple error information, including multiple error address, ECC code, error data.

Public Members

uint32_t multiErrorAddress
: Multiple error address on Ocram bank n

uint32_t multiErrorDataLow
: Multiple error low 32 bits read data on Ocram bank n

uint32_t multiErrorDataHigh
: Multiple error high 32 bits read data on Ocram bank n

uint8_t multiErrorEccCode
: Multiple error ECC code on Ocram bank n

2.55 MSGINTR: Message Unit

2.56 MSGINTR Driver

2.57 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)
{
    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)
{
}
```

(continues on next page)

(continued from previous page)

```
if (kMU_GeneralPurposeInterrupt3 & flags)
{
}
```

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)
{
```

(continues on next page)

(continued from previous page)

```
    Previous general purpose interrupt 0 or general purpose interrupt 2
    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.

2.58 NETC driver

Status code for the NETC module.

Values:

enumerator kStatus_NETC_RxFrameEmpty

Rx BD ring empty.

enumerator kStatus_NETC_RxTsrResp

Rx timestamp reference response

enumerator kStatus_NETC_RxFrameError

Rx frame error.

enumerator kStatus_NETC_TxFrameOverLen

Tx frame over length.

enumerator kStatus_NETC_LackOfResource

Lack of resources to configure certain features.

enumerator kStatus_NETC_Unsupported

Unsupported operation/feature.

enumerator kStatus_NETC_RxHRZeroFrame

Rx frame host reason is zero

enumerator kStatus_NETC_RxHRNotZeroFrame

Rx frame host reason is not zero

enumerator kStatus_NETC_NotFound
No entry found in hardware tables

enumerator kStatus_NETC_EntryExists
An entry already exists in hardware tables

enum _netc_ep_event
Defines the common interrupt event for callback use.

Values:

enumerator kNETC_EPRxEvent
EP Rx interrupt event.

enumerator kNETC_EPTxEvent
EP Tx interrupt event.

enum _netc_ep_tx_status
Status for the transmit buffer descriptor.

Values:

enumerator kNETC_EPTxSuccess
Success transmission.

enumerator kNETC_EPTxProgramErr
Error exists in either the Tx BD, the Tx ring registers, or both.

enumerator kNETC_EPTxTsdDrop
The time defined in TX_START expired before frame could be transmitted.

enumerator kNETC_EPTxFrameSizeErr
Frame size error.

enumerator kNETC_EPTxNullAddr
Null address.

enumerator kNETC_EPTxInvalidLength
Invalid frame/buffer/chain length.

enumerator kNETC_EPTxSrcMacSpoofingDetect
Source MAC address spoofing detected.

enumerator kNETC_EPTxPortRestDrop
Frame dropped due to port reset.

enumerator kNETC_EPTxPortDisableDrop
Frame dropped due to port disable.

enumerator kNETC_EPTxVlanTpidDrop
VLAN TPID not allowed.

enumerator kNETC_EPTxSmsoParamErr
Programming error in buffer descriptor used for direct switch enqueue.

enumerator kNETC_EPTxFrameGateErr
Frame too large for time gating window.

enumerator kNETC_EPTxAxiReadErr
AXI read error.

enumerator kNETC_EPTxAxiWriteErr
AXI write error.

enumerator kNETC_EPTxMultiBitECCErr
Frame not transmitted(dropped) due to a multi-bit ECC error detected.

enumerator kNETC_EPTxParityErr
Parity error.

enumerator kNETC_EPTxSwCongestion
Frame dropped due to switch congestion.

enum _netc_vlan_tpid_select
Ethernet VLAN Tag protocol identifier.

Values:

enumerator kNETC_StanCvlan
0x8100.

enumerator kNETC_StanSvlan
0x88A8.

enumerator kNETC_CustomVlan1
CVLANR1[ETYPE]

enumerator kNETC_CustomVlan2
CVLANR2[ETYPE]

enum _netc_packet_type
Ethernet packet type enumerator.

Values:

enumerator kNETC_PacketUnicast

enumerator kNETC_PacketMulticast

enumerator kNETC_PacketBroadcast

enum _netc_host_reason
Host reason.

Values:

enumerator kNETC_RegularFrame

enumerator kNETC_IngressMirror

enumerator kNETC_MACLearning

enumerator kNETC_TimestampResp

enumerator kNETC_SoftwareDefHR0

enumerator kNETC_SoftwareDefHR1

enumerator kNETC_SoftwareDefHR2

enumerator kNETC_SoftwareDefHR3

enumerator kNETC_SoftwareDefHR4

enumerator kNETC_SoftwareDefHR5

enumerator kNETC_SoftwareDefHR6

enumerator kNETC_SoftwareDefHR7

enum _netc_msix_vector_ctrl
MSIX vector control field.

Values:

enumerator kNETC_MsixIntrMaskBit
MSIX vector control interrupt mask bit.

enum _netc_tx_ext_flags
METC Extension Transmit Buffer Descriptor Extension flags field.

Values:

enumerator kNETC_TxExtVlanInsert
Enable VLAN insert.

enumerator kNETC_TxExtTwoStepTs
Enable two-step timestamp offload.

typedef enum *_netc_ep_event* netc_ep_event_t
Defines the common interrupt event for callback use.

typedef enum *_netc_ep_tx_status* netc_ep_tx_status_t
Status for the transmit buffer descriptor.

typedef struct *_netc_vlan* netc_vlan_t
VLAN tag struct.

typedef enum *_netc_vlan_tpid_select* netc_vlan_tpid_select_t
Ethernet VLAN Tag protocol identifier.

typedef enum *_netc_packet_type* netc_packet_type_t
Ethernet packet type enumerator.

typedef enum *_netc_host_reason* netc_host_reason_t
Host reason.

typedef struct *_ep_buffer_struct* netc_buffer_struct_t
Buffer structure. Driver can send/receive one frame spread across multiple buffers.

typedef struct *_ep_frame_struct* netc_frame_struct_t
Frame structure for single Tx/Rx frame.

typedef struct *_netc_frame_attr_struct* netc_frame_attr_t
Frame attribute struct.

typedef struct *_netc_tx_frame_info_struct* netc_tx_frame_info_t
Frame attribute structure.

typedef enum *_netc_msix_vector_ctrl* netc_msix_vector_ctrl_t
MSIX vector control field.

typedef struct *_netc_msix_entry* netc_msix_entry_t
NETC MSIX entry structure.

typedef enum *_netc_tx_ext_flags* netc_tx_ext_flags_t
METC Extension Transmit Buffer Descriptor Extension flags field.

FSL_NETC_DRIVER_VERSION
Driver Version.

NETC_ADDR_LOW_32BIT(x)
Macro to divides an address into a low 32 bits and a possible high 32 bits.

NETC_ADDR_HIGH_32BIT(x)

struct _netc_vlan

#include <fsl_netc.h> VLAN tag struct.

Public Members

uint32_t vid

Vlan Identifier.

uint32_t dei

Drop Eligible indicator.

uint32_t pcp

Priority.

uint32_t tpid

Tag protocol identifier.

struct _ep_buffer_struct

#include <fsl_netc.h> Buffer structure. Driver can send/receive one frame spread across multiple buffers.

Public Members

void *buffer

Buffer address.

uint16_t length

Buffer data length.

struct _ep_frame_struct

#include <fsl_netc.h> Frame structure for single Tx/Rx frame.

Public Members

netc_buffer_struct_t *buffArray

Buffer array. Tx: [in]App sets, Rx: [in/out]App sets prepared array, driver sets back received buffers array.

uint16_t length

Buffer array length. Tx: [in]App sets, Rx: [in/out]App sets prepared array length, driver sets back received buffers array length.

struct _netc_frame_attr_struct

#include <fsl_netc.h> Frame attribute struct.

Public Members

bool isTsAvail

Rx frame timestamp is available or not.

bool isVlanExtracted

Rx frame VLAN header is available or not.

uint32_t timestamp

The timestamp of this Rx frame.

struct _netc_tx_frame_info_struct
#include <fsl_netc.h> Frame attribute structure.

Public Members

bool isTsAvail
Tx frame timestamp is available or not.

uint32_t timestamp
The timestamp of this Tx frame, valid when isTsAvail is true.

void *context
Private context provided by the user.

netc_ep_tx_status_t status
Transmit status.

struct _netc_msix_entry
#include <fsl_netc.h> NETC MSIX entry structure.

Public Members

uint64_t msgAddr
Message address.

uint32_t msgData
Message data.

uint32_t control
Vector control, netc_msix_vector_ctrl_t.

2.59 Abbreviation in NETC driver

2.60 API layer

2.61 NETC Endpoint (EP) Driver

2.62 Endpoint (EP) Generic Configuration

typedef struct *_ep_handle* ep_handle_t
Endpoint handle.

typedef *status_t* (*ep_reclaim_cb_t)(*ep_handle_t* *handle, uint8_t ring, *netc_tx_frame_info_t* *frameInfo, void *userData)
Callback for reclaimed tx frames.

typedef void *(*ep_rx_alloc_cb_t)(*ep_handle_t* *handle, uint8_t ring, uint32_t length, void *userData)
Defines the EP Rx memory buffer alloc function pointer.

typedef void (*ep_rx_free_cb_t)(*ep_handle_t* *handle, uint8_t ring, void *address, void *userData)
Defines the EP Rx memory buffer free function pointer.

typedef *status_t* (*ep_get_link_status_cb_t)(*ep_handle_t* *handle, uint8_t *link)

Callback for getting link status.

typedef *status_t* (*ep_get_link_speed_cb_t)(*ep_handle_t* *handle, *netc_hw_mii_speed_t* *speed, *netc_hw_mii_duplex_t* *duplex)

Callback for getting link speed.

typedef *status_t* (*ep_preinit_vsi_cb_t)(*netc_enetc_hw_t* *hw, netc_hw_si_idx_t si)

Callback for vsi pre-init.

typedef struct *_ep_config* ep_config_t

Configuration for the endpoint handle.

typedef struct *_ep_config_const* ep_config_const_t

Configuration constant in handle.

status_t EP_Init(*ep_handle_t* *handle, uint8_t *macAddr, const *ep_config_t* *config, const *netc_bdr_config_t* *bdrConfig)

Initialize the endpoint with specified station interface.

Each station interface needs to call this API. In the case of a virtual station interface it's necessary that the physical station interface has been initialized beforehand.

Parameters

- handle –
- macAddr – Primary MAC address
- config – The user configuration
- bdrConfig – Array of buffer configurations (for each queue/ring)

Returns

status_t

status_t EP_Deinit(*ep_handle_t* *handle)

De-initialize the endpoint.

Parameters

- handle –

Returns

status_t

status_t EP_GetDefaultConfig(*ep_config_t* *config)

Get the default configuration.

Parameters

- config –

Returns

status_t

status_t EP_Up(*ep_handle_t* *handle, *netc_hw_mii_speed_t* speed, *netc_hw_mii_duplex_t* duplex)

Enable MAC transmission/reception To be called when the PHY link is up.

Parameters

- handle –
- speed –
- duplex –

Returns

status_t

status_t EP_Down(*ep_handle_t* *handle)

Disable MAC transmission/reception To be called when the PHY link is down.

Note: Must ensure all active Tx rings finish current transmission before call this API.

Parameters

- handle –

Returns

status_t

status_t EP_SetPrimaryMacAddr(*ep_handle_t* *handle, uint8_t *macAddr)

Set the Primary MAC address.

Parameters

- handle –
- macAddr –

Returns

status_t

static inline void EP_SetPortSpeed(*ep_handle_t* *handle, uint16_t pSpeed)

Set EP port speed.

Parameters

- handle –
- pSpeed –

struct _ep_config

#include <fsl_netc_endpoint.h> Configuration for the endpoint handle.

Public Members

netc_hw_si_idx_t si

Station interface index.

netc_hw_enetc_si_config_t siConfig

Station interface configuration.

uint8_t txPrioToTC[8]

Tx BD ring priority to Tx traffic class queue index mapping, range in TC0 ~ TC7.

netc_port_tx_tc_config_t txTcCfg[8]

Tx traffic class related configuration, vaild only on ENETC 0.

netc_ep_psfp_config_t psfpCfg

PSFP configuration,cover the ISI key construction profile and port ingress stream identification configuration.

bool enOuterAsInner

Enable use outer VLAN tag as the inner tag if only one tag is found.

netc_enetc_native_vlan_config_t rxOuterVLANCfg

Port outer native VLAN config.

netc_enetc_native_vlan_config_t rxInnerVLANCfg

Port inner native VLAN config.

netc_enetc_parser_config_t parserCfg
ENETC parser configuration.

uint32_t pauseOnThr
ENETC Port pause ON threshold value, value 0 means disables pause generation.

uint32_t pauseOffThr
ENETC Port pause OFF threshold value, value 0 means disables pause generation.

netc_msix_entry_t *msixEntry
MSIX table entry array.

uint8_t entryNum
MSIX entry number.

uint8_t cmdBdEntryIdx
MSIX entry index of command BD ring interrupt.

uint8_t siComEntryIdx
MSIX entry index of PSI-VSI communication interrupt.

uint8_t timerSyncEntryIdx
MSIX entry index of timer synchronous state change interrupt.

ep_reclaim_cb_t reclaimCallback
Callback for reclaimed Tx frames.

void *userData
User data, return in callback.

bool rxCacheMaintain
Enable/Disable Rx buffer cache maintain in driver.

bool txCacheMaintain
Enable/Disable Tx buffer cache maintain in driver.

bool rxZeroCopy
Enable/Disable zero-copy receive mode.

ep_rx_alloc_cb_t rxBuffAlloc
Callback function to alloc memory, must be provided for zero-copy Rx.

ep_rx_free_cb_t rxBuffFree
Callback function to free memory, must be provided for zero-copy Rx.

ep_preinit_vsi_cb_t preinitVsi
Callback function to pre-init VSI

netc_cmd_bdr_config_t cmdBdrConfig
Command BD ring configuration.

struct __ep__config__const
#include <fsl_netc_endpoint.h> Configuration constant in handle.

Public Members

netc_hw_si_idx_t si
Station interface index.

uint32_t rxRingUse
Number of Rx Rings to be used, when enable Rx ring group, this equal to the sum of all Rx group rings.

uint32_t txRingUse
: Number of Tx Rings to be used, note that when SI is Switch management ENETC SI, the number not include Tx ring 0.

uint32_t rxBdrGroupNum
: Rx BD ring group number, range in 0 ~ 2.

uint32_t ringPerBdrGroup
: The ring number in every Rx BD ring group, range in 1 ~ 8, active when rxBdrGroupNum not equal zero.

bool rxCacheMaintain
: Enable/Disable Rx buffer cache maintain in driver.

bool txCacheMaintain
: Enable/Disable Tx buffer cache maintain in driver.

bool rxZeroCopy
: Enable/Disable zero-copy receive mode.

uint8_t entryNum
: MSIX entry number.

ep_reclaim_cb_t reclaimCallback
: Callback for reclaimed Tx frames.

void *userData
: User data, return in callback.

ep_rx_alloc_cb_t rxBuffAlloc
: Callback function to alloc memory, must be provided for zero-copy Rx.

ep_rx_free_cb_t rxBuffFree
: Callback function to free memory, must be provided for zero-copy Rx.

struct __ep__handle
: *#include <fsl_netc_endpoint.h>* Handle for the endpoint Private internal data.

Public Members

netc_enetc_hw_t hw
: Hardware register map resource.

netc_enetc_cap_t capability
: ENETC capability.

ep_config_const_t cfg
: Endpoint configuration constant.

uint8_t ringShift
: Endpoint Tx ring shift.

netc_rx_bdr_t rxBdRing[1]
: Receive buffer descriptor ring.

netc_tx_bdr_t txBdRing[1]
: Transmit buffer descriptor ring.

netc_cmd_bdr_t cmdBdRing
: Command BD ring handle for endpoint.

uint8_t unicastHashCount[64]
: Unicast hash index collisions counter.

uint8_t multicastHashCount[64]
: Multicast hash index collisions counter.

uint8_t vlanHashCount[64]
: VLAN hash index collisions counter.

uint8_t macFilterCount[64]
: mac address filter index collisions counter.

uint8_t vlanFilterCount[64]
: vlan address filter index collisions counter.

ep_get_link_status_cb_t getLinkStatus
: Callback to get link status

ep_get_link_speed_cb_t getLinkSpeed
: Callback to get link speed

uint16_t vsiBitMapNotifyLinkStatus
: VSI bit map for link status notify

uint16_t vsiBitMapNotifyLinkSpeed
: VSI bit map for link speed notify

struct port

Public Members

netc_port_ethmac_t ethMac
: Ethernet MAC configuration.

netc_port_common_t common
: Port common configuration.

bool enableTg
: Enable port time gate scheduling.

bool enPseudoMacTxPad
: Enable pseudo MAC Port Transmit Padding, will pad the frame to a minimum of 60 bytes and append 4 octets of FCS.

2.63 Endpoint (EP) data path

typedef struct *_netc_ep_ipf_config* netc_ep_ipf_config_t
: Port Ingress Filter config.

typedef struct *_netc_ep_psfp_config* netc_ep_psfp_config_t
: PSFP config.

struct _netc_ep_ipf_config
: *#include <fsl_netc_endpoint.h>* Port Ingress Filter config.

Public Members

netc_ipf_config_t dosCfg
Configuration for L2/3 DOS.

netc_port_ipf_config_t portConfig
Configuration for port connected to enetc peripheral.

struct __netc_ep_psfp_config
#include <fsl_netc_endpoint.h> PSFP config.

2.64 Endpoint (EP) Interrupt Module

enum __ep_interrupt_flag
Interrupt enable/disable flags.

The value of the enumerator is not necessary match the bit in register. All interrupts in Endpoint are merged into this enum except the BDR specific interrupt. TODO SITMRIER

Values:

enumerator kNETC_EPPSIResetInterruptEnable

enumerator kNETC_EPPSIMsgRxInterruptEnable

typedef enum *_ep_interrupt_flag* ep_interrupt_flag_t
Interrupt enable/disable flags.

The value of the enumerator is not necessary match the bit in register. All interrupts in Endpoint are merged into this enum except the BDR specific interrupt. TODO SITMRIER

static inline void EP_CleanTxIntrFlags(*ep_handle_t* *handle, uint16_t txFrameIntrMask, uint16_t txThresIntrMask)
Clean the SI transmit interrupt flags.

Parameters

- handle – The EP handle.
- txFrameIntrMask – IPV value to be mapped, bit x represents ring x.
- txThresIntrMask – The Rx BD ring index to be mapped, bit x represents ring x.

static inline void EP_CleanRxIntrFlags(*ep_handle_t* *handle, uint32_t rxIntrMask)
Clean the SI receive interrupt flags.

Parameters

- handle – The EP handle.
- rxIntrMask – Rx interrupt bit mask, bit x represents ring x.

status_t EP_MsixSetGlobalMask(*ep_handle_t* *handle, bool mask)
Set the global MSIX mask status.

This function masks/unmasks global MSIX message. Mask - All of the vectors are masked, regardless of their per-entry mask bit states. Unmask - Each entry's mask status determines whether the vector is masked or not.

Parameters

- handle – The EP handle
- mask – The mask state. True: Mask, False: Unmask.

Returns
status_t

status_t EP_MsixSetEntryMask(*ep_handle_t* *handle, uint8_t entryIdx, bool mask)

Set the MSIX entry mask status for specified entry.

This function masks/unmasks MSIX message for specified entry.

Parameters

- handle – The EP handle
- entryIdx – The entry index in the table.
- mask – The mask state. True: Mask, False: Unmask.

Returns
status_t

status_t EP_MsixGetPendingStatus(*ep_handle_t* *handle, uint8_t pbaIdx, uint64_t *status)

Get the MSIX pending status in MSIX PBA table.

This function is to get the entry pending status from MSIX PBA table. If interrupt occurs but masked by vector control of entry, pending bit in PBA will be set.

Parameters

- handle – The EP handle
- pbaIdx – The index of PBA array with 64-bit unit.
- status – Pending status bit mask, bit n for entry n.

Returns
status_t

2.65 Endpoint (EP) Table Management Module

status_t EP_CmdBDRInit(*ep_handle_t* *handle, const *netc_cmd_bdr_config_t* *config)

Initialize endpoint command BD ring.

Parameters

- handle –
- config – The command BD ring configuration

Returns
status_t

status_t EP_CmdBDRDeinit(*ep_handle_t* *handle)

Deinit endpoint command BD ring.

Parameters

- handle –

Returns
status_t

2.66 Endpoint (EP) PSI/VSI

void EP_PsiEnableInterrupt(*ep_handle_t* *handle, uint32_t mask, bool enable)

PSI enables/disables specified interrupt.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, refer to netc_psi_msg_flags_t which should be OR'd together.
- enable – Enable/Disable the interrupt.

uint32_t EP_PsiGetStatus(*ep_handle_t* *handle)

PSI gets interrupt event flag status.

Parameters

- handle – The EP handle.

Returns

The interrupt mask, refer to netc_psi_msg_flags_t which should be OR'd together.

void EP_PsiClearStatus(*ep_handle_t* *handle, uint32_t mask)

PSI clears interrupt event flag.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, refer to netc_psi_msg_flags_t which should be OR'd together.

status_t EP_PsiSendMsg(*ep_handle_t* *handle, uint16_t msg, *netc_vsi_number_t* vsi)

PSI sends message to specified VSI(s)

Parameters

- handle – The EP handle.
- msg – The message to be sent.
- vsi – The VSI number.

Returns

status_t

bool EP_PsiCheckTxBusy(*ep_handle_t* *handle, *netc_vsi_number_t* vsi)

PSI checks Tx busy flag which should be cleaned when VSI receive the message data.

Parameters

- handle – The EP handle.
- vsi – The VSI number.

Returns

The busy status of specified VSI.

status_t EP_PsiSetRxBuffer(*ep_handle_t* *handle, *netc_vsi_number_t* vsi, uint64_t buffAddr)

PSI sets Rx buffer to receive message from specified VSI.

Note: The buffer memory size should be big enough for the message data from VSI

Parameters

- handle – The EP handle.

- vsi – The VSI number.
- buffAddr – The buffer address to store message data from VSI.

status_t EP_PsiGetRxMsg(*ep_handle_t* *handle, *netc_vsi_number_t* vsi, *netc_psi_rx_msg_t* *msgInfo)

PSI gets Rx message from specified VSI.

Parameters

- handle – The EP handle.
- vsi – The VSI number.
- msgInfo – The Rx message information.

void EP_VsiEnableInterrupt(*ep_handle_t* *handle, uint32_t mask, bool enable)

Enable VSI interrupt.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, see netc_vsi_msg_flags_t which should be OR'd together.
- enable – Enable/Disable interrupt.

uint32_t EP_VsiGetStatus(*ep_handle_t* *handle)

Get VSI interrupt status.

Parameters

- handle – The EP handle.

Returns

A bitmask composed of netc_vsi_msg_flags_t enumerators OR'd together.

void EP_VsiClearStatus(*ep_handle_t* *handle, uint32_t mask)

Clear VSI interrupt status.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, see netc_vsi_msg_flags_t which should be OR'd together.

status_t EP_VsiSendMsg(*ep_handle_t* *handle, uint64_t msgAddr, uint32_t msgLen)

VSI sends message to PSI.

Parameters

- handle – The EP handle.
- msgAddr – Address to store message ready to be sent, must be 64 bytes aligned.
- msgLen – The message length, must be 32 bytes aligned.

Returns

status_t

void EP_VsiCheckTxStatus(*ep_handle_t* *handle, *netc_vsi_msg_tx_status_t* *status)

Check VSI Tx status.

Parameters

- handle – The EP handle.
- status – The VSI Tx status structure.

status_t EP_VsiReceiveMsg(*ep_handle_t* *handle, uint16_t *msg)

VSI receives message from PSI.

Parameters

- handle – The EP handle.
- msg – The message from PSI.

Returns
status_t

2.67 Endpoint (EP) Ingress data path configuration

static inline *status_t* EP_RxParserConfig(*ep_handle_t* *handle, *netc_port_parser_config_t* *config)

Configure Parser in Receive Data Path.

Parameters

- handle –
- config –

Returns
status_t

static inline *status_t* EP_RxVlanCInit(*ep_handle_t* *handle, const *netc_vlan_classify_config_t* *config)

Configure the customer vlan type.

Parameters

- handle –
- config –

Returns
status_t

static inline *status_t* EP_RxVlanCConfigPort(*ep_handle_t* *handle, *netc_port_vlan_classify_config_t* *config)

Configure the Accepted Vlan.

Parameters

- handle –
- config –

Returns
status_t

status_t EP_RxIPFInit(*ep_handle_t* *handle, *netc_ep_ipf_config_t* *config)

Enable / Disable Ingress Port Filtering.

Applied for both Switch and ENETC

Parameters

- handle –
- config – IPF general features

Returns
status_t

static inline uint32_t EP_RxIPFGetTableRemainWordNum(*ep_handle_t* *handle)

Get remaining available word number (words size is 6 bytes) of the ingress Port Filter Table.

Note: This is a ternary match table, and the entries can vary in size, from 2 to 14 words.

Parameters

- handle –

Returns
uint32_t

status_t EP_RxIPFAddTableEntry(*ep_handle_t* *handle, *netc_tb_ipf_config_t* *config, uint32_t *entryID)

Add an entry for the ingress Port Filter Table.

This function do an add & query with return hardware id which can be used as future query / delete / update.

Parameters

- handle –
- config – IPF instance configuaration
- entryID – The table entry ID read out

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxIPFUpdateTableEntry(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_ipf_cfge_t* *cfg)

Update entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxIPFDelTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete an entry for the ingress Port Filter Table.

Parameters

- handle –
- entryID – The table entry ID

Returns
status_t

status_t EP_RxIPFResetMatchCounter(*ep_handle_t* *handle, uint32_t entryID)

Reset the counter of an ingress port filter entry.

Parameters

- handle –
- entryID – The table entry ID

Returns
status_t

status_t EP_RxIPFGetMatchedCount(*ep_handle_t* *handle, uint32_t entryID, uint64_t *count)

Get the matched count for entry in IPF.

Parameters

- handle –
- entryID – The table entry ID
- count – A count of how many times this entry has been matched.

Returns
status_t

static inline *status_t* EP_RxPSFPInit(*ep_handle_t* *handle, const *netc_ep_psfp_config_t* *config)

Init the ENETC PSFP, inlcude.

Parameters

- handle –
- config –

Returns
status_t

static inline uint32_t EP_RxPSFPGetISITableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of stream identification table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Filter, table.

Parameters

- handle –

Returns
uint32_t

status_t EP_RxPSFPAddISITableEntry(*ep_handle_t* *handle, *netc_tb_isi_config_t* *config, uint32_t *entryID)

Add an entry into the stream identification table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPDelISITableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete an entry in the stream identification table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetISTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of ingress stream table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

status_t EP_RxPSFPAddISTableEntry(*ep_handle_t* *handle, *netc_tb_is_config_t* *config)

Add an entry into the ingress stream table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPUpdateISTableEntry(*ep_handle_t* *handle, *netc_tb_is_config_t* *config)

Update an entry in the ingress stream table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPDelISTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete an entry in the stream identification table.

Parameters

- handle –
- entryID –

Returns
status_t

static inline uint32_t EP_RxPSFPGetISFTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of ingress stream filter table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification table.

Parameters

- handle –

Returns
uint32_t

status_t EP_RxPSFPAddISFTableEntry(*ep_handle_t* *handle, *netc_tb_isf_config_t* *config, uint32_t *entryID)

Add an entry into the ingress stream filter table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPUpdateISFTableEntry(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_isf_cfge_t* *cfg)

Update an entry into the ingress stream filter table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPDelISFTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Del an entry into the stream filter table.

Parameters

- handle –
- entryID –

Returns
status_t

static inline uint32_t EP_RxPSFPGetRPTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of Rate Policer table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

Parameters

- handle –

Returns
uint32_t

status_t EP_RxPSFPAddRPTableEntry(*ep_handle_t* *handle, *netc_tb_rp_config_t* *config)

Add entry to Rate Policer table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPUpdateRPTableEntry(*ep_handle_t* *handle, *netc_tb_rp_config_t* *config)

Update entry in Rate Policer table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPAddOrUpdateRPTableEntry(*ep_handle_t* *handle, *netc_tb_rp_config_t* *config)

Add or update entry in Rate Policer table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPDelRPTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete entry in the Rate Policer table.

Parameters

- handle –
- entryID –

Returns
status_t

status_t EP_RxPSFPGetRPStatistic(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_rp_stse_t* *statis)

Get statistic of specified Rate Policer entry.

Parameters

- handle –
- entryID –
- statis –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetISCTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of ingress stream count table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

status_t EP_RxPSFPAddISCTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Add entry in ingress stream count table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPGetISCStatistic(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_isc_stse_t* *statistic)

Get ingress stream count statistic.

Parameters

- handle –
- entryID –
- statistic –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPResetISCStatistic(*ep_handle_t* *handle, uint32_t entryID)

Reset the count of the ingress stream count.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetSGITableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of stream gate instance table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

status_t EP_RxPSFPAddSGITableEntry(*ep_handle_t* *handle, *netc_tb_sgi_config_t* *config)

Add entry in stream gate instance table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPUpdateSGITableEntry(*ep_handle_t* *handle, *netc_tb_sgi_config_t* *config)

Update entry in stream gate instance table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPDelSGITableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete entry in stream gate instance table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPGetSGIState(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_sgise_t* *state)

Get state of the stream gate instance for specified entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetSGCLTableRemainWordNum(*ep_handle_t* *handle)

Get remaining available words number of Stream Gate Control List table.

Note: This is a dynamic bounded index table, and number of words required for a stream gate control list is 1+N/2 where N is number of gate time slots in the stream gate control list. The remaining word should be greater than the want added entry size

---

Parameters

- handle –

Returns
uint32_t

status_t EP_RxPSFPAddSGCLTableEntry(*ep_handle_t* *handle, *netc_tb_sgcl_gcl_t* *config)

Add entry into Stream Gate Control List Table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPDelSGCLTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete entry of Stream Gate Control List Table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPGetSGCLGateList(*ep_handle_t* *handle, *netc_tb_sgcl_gcl_t* *gcl, uint32_t length)

Get Stream Gate Control List Table entry gate control list.

Parameters

- handle –
- gcl –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxPSFPGetSGCLState(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_sgcl_sgclse_t* *state)

Get state (ref count) for Stream Gate Control List table entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxL2MFInit(*ep_handle_t* *handle, *netc_si_l2mf_config_t* *config)

Init the L2 MAC Filter for a specified SI.

Parameters

- handle – EP handle
- config – The L2 MAC Filter configuration

Returns
status_t

status_t EP_RxL2MFAddHashEntry(*ep_handle_t* *handle, *netc_packet_type_t* type, uint8_t *macAddr)

Add entry into the MAC address hash filter with given MAC address Hardware layer will not maitain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle – EP handle
- type – Unicast or multicast MAC address
- macAddr – MAC address to be added in filter table

Returns
status_t

status_t EP_RxL2MFDelHashEntry(*ep_handle_t* *handle, *netc_packet_type_t* type, uint8_t *macAddr)

Delete entry into the MAC address hash filter with given MAC address Hardware layer will not maitain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle – EP handle
- type – Unicast or multicast MAC address
- macAddr – MAC address to be deleted from filter table

Returns
status_t

status_t EP_RxL2MFAddEMTableEntry(*ep_handle_t* *handle, uint32_t idx, uint8_t *macAddr)

Add entry into the MAC filter exact match table.

The entry is associated to the current Station Interface

Parameters

- handle –
- idx – Index in the entry table
- macAddr – MAC address for filter

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxL2MFDelEMTableEntry(*ep_handle_t* *handle, uint32_t idx)

Delete entry into the MAC filter exact match table.

Parameters

- handle – EP handle
- idx – Index in the entry table

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxL2VFInit(*ep_handle_t* *handle, *netc_si_l2vf_config_t* *config)

For VLAN filter, use inner vlan tag or outer vlan tag.

Parameters

- handle –
- config –

Returns
status_t

status_t EP_RxL2VFAddHashEntry(*ep_handle_t* *handle, uint16_t vlanId)

Add entry into the VLAN hash filter with given MAC address Hardware layer will not maitain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle –
- vlanId – VLAN identifier for filter

Returns
status_t

status_t EP_RxL2VFDelHashEntry(*ep_handle_t* *handle, uint16_t vlanId)

Delete entry into the VLAN hash filter with given MAC address Hardware layer will not maitain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle –
- vlanId – VLAN identifier for filter

Returns
status_t

status_t EP_RxL2VFAddEMTableEntry(*ep_handle_t* *handle, uint32_t idx, uint16_t vlanId, *netc_vlan_tpid_select_t* tpid)

Add entry into the MAC filter exact match table.

The entry is associated to the current Station Interface

Parameters

- handle –
- idx – Index in the entry table
- vlanId – VLAN identifier
- tpid – VLAN TPID

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_RxL2VFDelEMTableEntry(*ep_handle_t* *handle, uint32_t idx)

Delete entry into the VLAN filter exact match table.

Parameters

- handle –
- idx – Index in the entry table

Returns
status_t

Returns
See netc_cmd_error_t

static inline *status_t* EP_RxMapVlanToIpv(*ep_handle_t* *handle, *netc_vlan_t* vlan, uint8_t ipv)

Set the received Frame vlan to IPV mapping.

Parameters

- handle –
- vlan – Frame VLAN tag.
- ipv – The IPV value to be mapped.

Returns
status_t

static inline *status_t* EP_RxMapIpvToRing(*ep_handle_t* *handle, uint8_t ipv, uint8_t ring)

Set the IPV to Rx ring mapping.

Parameters

- handle –

- ipv – IPV value to be mapped.
- ring – The Rx BD ring index to be mapped.

Returns
status_t

static inline *status_t* EP_RxSetDefaultBDRGroup(*ep_handle_t* *handle, *netc_hw_enetc_si_rxr_group* groupIdx)

Set the default used receive Rx BD ring group.

Note: The IPV mapped ring index is the relative index inside the default used group.

Parameters

- handle –
- groupIdx – The default Rx group index.

Returns
status_t

2.68 Endpoint (EP) Statistic Module

enum _ep_flags

Status/interrupt detect flags merged to same set of enum. TODO SITMRIDR.

Values:

enumerator kNETC_EPTimerSyncedFlag

enumerator kNETC_EPICMBlockedFlag

enumerator kNETC_EPWakeOnLANActiveFlag

typedef enum *_ep_flags* ep_flags_t

Status/interrupt detect flags merged to same set of enum. TODO SITMRIDR.

static inline *status_t* EP_GetPortDiscardStatistic(*ep_handle_t* *handle, bool useTx, *netc_port_discard_statistic_t* *statistic)

Get the ENETC port discard statistic and reason.

Get the discarded count of frames and its reasons.

Parameters

- handle –
- useTx – true - Tx port. false - Rx port.
- statistic – pointer to the statistic data

Returns
status_t

static inline *status_t* EP_ClearPortDiscardReason(*ep_handle_t* *handle, bool useTx, uint32_t reason0, uint32_t reason1)

Clean the EP Port Rx discard reason. Set the related bits to 1 to clear the specific reasons.

Parameters

- handle –

- useTx – true - Tx port. false - Rx port.
- reason0 –
- reason1 –

Returns
status_t

static inline uint32_t EP_GetPortTGSListStatus(*ep_handle_t* *handle)

Get EP port time gate scheduling gate list status.

Parameters

- handle –

Returns
Port status flags which are ORed by the enumerators in the netc_port_tgsl_status_t

2.69 Endpoint (EP) Egress data path configuration

status_t EP_TxTGSConfigAdminGcl(*ep_handle_t* *handle, *netc_tb_tgs_gcl_t* *config)

Config the Time Gate Scheduling entry admin gate control list.

This function is used to program the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_TxPortTGSEnable(*ep_handle_t* *handle, bool enable, uint8_t gateState)

Enable the EP port time gate scheduling.

Parameters

- handle –
- enable –
- gateState –

Returns
status_t

status_t EP_TxtTGSGetOperGcl(*ep_handle_t* *handle, *netc_tb_tgs_gcl_t* *gcl, uint32_t length)

Get Time Gate Scheduling entry operation gate control list.

This function is used to read the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

Parameters

- handle –
- gcl –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

status_t EP_TxTrafficClassConfig(*ep_handle_t* *handle, *netc_hw_tc_idx_t* tcIdx, const *netc_port_tx_tc_config_t* *config)

Config the TC (traffic class) property.

Parameters

- handle –
- tcIdx –
- config –

Returns
status_t

static inline void EP_TxTcConfigPreemption(*ep_handle_t* *handle, *netc_hw_tc_idx_t* tcIdx, const bool enable)

Config Preemption for each port TC (traffic class)

Parameters

- handle –
- tcIdx –
- enable –

static inline void EP_TxGetTcPreemption(*ep_handle_t* *handle, *netc_hw_tc_idx_t* tcIdx, bool *enabled)

Get Preemption configuration for each port TC (traffic class)

Parameters

- handle –
- tcIdx –
- enabled –

static inline void EP_TxPortEthMacConfigPreemption(*ep_handle_t* *handle, const *netc_port_preemption_config* *config)

Configure Preemption control configuration for an ethernet MAC.

Parameters

- handle –
- config –

static inline void EP_TxPortGetEthMacPreemption(*ep_handle_t* *handle, *netc_port_preemption_config* *config, *netc_port_phy_mac_preemption_status_t* *status)

Get Preemption configuration from ethernet MAC port.

Parameters

- handle –
- config –
- status –

2.70 Endpoint (EP) Transmit/Receive

enum _ep_rx_flags

Values:

enumerator kEP_RX_RSS_VALID

Request timestamp.

enumerator kEP_RX_VLAN_VALID

Specifiy frame departure time.

enumerator kEP_RX_TIMESTAMP_VALID

Enable port masquerading.

enum _ep_tx_opt_flags

Values:

enumerator kEP_TX_OPT_REQ_TS

Request timestamp (IEEE 1588 PTP two-step timestamp).

enumerator kEP_TX_OPT_VLAN_INSERT

Enable VLAN insert.

enumerator kEP_TX_OPT_START_TIME

Specifiy frame departure time.

typedef enum *_ep_rx_flags* ep_rx_flags_t

typedef enum *_ep_tx_opt_flags* ep_tx_opt_flags

typedef struct *_ep_tx_offload* netc_tx_offload_t

typedef struct *_ep_tx_opt* ep_tx_opt

status_t EP_SendFrameCommon(*ep_handle_t* *handle, *netc_tx_bdr_t* *txBdRing, uint8_t hwRing, *netc_frame_struct_t* *frame, void *context, *netc_tx_bd_t* *txDesc, bool txCacheMaintain)

Common part for transfer regular frame or Switch management frame.

Note: This function is internal used. Please use EP_SendFrame() or SWT_SendFrame() API to send frames.

Parameters

- handle –
- txBdRing – The Transmit buffer descriptor ring handle
- hwRing – The hardware Tx ring index
- frame – The frame descriptor pointer
- context – Private context provided back by ep_reclaim_cb_t
- txDesc – Point to the Transmits BD Description array.
- txCacheMaintain – Enable/Disable Tx buffer Cache Maintain.

Return values

status_t –

status_t EP_SendFrame(*ep_handle_t* *handle, uint8_t ring, *netc_frame_struct_t* *frame, void *context, *ep_tx_opt* *opt)

Transmits a frame for specified ring. This API is zero-copy and requires the ep_reclaim_cb_t to be called to free the transmitted frame.

Parameters

- handle –
- ring – The ring index
- frame – The frame descriptor pointer
- context – Private context provided back by ep_reclaim_cb_t
- opt – Additional tx options. If NULL, default is tx timestamping enabled, no start time and no masquerading.

Return values

status_t –

static inline void EP_WaitUnitilTxComplete(*ep_handle_t* *handle, uint8_t ring)

Wait until the EP Tx ring has completed the transfer.

Note: Only call after EP_SendFrame() to do a no-interrupt transfer

---

Parameters

- handle –
- ring – The ring index

netc_tx_frame_info_t *EP_ReclaimTxDescCommon(*ep_handle_t* *handle, *netc_tx_bdr_t* *txBdRing, uint8_t hwRing, bool enCallback)

Common part of Reclaim tx descriptors for regular frame or Switch management frame.

Note: This function is internal used. Please use EP_ReclaimTxDescriptor() or SWT_ReclaimTxDescriptor() API to Reclaim tx descriptors.

Parameters

- handle –
- txBdRing – The Transmit buffer descriptor ring handle
- hwRing – The hardware Tx ring index
- enCallback – Enable/Disable call the Tx Reclaim callback functions.

void EP_ReclaimTxDescriptor(*ep_handle_t* *handle, uint8_t ring)

Reclaim tx descriptors. This function is used to update the tx descriptor status and get the tx timestamp. For each reclaimed transmit frame the ep_reclaim_cb_t is called.

This is called after being notified of a transmit completion from ISR. It runs until there are no more frames to be reclaimed in the BD ring.

Parameters

- handle –
- ring – The ring index

status_t EP_ReceiveFrameCommon(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint8_t ring, *netc_frame_struct_t* *frame, *netc_frame_attr_t* *attr, bool rxCacheMaintain)

Common part of receives one frame with zero copy from specified ring.

Note: This function is internal used. Please use EP_ReceiveFrame() or SWT_ReceiveFrame() API.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- ring – Ring index
- frame – Frame buffer point
- attr – Frame attribute pointer
- rxCacheMaintain – Enable/Disable Rx buffer Cache maintain

Returns
status_t

status_t EP_ReceiveFrame(*ep_handle_t* *handle, uint8_t ring, *netc_frame_struct_t* *frame, *netc_frame_attr_t* *attr)

Receives one frame with zero copy from specified ring.

Note: The sufficient rx frame data structure MUST be provided by appliction.

Parameters

- handle –
- ring – The ring index
- frame – The frame descriptor pointer
- attr – Frame attribute pointer

Returns
kStatus_Success Successfully receive a regular frame

Returns
kStatus_NETC_RxHRNotZeroFrame Frame in Rx BD ring is management frame, need call SWT_ReceiveFrame()

Returns
kStatus_NETC_RxTsrResp Frame in Rx BD ring is Transmit Timestamp Reference Response messages, need call SWT_GetTimestampRefResp() to get Transmit Timestamp Reference Response

Returns
kStatus_NETC_RxFrameEmpty Rx BD ring is empty

Returns
kStatus_NETC_RxFrameError Frame in Rx BD ring has error, need be dropped

Returns
kStatus_InvalidArgument Rx BD ring index is out of range

Returns
kStatus_NETC_LackOfResource Appliction provided buffer is not enough

void EP_DropFrame(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint8_t ring)

Drop one frame.

Note: This function is internal used.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- ring – Ring index

status_t EP_ReceiveFrameCopyCommon(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint8_t ring, void *buffer, uint32_t length, *netc_frame_attr_t* *attr, bool rxCacheMaintain)

Common part of receive regular frame or Switch management frame which will be copied in the provided buffer.

Note: This function is internal used. Please use EP_ReceiveFrameCopy() or SWT_ReceiveFrameCopy() API.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- ring – Ring index
- buffer – Buffer address
- length – Buffer length
- attr – Frame attribute pointer
- rxCacheMaintain – Enable/Disable Rx buffer Cache maintain

Returns

status_t

status_t EP_ReceiveFrameCopy(*ep_handle_t* *handle, uint8_t ring, void *buffer, uint32_t length, *netc_frame_attr_t* *attr)

Receives one frame which will be copied in the provided buffer from specified ring.

Note: The buffer size MUST be queried using EP_GetRxFrameSize() beforehand.

Parameters

- handle –
- ring – Ring index
- buffer – Buffer address
- length – Buffer length
- attr – Frame attribute pointer

Returns

kStatus_Success Successfully receive a regular frame

Returns
kStatus_InvalidArgument Rx BD ring index is out of range

status_t EP__GetRxFrameSizeCommon(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint32_t *length)

Common part of get pending frame size API for regular frame or Switch management frame.

Note: This function is internal used. Please use EP_GetRxFrameSize() or SWT_GetRxFrameSize() API.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- length – The length of the valid frame received.

Returns
status_t

status_t EP__GetRxFrameSize(*ep_handle_t* *handle, uint8_t ring, uint32_t *length)

Gets the size of the pending frame in the specified receive ring buffer.

Note: Frame size without FCS

Parameters

- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- ring – The ring index
- length – The length of the valid frame received.

Returns
kStatus_Success Successfully get the length of a regular frame

Returns
kStatus_NETC_RxHRNotZeroFrame Frame in Rx BD ring is management frame, need call SWT_GetRxFrameSize() to get frame size

Returns
kStatus_NETC_RxTsrResp Frame in Rx BD ring is Transmit Timestamp Reference Response messages, need call SWT_GetTimestampRefResp() to get Transmit Timestamp Reference Response

Returns
kStatus_NETC_RxFrameEmpty Rx BD ring is empty

Returns
kStatus_NETC_RxFrameError Frame in Rx BD ring has error, need be dropped

Returns
kStatus_InvalidArgument Rx BD ring index is out of range

struct __ep__tx__offload

#include <fsl_netc_endpoint.h>

Public Members

bool lso
: Large send offload.

bool l4Checksum
: L4 checksum offload.

bool ipv4Checksum
: IPv4 checksum offload.

uint32_t lsoMaxSegSize
: Large send offload maximum segment size.

uint32_t l4Type
: L4 type. 1-UDP, 2-TCP.

uint32_t l3Type
: L3 type. 0-IPv4, 1-IPv6.

uint32_t l3HeaderSize
: L3 IP header size in units of 32-bit words.

uint32_t l3Start
: Offset of the IPv4/IPv6 header in units of bytes.

struct __ep__tx__opt
: *#include <fsl_netc_endpoint.h>*

Public Members

uint32_t flags
: A bitmask of ep_tx_opt_flags

uint32_t timestamp
: Departure timestamp, used if kEP_TX_OPT_START_TIME is set

netc_enetc_vlan_tag_t vlan
: VLAN tag which will be inserted, used if kEP_TX_OPT_VLAN_INSERT is set

2.71 Hardware layer

enum __netc__hw__enetc__idx
: ENETC index enumerator.

Values:

enumerator kNETC_ENETC0
: ENETC hardware 0

enumerator kNETC_ENETC1
: ENETC hardware 0

enum __netc__hw__switch__idx
: SWITCH index enumerator.

Values:

enumerator kNETC_SWITCH0
: SWITCH hardware 0

enum _netc_hw_port_idx

Port Resource for the NETC module.

Values:

enumerator kNETC_ENETC0Port

MAC port for ENETC0

enumerator kNETC_ENETC1Port

Pseudo MAC port for ENETC1

enumerator kNETC_SWITCH0Port0

MAC port0 for SWITCH

enumerator kNETC_SWITCH0Port1

MAC port1 for SWITCH

enumerator kNETC_SWITCH0Port2

MAC port2 for SWITCH

enumerator kNETC_SWITCH0Port3

MAC port3 for SWITCH

enumerator kNETC_SWITCH0Port4

Pseudo port4 for SWITCH

enum _netc_hw_tc_idx

Traffic class enumerator.

Values:

enumerator kNETC_TxTC4

Traffic class 4

enumerator kNETC_TxTC5

Traffic class 5

enumerator kNETC_TxTC6

Traffic class 6

enumerator kNETC_TxTC7

Traffic class 7

enumerator kNETC_TxTC0

Traffic class 0

enumerator kNETC_TxTC1

Traffic class 1

enumerator kNETC_TxTC2

Traffic class 2

enumerator kNETC_TxTC3

Traffic class 3

enum _netc_hw_bdr_idx

Enumeration for the ENETC SI BDR identifier.

Values:

enumerator kNETC_BDR0

enumerator kNETC_BDR1

enumerator kNETC_BDR2

enumerator kNETC_BDR3

enumerator kNETC_BDR4

enumerator kNETC_BDR5

enumerator kNETC_BDR6

enumerator kNETC_BDR7

enumerator kNETC_BDR8

enumerator kNETC_BDR9

enumerator kNETC_BDR10

enumerator kNETC_BDR11

enumerator kNETC_BDR12

enumerator kNETC_BDR13

enum _netc_hw_swt_cbdr_idx

Switch command BD ring index enumerator.

Values:

enumerator kNETC_SWTCBDR0

Switch command BD ring 0

enumerator kNETC_SWTCBDR1

Switch command BD ring 1

enum _netc_hw_classs_queue_idx

Enumerator for ETM class queue identifier.

Values:

enumerator kNETC_ClassQueue0

ETM Class Queue 0

enumerator kNETC_ClassQueue1

ETM Class Queue 1

enumerator kNETC_ClassQueue2

ETM Class Queue 2

enumerator kNETC_ClassQueue3

ETM Class Queue 3

enumerator kNETC_ClassQueue4

ETM Class Queue 4

enumerator kNETC_ClassQueue5

ETM Class Queue 5

enumerator kNETC_ClassQueue6

ETM Class Queue 6

enumerator kNETC_ClassQueue7

ETM Class Queue 7

enum _netc_hw_congestion_group_idx

Enumerator for the ETM congestion group.

Values:

enumerator kNETC_CongGroup0

enumerator kNETC_CongGroup1

enum _netc_hw_mii_mode

Defines the MII/RGMII mode for data interface between the MAC and the PHY.

Values:

enumerator kNETC_XgmiiMode

XGMII mode for data interface.

enumerator kNETC_MiiMode

MII mode for data interface.

enumerator kNETC_GmiiMode

GMII mode for data interface.

enumerator kNETC_RmiiMode

RMII mode for data interface.

enumerator kNETC_RgmiiMode

RGMII mode for data interface.

enumerator kNETC_SgmiiMode

SGMII mode for data interface.

enum _netc_hw_mii_speed

Defines the speed for the *MII data interface.

Values:

enumerator kNETC_MiiSpeed10M

Speed 10 Mbps.

enumerator kNETC_MiiSpeed100M

Speed 100 Mbps.

enumerator kNETC_MiiSpeed1000M

Speed 1000 Mbps.

enumerator kNETC_MiiSpeed2500M

Speed 2500 Mbps.

enumerator kNETC_MiiSpeed5G

Speed 5Gbps.

enumerator kNETC_MiiSpeed10G

Speed 10Gbps Mbps.

enum _netc_hw_mii_duplex

Defines the half or full duplex for the MII data interface.

Values:

enumerator kNETC_MiiHalfDuplex

Half duplex mode.

enumerator kNETC_MiiFullDuplex

Full duplex mode.

typedef enum *_netc_hw_enetc_idx* netc_hw_enetc_idx_t
: ENETC index enumerator.

typedef enum *_netc_hw_switch_idx* netc_hw_switch_idx_t
: SWITCH index enumerator.

typedef enum *_netc_hw_port_idx* netc_hw_port_idx_t
: Port Resource for the NETC module.

typedef enum *_netc_hw_tc_idx* netc_hw_tc_idx_t
: Traffic class enumerator.

typedef enum *_netc_hw_bdr_idx* netc_hw_bdr_idx_t
: Enumeration for the ENETC SI BDR identifier.

typedef enum *_netc_hw_swt_cbdr_idx* netc_hw_swt_cbdr_idx_t
: Switch command BD ring index enumerator.

typedef enum *_netc_hw_classs_queue_idx* netc_hw_etm_class_queue_idx_t
: Enumerator for ETM class queue identifier.

typedef enum *_netc_hw_congestion_group_idx* netc_hw_congestion_group_idx_t
: Enumerator for the ETM congestion group.

typedef enum *_netc_hw_mii_mode* netc_hw_mii_mode_t
: Defines the MII/RGMII mode for data interface between the MAC and the PHY.

typedef enum *_netc_hw_mii_speed* netc_hw_mii_speed_t
: Defines the speed for the *MII data interface.

typedef enum *_netc_hw_mii_duplex* netc_hw_mii_duplex_t
: Defines the half or full duplex for the MII data interface.

typedef struct *_netc_psfp_kc_profile* netc_isi_kc_rule_t
: NETC PSFP kc profile configuration, the key size (not include the spmp and portp) is up to 16 bytes.

typedef struct *_netc_vlan_classify_config* netc_vlan_classify_config_t
: NETC Vlan classification config.

typedef struct *_netc_qos_classify_profile* netc_qos_classify_profile_t
: NETC Qos Classification profile file (vlan PCP/DEI to IPV/DR map)

typedef struct *_netc_ipf_config* netc_ipf_config_t
: NETC Ingress Filter config.

typedef struct *_netc_func* netc_func_t
: Register groups for the PCIe function.

typedef struct *_netc_port_hw* netc_port_hw_t
: Register groups for the Port/Link hardware.

typedef struct *_netc_enetc_hw* netc_enetc_hw_t
: Register group for the ENETC peripheral hardware.

typedef struct *_netc_timer_hw* netc_timer_hw_t
: Register group for the Timer peripheral hardware.

typedef struct *_netc_mdio_hw* netc_mdio_hw_t
: Register group for both EMDIO and port external MDIO.

getSiInstance(si)

Get SI information from netc_hw_si_idx_t.

The ENETC instance of this SI.

getSiNum(si)

The SI number in the ENETC.

getSiIdx(si)

The actaul index in the netc_hw_si_idx_t.

NETC_MSIX_TABLE_OFFSET

MSIX table address offset.

NETC_MSIX_TABLE_PBA_OFFSET

MSIX PBA address offset.

NETC_NANOSECOND_ONE_SECOND

Nanosecond in one second.

struct _netc_psfp_kc_profile

#include <fsl_netc.h> NETC PSFP kc profile configuration, the key size (not include the spmp and portp) is up to 16 bytes.

Public Members

bool etp

2 Byte Ethertype field present in the key

bool sqtp

1 Byte Sequence Tag present in the key

bool ipcpp

inner VLAN header's PCP field present in the key

bool ividp

inner VLAN ID present in the key

bool opcpp

outer VLAN header's PCP field present in the key

bool ovidp

outer VLAN ID present in the key

bool smacp

6 bytes of source MAC address present in the key

bool dmacp

6 bytes of destination MAC address present in the key

bool spmp

switch port masquerading flag present in the key

bool portp

source port present in the key

bool valid

Key Construction is valid

struct _netc_vlan_classify_config

#include <fsl_netc.h> NETC Vlan classification config.

Public Members

bool enableCustom1
: Enable/Disable custom0 ether type

uint16_t custom1EtherType
: Ethertype

bool enableCustom2
: Enable/Disable custom0 ether type

uint16_t custom2EtherType
: Ethertype

uint16_t preStandRTAGType
: 802.1CB draft 2.0 R-TAG Ethertype value. PSRTAGETR. Only applicable for switch

struct __netc__qos__classify__profile
: *#include <fsl_netc.h>* NETC Qos Classification profile file (vlan PCP/DEI to IPV/DR map)

Public Members

uint8_t ipv[16]
: Index is created from PCP (3 bits) + DEI (1 bit) field. Value is the mapped IPV for Qos.

uint8_t dr[16]
: Index is created from PCP (3 bits) + DEI (1 bit) field. Value is the mapped DR for QoS.

struct __netc__ipf__config
: *#include <fsl_netc.h>* NETC Ingress Filter config.

Public Members

bool l2DiscardMCSmac
: DOSL2CR. Discard received frames with Multicast SMAC address

bool l2DiscardSmacEquDmac
: DOSL2CR. Discard received frames with SMAC = DMAC

bool l3DiscardSipEquDip
: DOSL3CR. Discard IPV3/IPV6 source address == destination address

struct __netc__func
: *#include <fsl_netc_hw.h>* Register groups for the PCIe function.

struct __netc__port__hw
: *#include <fsl_netc_hw.h>* Register groups for the Port/Link hardware.

Public Members

NETC_PORT_Type *port
: Port Address

struct __netc__enetc__hw
: *#include <fsl_netc_hw.h>* Register group for the ENETC peripheral hardware.

Public Members

netc_func_t func
PCIE function register

NETC_ENETC_Type *base
Base register of ENETC module

NETC_SW_ENETC_Type *common
Common register of ENETC module

netc_port_hw_t portGroup
Port register group

ENETC_GLOBAL_Type *global
Global NETC address

ENETC_SI_Type *si
Station Interfce for the P/V SI

netc_msix_entry_t *msixTable
MSIX table address

struct __netc__timer__hw
#include <fsl_netc_hw.h> Register group for the Timer peripheral hardware.

Public Members

ENETC_PCI_TYPE0_Type *func
PCIE function register

ENETC_PF_TMR_Type *base
Base register address for timer module

ENETC_GLOBAL_Type *global
Global NETC register address

netc_msix_entry_t *msixTable
MSIX table address

struct __netc__mdio__hw
#include <fsl_netc_hw.h> Register group for both EMDIO and port external MDIO.

Public Members

___IO uint32_t EMDIO_CFG
External MDIO configuration register, offset: 0x1C00

___IO uint32_t EMDIO_CTL
External MDIO interface control register, offset: 0x1C04

___IO uint32_t EMDIO_DATA
External MDIO interface data register, offset: 0x1C08

___IO uint32_t EMDIO_ADDR
External MDIO register address register, offset: 0x1C0C

___I uint32_t EMDIO_STAT
External MDIO status register, offset: 0x1C10

___IO uint32_t PHY_STATUS_CFG
PHY status configuration register, offset: 0x1C20

___IO uint32_t PHY_STATUS_CTL
PHY status control register, offset: 0x1C24

___I uint32_t PHY_STATUS_DATA
PHY status data register, offset: 0x1C28

___IO uint32_t PHY_STATUS_ADDR
PHY status register address register, offset: 0x1C2C

___IO uint32_t PHY_STATUS_EVENT
PHY status event register, offset: 0x1C30

___IO uint32_t PHY_STATUS_MASK
PHY status mask register, offset: 0x1C34

struct payload

Public Members

uint8_t lbMask
Payload Last Byte Mask

uint8_t fbMask
Payload First Byte Mask

uint8_t byteOffset
Payload Byte Offset where field extraction begins

uint8_t numBytes
Specify the size (numBytes + 1) of the payload key field

uint8_t pfp
Payload field Present

union ___unnamed199___

Public Members

ENETC_PCI_TYPE0_Type *pf
PSI function

ENETC_VF_PCI_TYPE0_Type *vf
VSI function

union ___unnamed201___

Public Members

NETC_ETH_LINK_Type *eth
MAC Port Address

2.72 Hardware Common Functions

static inline uint16_t EP_IncreaseIndex(uint16_t index, uint32_t max)

uint16_t NETC_SIGetVsiIndex(*netc_vsi_number_t* vsi)

Get the VSI index.

Parameters

- vsi – The VSI number.

static inline void NETC_IPFInit(NETC_SW_ENETC_Type *base, const *netc_ipf_config_t* *config)

Set layer2/3 Dos configuration.

Parameters

- base –
- config –

void NETC_PSFPKcProfileInit(NETC_SW_ENETC_Type *base, const *netc_isi_kc_rule_t* *rule, bool enKcPair1)

Initialize the Ingress Stream Identification Key construction rule profiles.

Parameters

- base –
- rule –
- enKcPair1 –

Returns

void

void NETC_RxVlanCInit(NETC_SW_ENETC_Type *base, const *netc_vlan_classify_config_t* *config, bool enRtag)

Initialize the customer vlan type.

Parameters

- base –
- config –
- enRtag –

Returns

void

void NETC_RxQosCInit(NETC_SW_ENETC_Type *base, const *netc_qos_classify_profile_t* *profile, bool enProfile1)

Initialize the ingress QoS classification.

Parameters

- base –
- profile –
- enProfile1 –

2.73 Hardware ENETC

typedef struct *_netc_enetc_vlan_tag_t* netc_enetc_vlan_tag_t
ENETC Port outer/inner VLAN tag.

typedef struct *_netc_enetc_discard_statistic* netc_enetc_port_discard_statistic_t
PORT discard count statistic.

typedef struct *_netc_enetc_native_vlan_config_t* netc_enetc_native_vlan_config_t
ENETC Port outer/inner native VLAN config.

typedef struct *_netc_enetc_parser_config_t* netc_enetc_parser_config_t
ENETC parser configuration.

typedef struct *_netc_enetc_cap* netc_enetc_cap_t
ENETC capability.

static inline bool NETC_EnetcHasManagement(NETC_ENETC_Type *base)
Check whether ENETC has switch management capability.

Parameters

- base – NETC peripheral base address.

Returns
true or false

void NETC_EnetcGetCapability(NETC_ENETC_Type *base, *netc_enetc_cap_t* *capability)
Get ENETC capability.

Parameters

- base – NETC peripheral base address.
- capability – Pointer to capability structure.

void NETC_EnetcSetSIMacAddr(NETC_ENETC_Type *base, uint8_t si, uint8_t *macAddr)
Set MAC address for specified VSI of ENETC.

Parameters

- base –
- macAddr –

status_t NETC_EnetcConfigureSI(NETC_ENETC_Type *base, uint8_t si, const *netc_hw_enetc_si_config_t* *psConfig)
Configure SI.

Parameters

- base – ENETC peripheral base address.
- si – The SI number
- psConfig – The SI configuration

Returns
status_t

status_t NETC_EnetcSetMsixEntryNum(NETC_ENETC_Type *base, uint8_t si, uint32_t msixNum)
Set SI MSIX table entry number.

Parameters

- base – ENETC peripheral base address.
- si – The SI number.
- msixNum – The MSIX table entry number.

Returns
status_t

static inline void NETC_EnetcEnableSI(NETC_ENETC_Type *base, uint8_t si, bool enable)

Enable/Disable specified SI.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- enable – Enable/Disable SI from ENETC layer.

void NETC_EnetcGetPortDiscardStatistic(NETC_ENETC_Type *base, *netc_enetc_port_discard_statistic_t* *statistic)

Get ENETC discard statistic data.

Parameters

- base – ENETC peripheral base address.
- statistic – Statistic data.

void NETC_EnetcEnablePromiscuous(NETC_ENETC_Type *base, uint8_t si, bool enableUCPromis, bool enableMCPromis)

Enable MAC promiscuous mode.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- enableUCPromis – Enable unicast frame promiscuous.
- enableMCPromis – Enable multicast frame promiscuous.

void NETC_EnetcConfigureVlanFilter(NETC_ENETC_Type *base, uint8_t si, *netc_si_l2vf_config_t* *config)

Configure VLAN filter.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- config – Enable untagged VLAN frame promiscuous.

void NETC_EnetcAddMacAddrHash(NETC_ENETC_Type *base, uint8_t si, *netc_packet_type_t* type, uint8_t hashIndex)

Add the hash filter for the MAC address.

Hardware layer will not maitain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- type – Unicast or multicast frame type.
- hashIndex – The calculated hash index of MAC address.

void NETC_EnetcDelMacAddrHash(NETC_ENETC_Type *base, uint8_t si, *netc_packet_type_t* type, uint8_t hashIndex)

Remove the hash filter for the MAC address.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- type – Unicast or multicast frame type.
- hashIndex – The calculated hash index of MAC address.

void NETC_EnetcAddVlanHash(NETC_ENETC_Type *base, uint8_t si, uint8_t hashIndex)

Add the hash filter for the VLAN.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- hashIndex – The calculated hash index of MAC address.

void NETC_EnetcDelVlanHash(NETC_ENETC_Type *base, uint8_t si, uint8_t hashIndex)

Remove the hash filter for the VLAN.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- hashIndex – The calculated hash index of MAC address.

static inline *status_t* NETC_EnetcPortEnableTSD(NETC_ENETC_Type *base, *netc_hw_tc_idx_t* tcIdx, bool isEnable)

Enable / Disable ENETC Port Time Specific Departure (TSD) feature.

It can't work with QBV CBS

Parameters

- base –
- tcIdx –
- isEnable –

Returns

status_t

static inline void NETC_EnetcPortSetNativeVLAN(NETC_ENETC_Type *base, const *netc_enetc_native_vlan_config_t* *config, bool isOuter)

Set ENETC Rx native outer/inner VLAN.

It is used for classification when untagged frames are received by the port.

Parameters

- base –
- config –
- isOunter –

static inline void NETC_EnetcSetParser(NETC_ENETC_Type *base, const *netc_enetc_parser_config_t* *config)

Set ENETC Parser configuration.

PARCSCR and PARCE0CR - PARCE3CR.

Parameters

- base –
- config –

static inline void NETC_EnetcEnableWakeOnLan(NETC_ENETC_Type *base, bool isEnable)

Enable / Disable ENETC Wake-on-LAN mode.

Only available on ENETC 0

Parameters

- base –
- isEnable –

Returns

status_t

struct __netc_enetc_vlan_tag_t

#include <fsl_netc.h> ENETC Port outer/inner VLAN tag.

Public Members

uint16_t pcp

Priority code point

uint16_t dei

Drop eligible indicator

uint16_t vid

VLAN identifier

netc_vlan_tpid_select_t tpid

Tag protocol identifier

struct __netc_enetc_discard_statistic

#include <fsl_netc.h> PORT discard count statistic.

Public Members

uint32_t ingressDR[4]

Discard count for port ingress congestion different DR

uint32_t broadcastReject

Broadcast frame drops count due to all SI enable broadcast reject

uint32_t smacPruning

Frames discard count due to port MAC source address pruning

uint32_t unicastMacFilt

Unicast frame discard count due to port MAC filtering

uint32_t multicastMacFilt

Multicast frame discard count due to MAC filtering

uint32_t unicastVlanFilt
: Unicast frame discard count due to VLAN filtering

uint32_t multicastVlanFilt
: Multicast frame discard count due to VLAN filtering

uint32_t boradcastVlanFilt
: Broadcast frame discard count due to VLAN filtering

struct __netc__enetc__native__vlan__config__t

#include <fsl_netc.h> ENETC Port outer/inner native VLAN config.

Public Members

bool enUnderZeroVid
: Enable use the port default VLAN VID when the VID in the packet's is zero

bool enUnderNoVlan
: Enable use the port default VLAN VID when the VLAN tag is not present

netc_enetc_vlan_tag_t vlanTag
: Port native outer/inner VLAN tag, valid when enUnderZeroVid or enUnderNoVlan is true

struct __netc__enetc__parser__config__t

#include <fsl_netc.h> ENETC parser configuration.

Public Members

bool disL3Checksum
: Disable Layer 3 IPv4 Header checksum validation.

bool disL4Checksum
: Disable Layer 4 TCP and UDP checksum validation.

struct __netc__enetc__cap

#include <fsl_netc_hw_enetc.h> ENETC capability.

Public Members

bool funcSafety
: Support for safety capability.

bool wol
: Support for Wake-on-LAN in low-power mode.

bool rss
: Support for RSS.

bool tsd
: Support for time specific departure.

bool rfs
: Support for RFS.

uint32_t ipvNum
: IPV number.

uint32_t vsiNum
: VSI number.

uint32_t msixNum
: MSIX table vector/entry number.

uint32_t tcsNum
: Traffic class number.

uint16_t uchNum
: Unicast hash entry number.

uint16_t mchNum
: Multicast hash entry number.

uint16_t rxBdrNum
: Rx BD ring number.

uint16_t txBdrNum
: Tx BD ring number.

struct custEtype

Public Members

uint16_t etype
: Custom Ethertype value. Upon detecting this ether type the associated code point will be mapped to the parse summary as a Non IP code point.

bool en
: Enables the detection and mapping.

uint8_t cp
: This value is mapped to the parse summary as a Non IP code point.

2.74 Hardware Port

enum __netc_port_tgsl_status
: Port time gate scheduling gate list status.

Values:

enumerator kNETC_OperListActive
: Port operational gate control list is active.

enumerator kNETC_AdminListPending
: Administrative gate control list is pending (configured but not installed yet).

enum __netc_port_discard_tpye
: Port Tx/Rx discard counter in the datapath processing pipeline or bridge forwarding processing function.

Values:

enumerator kNETC_RxDiscard
: Discarded frames in the receive port datapath processing pipeline.

enumerator kNETC_TxDiscard
: Discarded frames in the egress datapath processing pipeline, only for switch.

enumerator kNETC_BridgeDiscard
Discarded frames in the bridge forwarding processing function, only for switch.

enum _netc_port_tpidlist
Defines Port TPID acceptance.

Values:

enumerator kNETC_OuterStanCvlan
Accept outer Standard C-VLAN 0x8100.

enumerator kNETC_OuterStanSvlan
Accept outer Standard S-VLAN 0x88A8.

enumerator kNETC_OuterCustomVlan1
Accept outer Custom VLAN as defined by CVLANR1[ETYPE].

enumerator kNETC_OuterCustomVlan2
Accept outer Custom VLAN as defined by CVLANR2[ETYPE].

enumerator kNETC_InnerStanCvlan
Accept inner Standard C-VLAN 0x8100.

enumerator kNETC_InnerStanSvlan
Accept inner Standard S-VLAN 0x88A8.

enumerator kNETC_InnerCustomVlan1
Accept inner Custom VLAN as defined by CVLANR1[ETYPE].

enumerator kNETC_InnerCustomVlan2
Accept inner Custom VLAN as defined by CVLANR2[ETYPE].

enum _netc_port_ts_select
Defines port timestamp selection.

Values:

enumerator kNETC_SyncTime
Synchronized time.

enumerator kNETC_FreeRunningTime
Free running time.

enum _netc_hw_preemption_mode
Port MAC preemption mode.

Values:

enumerator kNETC_PreemptDisable
Frame preemption is not enabled

enumerator kNETC_PreemptOn64B
Frame preemption is enabled, but transmit only preempts frames on 64B boundaries

enumerator kNETC_PreemptOn4B
Frame preemption is enabled, but transmit only preempts frames on 4B boundaries

enum _netc_hw_raf_size
Port MAC Remote Additional Fragment Size.

Values:

enumerator kNETC_RafSize64
Additional Fragment Size of 64 octets

enumerator kNETC_RafSize128
Additional Fragment Size of 128 octets

enumerator kNETC_RafSize256
Additional Fragment Size of 256 octets

enumerator kNETC_RafSize512
Additional Fragment Size of 512 octets

enum _netc_tc_sdu_type
Type of PDU/SDU (Protocol/Service Data Unit).

Note: Overhead values which adding to the transmitted frame of Length are specified by Port SDU config as follows:

- PPDU = add rxPpduBco/txPpduBco + rxMacsecBco/txMacsecBco bytes
- MPDU = add rxMacsecBco/txMacsecBco bytes
- MSDU = minus 16B (12B MAC Header + 4B FCS)

Values:

enumerator kNETC_PDU
Physical Layer PDU, Preamble, IFG, SFD along with MPDU. Not supported if cut-through frames are expected

enumerator kNETC_MPDU
MAC PDU, MAC Header, MSDU and FCS

enumerator kNETC_MSDU
MAC SDU, MPDU minus 12B MAC Header and 4B FCS. Not supported if cut-through frames are expected

enum _netc_port_sg_ogc_mode
Defines the Port's Stream Gate Open Gate Check mode.

Values:

enumerator kNETC_SGCheckSFD
Check whether frame SFD is within the open gate interval.

enumerator kNETC_SGCheckEntire
Check whether the entire frame is within the open gate interval.

enum _netc_port_intr_flags
Values:

enumerator kNETC_TxEmptyFlag
Tx FIFO empty flag.

enumerator kNETC_RxEmptyFlag
Rx FIFO empty flag.

enumerator kNETC_TxOverflowFlag
Tx overflow flag.

enumerator kNETC_TxUnderflowFlag
Tx underflow flag.

enumerator kNETC_RxOverflowFlag
Rx overflow flag.

enumerator kNETC_MagicPacketFlag
Magic packet detection indication flag.

enumerator kNETC_TxClkStopFlag
Tx clock stop detection flag.

enumerator kNETC_RxClkStopFlag
Rx clock stop detection flag.

enumerator kNETC_SpeedDuplexChangeFlag
Speed/Duplex Change flag

enumerator kNETC_MacMergeSMDErrFlag
MAC merge frame SMD error received event flag

enumerator kNETC_MacMergeAssemblyErrFlag
MAC merge frame assembly error event flag

enum __netc_port_loopback_mode_t
Defines the port MAC frame loopback mode.

Values:

enumerator kNETC_PortLpbWithExtTxClk
Port MAC frame loopback with external Tx clock.

enumerator kNETC_PortLpbWithIntTxClk
Port MAC frame loopback with internal Tx clock.

typedef enum *_netc_port_tgsl_status* netc_port_tgsl_status_t
Port time gate scheduling gate list status.

typedef enum *_netc_port_discard_tpye* netc_port_discard_tpye_t
Port Tx/Rx discard counter in the datapath processing pipeline or bridge forwarding processing function.

typedef enum *_netc_port_tpidlist* netc_port_tpidlist_t
Defines Port TPID acceptance.

typedef enum *_netc_port_ts_select* netc_port_ts_select_t
Defines port timestamp selection.

typedef struct *_netc_port_qos_mode* netc_port_qos_mode_t
Port Qos mode.

typedef struct *_netc_port_parser_config* netc_port_parser_config_t
Port Parser config.

typedef struct *_netc_port_tg_config* netc_port_tg_config_t
Port time gate config.

typedef struct *_netc_port_tg_preemption_config* netc_port_tg_preemption_config
Port time gate config when used with Frame Preemption.

typedef enum *_netc_hw_preemption_mode* netc_hw_preemption_mode_t
Port MAC preemption mode.

typedef enum *_netc_hw_raf_size* netc_hw_raf_size_t
Port MAC Remote Additional Fragment Size.

typedef struct *_netc_port_preemption_config* netc_port_preemption_config
Frame Preemption Portconfig.

typedef struct *_netc_port_tc_cbs_config* netc_port_tc_cbs_config_t

Configuration for the Credit Based Shaped for port TC.

Note: The 802.1Qav bandwidth availability parameters is is calculated as follows:

- idleSlope (bits) = portTxRate * bwWeight / 100
- sendSlope (bits) = portTxRate * (100 - bwWeight) / 100
- lowCredit (bits) = tcMaxFrameSize * (100 - bwWeight) / 100
- hiCredit (bits) calculation formula depends on the traffic class, Please refer to the Reference manual.
- hiCredit (credits) = (enetClockFrequency / portTxRate) * 100 * hiCredit (bits)

---

typedef enum *_netc_tc_sdu_type* netc_tc_sdu_type_t

Type of PDU/SDU (Protocol/Service Data Unit).

Note: Overhead values which adding to the transmitted frame of Length are specified by Port SDU config as follows:

- PPDU = add rxPpduBco/txPpduBco + rxMacsecBco/txMacsecBco bytes
- MPDU = add rxMacsecBco/txMacsecBco bytes
- MSDU = minus 16B (12B MAC Header + 4B FCS)

---

typedef struct *_netc_port_tc_sdu_config* netc_port_tc_sdu_config_t

typedef struct *_netc_port_tx_tc_config* netc_port_tx_tc_config_t

Configuration for the port Tx Traffic Class.

typedef struct *_netc_port_discard_statistic* netc_port_discard_statistic_t

Switch or ENETC port Tx/Rx/Bridge discard statistic / reason.

typedef struct *_netc_port_vlan_classify_config* netc_port_vlan_classify_config_t

Port accepted Vlan classification config.

typedef struct *_netc_port_qos_classify_configs* netc_port_qos_classify_config_t

Port Qos Classification Config.

typedef struct *_netc_port_ipf_config_t* netc_port_ipf_config_t

Port Ingress Filter Config.

typedef struct *_netc_port_psfp_isi_config* netc_port_psfp_isi_config

PSFP port config.

Port ingress stream identification config

Note: The first stream identification find IS_EID has higher precedence value than the second, and the priority of the IS_EID found by the IPF is specified by the IPF entry RRR bit. The possible orderings are as follows

- RRR = 00b : IPF > enKC0 > enKC1 > defaultISEID
- RRR = 01b : enKC0 > IPF > enKC1 > defaultISEID
- RRR = 10b : enKC0 > enKC1 > IPF > defaultISEID

---

typedef struct *_netc_port_ethmac* netc_port_ethmac_t

typedef enum *_netc_port_sg_ogc_mode* netc_port_sg_ogc_mode_t
Defines the Port's Stream Gate Open Gate Check mode.

typedef struct *_netc_port_common* netc_port_common_t
Port common configuration.

typedef enum *_netc_port_intr_flags* netc_port_intr_flags_t

typedef enum *_netc_port_loopback_mode_t* netc_port_loopback_mode_t
Defines the port MAC frame loopback mode.

status_t NETC_PortConfig(NETC_PORT_Type *base, const *netc_port_common_t* *config)
Configure specified PORT.

Parameters

- base – NETC port module base address.
- config – Port configuration structure.

Returns
status_t

void NETC_PortSetMacAddr(NETC_PORT_Type *base, const uint8_t *macAddr)
Set the MAC address.

Parameters

- handle –
- macAddr –

bool NETC_PortIsPseudo(NETC_PORT_Type *base)
Check whether this port a pseudo MAC port.

Parameters

- base – PORT peripheral base address.

void NETC_PortGetDiscardStatistic(NETC_PORT_Type *base, *netc_port_discard_tpye_t* discardType, *netc_port_discard_statistic_t* *statistic)
Get specified PORT discard counter.

Parameters

- base – NETC port module base address.
- discardType – Port discard type.
- statistic – pointer to the statistic data

void NETC_PortClearDiscardReason(NETC_PORT_Type *base, *netc_port_discard_tpye_t* discardType, uint32_t reason0, uint32_t reason1)
Clean the Port Rx discard reason. Set the related bits to 1 to clear the specific reasons.

Parameters

- base – NETC port module base address.
- discardType – Port discard type.
- reason0 –
- reason1 –

static inline uint32_t NETC_PortGetTGSListStatus(NETC_PORT_Type *base)
Get port time gate scheduling gate list status.

Parameters

- base – NETC port module base address.

Returns

Port status flags which are ORed by the enumerators in the netc_port_tgsl_status_t

void NETC_PortEthMacGracefulStop(NETC_PORT_Type *base)

Do graceful stop for Port Ethernet MAC receive/transmit.

Parameters

- base – NETC port module base address.

static inline void NETC_PortSetSpeed(NETC_PORT_Type *base, uint16_t pSpeed)

Set port speed.

Parameters

- base – NETC port module base address.
- pSpeed – Transmit Port Speed = 10Mbps * (pSpeed+1), Used by ETS, Qbu and to determine if cut-through is permissable.

NETC_PORT_MIN_FRAME_SIZE

The port supported minimum/maximum frame size.

NETC_PORT_MAX_FRAME_SIZE

struct _netc_port_qos_mode

#include <fsl_netc.h> Port Qos mode.

Public Members

uint8_t qosVlanMap

Transmit QoS to VLAN PCP Mapping Profile index, only active on switch port

uint8_t vlanQosMap

Receive VLAN PCP/DE to QoS Mapping Profile index, only active on switch port

uint8_t defaultIpv

Port default IPV

uint8_t defaultDr

Port default DR

bool enVlanInfo

Enable use VLAN info to determine IPV and DR (base on VLANIPVMPaR0/1 and VLANDRMPaR)

bool vlanTagSelect

True: Outer VLAN, False: Innner VLAN. Active when enVlanInfo is true

struct _netc_port_parser_config

#include <fsl_netc.h> Port Parser config.

Public Members

uint8_t l2PloadCount

L2 payload fields size in bytes

bool enableL3Parser

Enable/Disable parser for L3

uint8_t l3PayloadCount

L3 payload fields size in bytes

bool enableL4Parser

Enable/Disable parser for L4

uint8_t l4PayloadCount

L4 payload fields size in bytes

struct __netc_port_tg_config

#include <fsl_netc.h> Port time gate config.

Public Members

uint16_t advOffset

Advance time offset in ns.

uint32_t holdSkew

Hold-Skew in ns, not effective on ports connected to a pseudo-MAC

struct __netc_port_tg_preemption_config

#include <fsl_netc.h> Port time gate config when used with Frame Preemption.

Public Members

uint16_t holdAdvance

the amount of time in ns prior to the Set-And-Hold-MAC time slot for asserting a Hold request. Used with frame Preemption.

uint16_t releaseAdvance

the amount of time in ns prior to the Set-And-Release-MAC time slot for asserting a Release request. Used with Frame Preemption.

struct __netc_port_preemption_config

#include <fsl_netc.h> Frame Preemption Portconfig.

Public Members

bool enMergeVerify

Enable verify the merged preemption frame, need to enable when preemptMode is not zero

uint8_t mergeVerifyTime

The nominal wait time between verification attempts in milliseconds, range in 1 ~ 128

netc_hw_preemption_mode_t preemptMode

When set to not zero, PMAC frames may be preempted by EMAC frames

netc_hw_raf_size_t raf_size

Additional Fragment Size. Indicates the smallest sized fragments that can be sent on Tx

bool PreemptionActive

Local preemption active. Indicates whether preemption is active for this port. This bit will be set if preemption is both enabled and has completed the verification process

struct __netc_port_tc_cbs_config

#include <fsl_netc.h> Configuration for the Credit Based Shaped for port TC.

Note: The 802.1Qav bandwidth availability parameters is is calculated as follows:

- idleSlope (bits) = portTxRate * bwWeight / 100
- sendSlope (bits) = portTxRate * (100 - bwWeight) / 100
- lowCredit (bits) = tcMaxFrameSize * (100 - bwWeight) / 100
- hiCredit (bits) calculation formula depends on the traffic class, Please refer to the Reference manual.
- hiCredit (credits) = (enetClockFrequency / portTxRate) * 100 * hiCredit (bits)

Public Members

uint8_t bwWeight

Percentage units of the port transmit rate and the credit-based shaper (range from 0 ~ 100), the sum of all traffic class credit-based shaper's bandwidth cannot exceed 100

uint32_t hiCredit

The maximum allowed accumulation of credits when conflicting transfers occur, in credit units ((enetClockFrequency / portTxRate) * 100)

struct __netc_port_tc_sdu_config

#include <fsl_netc.h>

Public Members

bool enTxMaxSduCheck

Enable Tx Max SDU check for Store and Forward frames, the frame which greater than maxSduSized wiil be discarded, Cut-Through frames will always perform Max SDU check

netc_tc_sdu_type_t sduType

Specifies the type of PDU/SDU whose length is being validated as seen on the link

uint16_t maxSduSized

Transmit Maximum SDU size in bytes, the dequeued frame will be discarded when it SDU size exceeds this value

struct __netc_port_tx_tc_config

#include <fsl_netc.h> Configuration for the port Tx Traffic Class.

Public Members

bool enPreemption

Frames from traffic class are transmitted on the preemptable MAC, not supported on internal port (ENETC 1 port and Switch port 4)

bool enTcGate

Enable the traffic class gate when no gate control list is operational, or when time gate scheduling is disabled.

bool enableTsd

Enable Time Specific Departure traffic class, only applicable to ENETC

bool enableCbs

Enable Credit based shaper for traffic class

netc_port_tc_cbs_config_t cbsCfg

Configure transmit traffic class credit based shaper (PTC0CBSR0/PTC0CBSR1) if enableCbs set to ture

struct __netc_port_discard_statistic

#include <fsl_netc.h> Switch or ENETC port Tx/Rx/Bridge discard statistic / reason.

Public Members

uint32_t count

Count of discarded frames. PRXDCR, PTXDCR or BPDCR.

uint32_t reason0

Discard Reason. Find bit detail from PT/RXDCRR0 or BPDCRR0.

uint32_t reason1

Discard Reason. Find bit detail from PT/RXDCRR1 or BPDCRR1.

struct __netc_port_vlan_classify_config

#include <fsl_netc.h> Port accepted Vlan classification config.

Public Members

uint8_t innerMask

Bitmap identifying which TPIDs are acceptable as Inner VLAN tag. See PTAR

uint8_t outerMask

Bitmap identifying which TPIDs are acceptable as Outter VLAN tag. See PTAR

struct __netc_port_qos_classify_configs

#include <fsl_netc.h> Port Qos Classification Config.

Public Members

uint8_t vlanQosMap

Receive VLAN PCP/DE to QoS Mapping Profile index

uint8_t defaultIpv

Port default IPV

uint8_t defaultDr

Port default DR

bool enVlanInfo

Enable use VLAN info to determine IPV and DR ,base on VLAN to IPV map (VLANIPVMPaR0/1) and VLAN to DR map (VLANDRMPaR)

bool vlanTagSelect

True: Use received Outer VLAN, False: Use received Innner VLAN. Active when enVlanInfo is true

struct __netc_port_ipf_config_t

#include <fsl_netc.h> Port Ingress Filter Config.

Public Members

bool enL2Dos

Enable port L2 Ethernet DoS Protection

bool enL3Dos

Enable port L3 IP DoS Protection

bool enIPFTable

Enable port IPF lookup

struct __netc__port__psfp__isi__config

#include <fsl_netc.h> PSFP port config.

Port ingress stream identification config

Note: The first stream identification find IS_EID has higher precedence value than the second, and the priority of the IS_EID found by the IPF is specified by the IPF entry RRR bit. The possible orderings are as follows

- RRR = 00b : IPF > enKC0 > enKC1 > defaultISEID
- RRR = 01b : enKC0 > IPF > enKC1 > defaultISEID
- RRR = 10b : enKC0 > enKC1 > IPF > defaultISEID

Public Members

uint16_t defaultISEID

Default Ingress Stream Entry ID, has lower precedence value than ISI entry and IPF entry defined IS_EID. 0xFFFF means NULL

bool enKC1

Enable do the second stream identification with key construction rule 1 or rule 3

bool enKC0

Enable do the first stream identification with key construction rule 0 or rule 2

bool kcPair

Indicates which Key Construction pair to use for this port, false - user pair0. true - use pair1 only applicable for Switch

struct __netc__port__ethmac

#include <fsl_netc.h>

Public Members

bool enableRevMii

Enable RevMII mode.

netc_port_ts_select_t txTsSelect

Tx timestamp clock source.

bool isTsPointPhy

True: Timestamp is captured based on PHY SFD detect pulse on Rx and Tx for 2-step timestamping. False: Based on SFD detect at boundary of MAC merge layer and pins/protocol gaskets.

netc_hw_mii_mode_t miiMode

MII mode.

netc_hw_mii_speed_t miiSpeed

MII Speed.

netc_hw_mii_duplex_t miiDuplex

MII duplex.

bool enTxPad

Enable ETH MAC Tx Padding, which will pad the frame to a minimum of 60 bytes and append 4 octets of FCS.

uint8_t rxMinFrameSize

Receive Minimum Frame Length size in bytes, range in 18 ~ 64, received frames shorter than 18B are discarded silently. Both for express MAC and preemptable MAC.

uint16_t rxMaxFrameSize

Receive Maximum Frame Length size in bytes, up to 2000, received frames that exceed this stated maximum are truncated. Both for express MAC and preemptable MAC.

netc_port_preemption_config PreemptionConfig

Frame Preemption configuration

bool rgmiiClkStop

True: RGMII transmit clock is stoppable during low power idle. False: It's not stoppable.

bool enableHalfDuplexFlowCtrl

Enable/Disable half-duplex flow control.

uint16_t maxBackPressOn

Maximum amount of time backpressure can stay asserted before stopping to prevent excess defer on link partner, in byte times.

uint16_t minBackPressOff

Minimum amount of time backpressure will stay off after reaching the ON max, before backpressure can reassert after checking if icm_pause_notification is still or again asserted, in byte times.

uint32_t txWakeupTimeCycleEEE

Energy Efficient Ethernet feature. Defines the number of NETC cycles (which represents time) required by the PHY to wait before transmitting a new frame after the application has indicated it wants to end the low power state.

uint32_t txSleepTimeCycleEEE

Energy Efficient Ethernet feature. Defines the number of NETC cycles (which represents time) where Tx is idle before mac transmits low power EEE. A value of 0 does not activate low power EEE transmission.

struct __netc__port__common

#include <fsl_netc.h> Port common configuration.

Public Members

netc_port_vlan_classify_config_t acceptTpid

Port acceptable VLAN tpid configure.

netc_port_ts_select_t rxTsSelect

Eth MAC Rx or pseudo MAC Tx timestamp clock source

uint16_t pSpeed

Transmit Port Speed = 10Mbps * (pSpeed+1), Used by ETS, Qbu and to determine if cut-through is permissable

uint8_t rxMacsecBco

Port receive MACSec byte count overhead which due to MACSec encapsulation

uint8_t rxPpduBco

Port receive PPDU Byte count overhead which includes IPG, SFD and Preamble

uint8_t txMacsecBco

Port transmit MACSec byte count overhead which due to MACSec encapsulation

uint8_t txPpduBco

Port transmit PPDU Byte count overhead which includes IPG, SFD and Preamble

netc_port_sg_ogc_mode_t ogcMode

Stream Gate Open Gate Check mode, 0b is check whether SFD is within the open gate interval, 1b is check whether the entire frame is within the open gate interval

uint32_t pDelay

Link propagation delay in ns

uint8_t macAddr[6]

Port MAC address, used for Switch egress frame modification action or ENETC SI0 primary MAC address

netc_port_qos_classify_config_t qosMode

Port Rx Qos Classification config

netc_port_ipf_config_t ipfCfg

Port ingress port filter configuration

netc_port_tg_config_t timeGate

Port Tx time gate config

netc_port_parser_config_t parser

Port Rx Parser config

2.75 Hardware Port MAC

enum __netc_port_phy_mac_type

Defines the Ethernet MAC physical port type.

Values:

enumerator kNETC_ExpressMAC

The MAC which handles express traffic when frame preemption is enabled or handles all traffic when frame preemption is disabled.

enumerator kNETC_PreemptableMAC

The MAC which handles preemptive traffic when frame preemption is enabled.

enum __netc_port_preemption_verify_status

Definesthe state of the mac merge sublayer with respect to verification as defined in IEEE Std 802.3br-2016.

Values:

enumerator kNETC_VerifyDisable
Verification is disabled

enumerator kNETC_VerifyInProgress
Verification is in progress

enumerator kNETC_VerifySuccess
Verification was successful

enumerator kNETC_VerifyFaile
Verification failed

enumerator kNETC_VerifyUndefined
Verification is in an undefined state

typedef enum *_netc_port_phy_mac_type* netc_port_phy_mac_type_t
Defines the Ethernet MAC physical port type.

typedef enum *_netc_port_preemption_verify_status* netc_port_preemption_verify_status_t
Definesthe state of the mac merge sublayer with respect to verification as defined in IEEE Std 802.3br-2016.

typedef struct *_netc_port_phy_mac_preemption_status* netc_port_phy_mac_preemption_status_t
Port MAC preemption Status.

typedef struct *_netc_port_phy_mac_traffic_statistic* netc_port_phy_mac_traffic_statistic_t
Ethernet MAC physical port traffic (Tx/Rx) statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

typedef struct *_netc_port_phy_mac_discard_statistic* netc_port_phy_mac_discard_statistic_t
Ethernet MAC physical port frame discard/errors status statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

typedef struct *_netc_port_phy_mac_preemption_statistic*
netc_port_phy_mac_preemption_statistic_t
Ethernet physical MAC port preemption (Tx/Rx) related statistics counters.

typedef struct *_netc_port_pseudo_mac_traffic_statistic* netc_port_pseudo_mac_traffic_statistic_t
Ethernet pseudo MAC port traffic (Tx/Rx) statistics counters.

uint32_t NETC_GetPortMacInterruptFlags(NETC_ETH_LINK_Type *base,
netc_port_phy_mac_type_t macType)
Get port MAC interrupt flags.

Parameters

- base – NETC ETH link base register.
- mac – MAC type.

void NETC_ClearPortMacInterruptFlags(NETC_ETH_LINK_Type *base, *netc_port_phy_mac_type_t*
macType, uint32_t mask)
Clear port MAC interrupt flags.

Parameters

- base – NETC ETH link base register.
- mac – MAC type.
- mask – Bit mask of interrupts to enable. See netc_port_intr_flags_t for the set of constants that should be OR'd together to form the bit mask.

void NETC_EnablePortMacInterrupts(NETC_ETH_LINK_Type *base, *netc_port_phy_mac_type_t* macType, uint32_t mask, bool enable)

Enable/Disable port MAC interrupts.

Parameters

- base – NETC ETH link base register.
- mac – MAC type.
- mask – Bit mask of interrupts to enable. See netc_port_intr_flags_t for the set of constants that should be OR'd together to form the bit mask.
- enable – Enable/Disable interrupts.

status_t NETC_PortEnableLoopback(NETC_ETH_LINK_Type *base, *netc_port_loopback_mode_t* loopMode, bool enable)

Enable/Disable Loopback for specified MAC.

Parameters

- base –
- loopMode –
- enable –

Returns
status_t

static inline *netc_hw_mii_mode_t* NETC_PortGetMIIMode(NETC_ETH_LINK_Type *base)

Get ethernet MAC port MII mode.

Parameters

- base – Ethernet MAC port peripheral base address.

Returns
netc_hw_mii_mode_t

status_t NETC_PortSetMII(NETC_ETH_LINK_Type *base, *netc_hw_mii_mode_t* miiMode, *netc_hw_mii_speed_t* speed, *netc_hw_mii_duplex_t* duplex)

Configure ethernet MAC interface mode, speed and duplex for specified PORT.

Parameters

- base – Ethernet MAC port peripheral base address.
- miiMode – The Ethernet MAC MII mode.
- speed – The Ethernet MAC speed.
- duplex – The Ethernet MAC duplex.

Returns
status_t

status_t NETC_PortSetMaxFrameSize(NETC_ETH_LINK_Type *base, uint16_t size)

Set the maximum supported received frame size.

Parameters

- base – Ethernet MAC port peripheral base address.
- size – Maximum frame size to set.

Returns
status_t

status_t NETC_PortConfigEthMac(NETC_ETH_LINK_Type *base, const *netc_port_ethmac_t* *config)

Configure ethernet MAC for specified PORT. Set the MII mode, speed/duplex, reverse mode, etc.

Parameters

- base – Ethernet MAC port peripheral base address.
- config – The Ethernet MAC configuration.

Returns
status_t

static inline void NETC_PortConfigEthMacPreemption(NETC_ETH_LINK_Type *base, const *netc_port_preemption_config* *config)

Configure ethernet MAC for Frame preemption on specified PORT.

Parameters

- base – Ethernet MAC port peripheral base address.
- config – The Ethernet MAC configuration.

Returns
status_t

static inline void NETC_PortSoftwareResetEthMac(NETC_ETH_LINK_Type *base)

Do software reset for Ethernet MAC.

Note: This can reset all statistic counters.

Parameters

- base – PORT MAC peripheral base address.

static inline void NETC_PortGetPhyMacPreemptionStatus(NETC_ETH_LINK_Type *base, *netc_port_phy_mac_preemption_status_t* *status)

Get Ethernet MAC preemption status.

Parameters

- base – PORT MAC peripheral base address.
- status – Point to the buffer which store status.

static inline void NETC_PortGetPhyMacPreemptionControl(NETC_ETH_LINK_Type *base, *netc_port_preemption_config* *config)

Get Ethernet MAC preemption control parameters.

Parameters

- base – PORT MAC peripheral base address.
- config – Pointer to the NETC port preemption configuration.

void NETC_PortGetPhyMacTxStatistic(NETC_ETH_LINK_Type *base, *netc_port_phy_mac_type_t* macType, *netc_port_phy_mac_traffic_statistic_t* *statistic)

Get Ethernet MAC Tx Traffic Statistics .

Parameters

- base – PORT MAC peripheral base address.
- macType – Express MAC or Preemptable MAC.

- status – Point to the buffer which store statistics.

void NETC_PortGetPhyMacRxStatistic(NETC_ETH_LINK_Type *base, *netc_port_phy_mac_type_t* macType, *netc_port_phy_mac_traffic_statistic_t* *statistic)

Get Ethernet MAC Rx Traffic Statistics .

Parameters

- base – PORT MAC peripheral base address.
- macType – Express MAC or Preemptable MAC.
- status – Point to the buffer which store statistics.

void NETC_PortGetPhyMacDiscardStatistic(NETC_ETH_LINK_Type *base, *netc_port_phy_mac_type_t* macType, *netc_port_phy_mac_discard_statistic_t* *statistic)

Get Ethernet MAC Rx/Tx Drops/Errors Statistics .

Parameters

- base – PORT MAC peripheral base address.
- macType – Express MAC or Preemptable MAC.
- status – Point to the buffer which store statistics.

void NETC_PortGetPhyMacPreemptionStatistic(NETC_ETH_LINK_Type *base, *netc_port_phy_mac_preemption_statistic_t* *statistic)

Get Ethernet Preemptable MAC preemption Statistics .

Parameters

- base – PORT MAC peripheral base address.
- status – Point to the buffer which store statistics.

status_t NETC_PortConfigTxIpgPreamble(NETC_ETH_LINK_Type *base, uint8_t preambleCnt, uint8_t ipgLen)

Configure the port MAC flexible preamble and IPG length.

Parameters

- base – PORT MAC peripheral base address.
- preambleCnt – Flexible Preamble Count. Valid values are 1 to 7(default).
- ipgLen – Transmit inter-packet gap value. Valid values are 4 to 24(default) with 12 being default.

Returns

status_t

struct __netc_port_phy_mac_preemption_status

#include <fsl_netc.h> Port MAC preemption Status.

Public Members

bool mergeActive

Transmit preemption is active or not

netc_port_preemption_verify_status_t verifyStatus

Transmit preemption is active or not

struct __netc_port_phy_mac_traffic_statistic

#include <fsl_netc.h> Ethernet MAC physical port traffic (Tx/Rx) statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

Public Members

uint64_t totalOctet

Count of MAC received/transmitted good/error Ethernet octets.

uint64_t validOctet

Count of MAC received/transmitted good Ethernet octets.

uint64_t pauseFrame

Count of MAC received/transmitted valid PAUSE frames.

uint64_t validFrame

Count of MAC received/transmitted valid frames.

uint64_t vlanFrame

Count of MAC received/transmitted valid VLAN tagged frames.

uint64_t unicastFrame

Count of MAC received/transmitted valid unicast frames.

uint64_t multicastFrame

Count of MAC received/transmitted valid multicast frames.

uint64_t boradcastFrame

Count of MAC received/transmitted valid broadcast frames.

uint64_t totalPacket

Count of MAC received/transmitted good/error packets.

uint64_t rxMinPacket

Count of MAC received min to 63-octet packets.

uint64_t total64BPacket

Count of MAC received/transmitted 64 octet packets.

uint64_t total65To127BPacket

Count of MAC received/transmitted 65 to 127 octet packets.

uint64_t total128To255BPacket

Count of MAC received/transmitted 128 to 255 octet packets.

uint64_t total256To511BPacket

Count of MAC received/transmitted 256 to 511 octet packets.

uint64_t total511To1023BPacket

Count of MAC received/transmitted 512 to 1023 octet packets.

uint64_t total1024To1522BPacket

Count of MAC received/transmitted 1024 to 1522 octet packets.

uint64_t total1523ToMaxBPacket

Count of MAC received/transmitted 1523 to Max octet packets.

uint64_t controlPacket

Count of MAC received/transmitted control packets.

struct __netc__port__phy__mac__discard__statistic

#include <fsl_netc.h> Ethernet MAC physical port frame discard/errors status statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

Public Members

uint64_t rxError

Count of MAC received error frames.

uint64_t rxUndersized

Count of MAC received undersized frames.

uint64_t rxOversized

Count of MAC received oversized frames.

uint64_t rxErrorFCS

Count of MAC received check sequence (FCS) error frames.

uint64_t rxFragment

Count of MAC frames which is shorter than the MIN length and received with a wrong FCS/CRC.

uint64_t rxJabber

Count of MAC frames which is larger than the MAX length and received with a wrong FCS/CRC.

uint64_t rxDiscard

Count of MAC drops frame.

uint64_t rxDiscardNoTruncated

Count of MAC non-truncated drops frame.

uint64_t txErrorFCS

Count of MAC transmitted bad FCS frames.

uint64_t txUndersized

Count of MAC transmitted less than 64B with good FCS frames.

struct __netc__port__phy__mac__preemption__statistic

#include <fsl_netc.h> Ethernet physical MAC port preemption (Tx/Rx) related statistics counters.

Public Members

uint32_t rxReassembledFrame

Count of MAC frames that were successfully reassembled and delivered to the MAC.

uint32_t rxReassembledError

Count of MAC frames with reassembly errors.

uint32_t rxMPacket

Count of the number of additional mPackets received due to preemption.

uint32_t rxSMDError

Count of received MAC frames / MAC frame fragments rejected due to unknown SMD.

uint32_t txPreemptionReq

Count of the number of tx preemption HOLD requests.

uint32_t txMPacket
: Count of the number of additional mPackets transmitted due to preemption.

struct __netc_port_pseudo_mac_traffic_statistic
: *#include <fsl_netc.h>* Ethernet pseudo MAC port traffic (Tx/Rx) statistics counters.

Public Members

uint64_t totalOctet
: Count of MAC received/transmitted octets.

uint64_t unicastFrame
: Count of MAC received/transmitted unicast frames.

uint64_t multicastFrame
: Count of MAC received/transmitted multicast frames.

uint64_t boradcastFrame
: Count of MAC received/transmitted broadcast frames .

2.76 Hardware Port Rx

static inline void NETC_PortSetParser(NETC_PORT_Type *base, const *netc_port_parser_config_t* *config)

Set port Parser.

Parameters

- base – PORT peripheral base address.
- config – The port Parser configuration.

static inline void NETC_PortSetVlanClassify(NETC_PORT_Type *base, const *netc_port_vlan_classify_config_t* *config)

Set port acceptable VLAN.

Parameters

- base – PORT peripheral base address.
- config – The port acceptable vlan classification configuration.

static inline *status_t* NETC_PortSetQosClassify(NETC_PORT_Type *base, const *netc_port_qos_classify_config_t* *config)

Set port Qos Classification.

Parameters

- base – PORT peripheral base address.
- config – The port QoS classification configuration.

Returns
status_t

static inline void NETC_PortSetIPF(NETC_PORT_Type *base, const *netc_port_ipf_config_t* *config)

Set port ingress filter.

Parameters

- base – PORT peripheral base address.

- config – The port ingress filter configuration.

static inline void NETC_PortSetISI(NETC_PORT_Type *base, const *netc_port_psfp_isi_config* *config)

Set port Ingress stream identification.

Parameters

- base – PORT peripheral base address.
- config – The port Ingress stream identification configuration.

2.77 Hardware Port Tx

static inline *status_t* NETC_PortConfigTGS(NETC_PORT_Type *base, const *netc_port_tg_config_t* *config)

Configure the port time gating Scheduling.

Parameters

- base –
- config –

Returns
status_t

status_t NETC_PortConfigTcCBS(NETC_PORT_Type *base, *netc_hw_tc_idx_t* tcIdx, const *netc_port_tc_cbs_config_t* *config)

Config the Credit-Based Shaper (CBS) for specified Port Traffic Class.

Parameters

- base –
- tcIdx –
- config –

Returns
status_t

static inline *status_t* NETC_PortConfigTcMaxSDU(NETC_PORT_Type *base, *netc_hw_tc_idx_t* tcIdx, const *netc_port_tc_sdu_config_t* *config)

Config the max Transmit max SDU for specified Port Traffic Class.

Parameters

- base –
- tcIdx –
- config –

Returns
status_t

static inline *status_t* NETC_PortGetTcMaxSDU(NETC_PORT_Type *base, *netc_hw_tc_idx_t* tcIdx, *netc_port_tc_sdu_config_t* *config)

Read the max Transmit max SDU for specified Port Traffic Class.

Parameters

- base –
- tcIdx –

- config –

Returns
status_t

static inline void NETC_PortConfigTcPreemption(NETC_PORT_Type *base, *netc_hw_tc_idx_t* tcIdx, const bool enable)

Config Frame Preemption for specified Port Traffic Class.

Parameters

- base – NETC PORT base peripheral address
- tcIdx – traffic class index
- enable – enable/disable feature on traffic class

static inline void NETC_PortGetTcPreemption(NETC_PORT_Type *base, *netc_hw_tc_idx_t* tcIdx, bool *enabled)

Get Frame Preemption configuration for specified Port Traffic Class.

Parameters

- base – NETC PORT base peripheral address
- tcIdx – traffic class index
- enabled – port tx traffic class enabled flag

static inline void NETC_PortGetTGSFPConfig(NETC_PORT_Type *base, *netc_port_tg_preemption_config* *config)

Get the port time gating Scheduling configuration specifc for when used with Frame Preemption.

Parameters

- base – NETC PORT base peripheral address
- config –

2.78 Hardware Station Interface(SI)

NETC_SI_TXDESCRIP_RD_FL(n)

Defines for read format.

NETC_SI_TXDESCRIP_RD_TSE_MASK

NETC_SI_TXDESCRIP_RD_TXSTART(n)

NETC_SI_TXDESCRIP_RD_L3START(n)

NETC_SI_TXDESCRIP_RD_IPCS(n)

NETC_SI_TXDESCRIP_RD_L3HDRSIZE(n)

NETC_SI_TXDESCRIP_RD_L3T(n)

NETC_SI_TXDESCRIP_RD_L4T(n)

NETC_SI_TXDESCRIP_RD_L4CS(n)

NETC_SI_TXDESCRIP_RD_LSO(n)

NETC_SI_TXDESCRIP_RD_LSO_MASK

enum _netc_hw_enetc_si_vlan_type

VLAN Ethertypes.

Values:

enumerator kNETC_ENETC_StanCVlan

Standard C-VLAN 0x8100.

enumerator kNETC_ENETC_StanSVlan

Standard S-VLAN 0x88A8.

enumerator kNETC_ENETC_CustomVlan1

Custom VLAN as defined by CVLANR1[ETYPE].

enumerator kNETC_ENETC_CustomVlan2

Custom VLAN as defined by CVLANR2[ETYPE].

enum _netc_hw_enetc_si_rxr_group

SI receive BD ring group index.

Values:

enumerator kNETC_SiBDRGroupOne

SI Rx BD ring group index one.

enumerator kNETC_SiBDRGroupTwo

SI Rx BD ring group index two.

enum _netc_tx_bdr_flags

Status/Interrupts flags for the TX BDR. Each flag get its own bit thus it support bit AND/OR operation.

Values:

enumerator kNETC_TxBDRSystemBusErrorFlag

enumerator kNETC_TxBDRBusyFlag

enumerator kNETC_TxBDRStatusFlagsMask

enum _netc_rx_bdr_flags

Status/Interrupts flags for the RX BDR. Each flag get its own bit thus it support bit AND/OR operation.

Values:

enumerator kNETC_RxBDRSystemBusErrorFlag

enumerator kNETC_RxBDREmptyFlag

enum _netc_psi_msg_flags_t

PSI message interrupt type.

Values:

enumerator kNETC_PsiRxMsgFromVsi1Flag

Message receive interrupt enable, initiated by VSI1.

enumerator kNETC_PsiRxMsgFromVsi2Flag

Message receive interrupt enable, initiated by VSI2.

enumerator kNETC_PsiRxMsgFromVsi3Flag

Message receive interrupt enable, initiated by VSI3.

enumerator kNETC_PsiFLRFromVsi1Flag
Function level reset interrupt enable, initiated by VSI1.

enum _netc_vsi_msg_flags
VSI message interrupt flags.

Values:

enumerator kNETC_VsiMsgTxFlag
Message sent to PSI has completed and response received.

enumerator kNETC_VsiMsgRxFlag
Message received from PSI.

enum _netc_vsi_number
VSI number bit map, VSI1 starts from bit1.

Values:

enumerator kNETC_Vsi1

enumerator kNETC_Vsi2

enumerator kNETC_Vsi3

enum _enetc_si_bdr_priority
ENETC Station Interface BD Ring priority enumeration.

Values:

enumerator kNETC_SIBdrPriorityLowest

enumerator kNETC_SIBdrPriority0

enumerator kNETC_SIBdrPriority1

enumerator kNETC_SIBdrPriority2

enumerator kNETC_SIBdrPriority3

enumerator kNETC_SIBdrPriority4

enumerator kNETC_SIBdrPriority5

enumerator kNETC_SIBdrPriority6

enumerator kNETC_SIBdrPriority7

enumerator kNETC_SIBdrPriorityHighest

typedef enum *_netc_hw_enetc_si_vlan_type* netc_hw_enetc_si_vlan_type
VLAN Ethertypes.

typedef enum *_netc_hw_enetc_si_rxr_group* netc_hw_enetc_si_rxr_group
SI receive BD ring group index.

typedef struct *_netc_hw_enetc_si_config* netc_hw_enetc_si_config_t
Station Interface configuration.

typedef struct *_netc_si_l2mf_config* netc_si_l2mf_config_t
L2 Mac Filter Configuration for SI.

typedef struct *_netc_si_l2vf_config* netc_si_l2vf_config_t
L2 VLAN Filter Configuration for SI.

typedef struct *_netc_si_discard_statistic* netc_si_discard_statistic_t

SI frame drop statistic struct.

typedef struct *_netc_si_traffic_statistic* netc_si_traffic_statistic_t

SI traffic statistic struct.

typedef struct *_netc_si_config* netc_si_config_t

SI Configuration.

typedef union *_netc_tx_bd* netc_tx_bd_t

Transmit Buffer Descriptor format.

A union type cover the BD used as Standard/Extended/WriteBack format.

typedef union *_netc_rx_bd* netc_rx_bd_t

Receive Buffer Descriptor format.

typedef struct *_netc_tx_bdr_config* netc_tx_bdr_config_t

Configuration for the SI Tx Buffer Descriptor Ring Configuration.

typedef struct *_netc_tx_bdr* netc_tx_bdr_t

Transmit BD ring handler data structure.

typedef enum *_netc_tx_bdr_flags* netc_tx_bdr_flags_t

Status/Interrupts flags for the TX BDR. Each flag get its own bit thus it support bit AND/OR operation.

typedef struct *_netc_rx_bdr_config* netc_rx_bdr_config_t

Configuration for the SI Rx Buffer Descriptor Ring Configuration.

typedef struct *_netc_rx_bdr* netc_rx_bdr_t

Receive BD ring handler data structure.

typedef enum *_netc_rx_bdr_flags* netc_rx_bdr_flags_t

Status/Interrupts flags for the RX BDR. Each flag get its own bit thus it support bit AND/OR operation.

typedef struct *_netc_bdr_config* netc_bdr_config_t

Configuration for the buffer descriptors ring.

typedef enum *_netc_psi_msg_flags_t* netc_psi_msg_flags_t

PSI message interrupt type.

typedef enum *_netc_vsi_msg_flags* netc_vsi_msg_flags_t

VSI message interrupt flags.

typedef enum *_netc_vsi_number* netc_vsi_number_t

VSI number bit map, VSI1 starts from bit1.

typedef struct *_netc_psi_rx_msg* netc_psi_rx_msg_t

PSI receive message information.

typedef struct *_netc_vsi_msg_tx_status* netc_vsi_msg_tx_status_t

VSI message transmit status.

typedef enum *_enetc_si_bdr_priority* enetc_si_bdr_priority_t

ENETC Station Interface BD Ring priority enumeration.

static inline void NETC_ClearTxDescriptor(*netc_tx_bd_t* *txDesc)

static inline void NETC_CopyTxDescriptor(*netc_tx_bd_t* *txDescDst, *netc_tx_bd_t* *txDescSrc)

static inline void NETC_SIEnable(ENETC_SI_Type *base, bool enable)

Enable the Station Interface(SI)

Parameters

- base –

static inline void NETC_SIRxRingEnable(ENETC_SI_Type *base, uint8_t ring, bool enable)

Enable the specified Rx BD ring.

Parameters

- base –
- ring – The ring index.
- base – Enable/Disable the ring.

static inline void NETC_SIEnablePromisc(ENETC_SI_Type *base, *netc_packet_type_t* type, bool enable)

Enable/Disable unicast/multicast/boardcast promisc mode for specified SI.

Parameters

- base –
- type –
- enable –

static inline void NETC_SISetTxProducer(ENETC_SI_Type *base, uint8_t ring, uint16_t producer)

Set producer index of specified Tx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.
- producer – The producer index of specified ring.

static inline uint16_t NETC_SIGetTxConsumer(ENETC_SI_Type *base, uint8_t ring)

Get consumer index of specified Tx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.

Returns

consumer The consumer index of specified ring.

static inline void NETC_SISetRxConsumer(ENETC_SI_Type *base, uint8_t ring, uint16_t consumer)

Set consumer index of specified Rx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.
- consumer – The consumer index of specified ring.

static inline uint16_t NETC_SIGetRxProducer(ENETC_SI_Type *base, uint8_t ring)

Get producer index of specified Rx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.

Returns
producer The producer index of specified ring.

status_t NETC_SIConfigTxBDR(ENETC_SI_Type *base, uint8_t ring, const *netc_tx_bdr_config_t* *bdrConfig)

Configure the Transmit Buffer Descriptor Ring for specified SI.

Parameters

- base – SI base address.
- ring – BD ring index.
- bdrConfig – The BD ring configuration.

Returns
status_t

status_t NETC_SIConfigRxBDR(ENETC_SI_Type *base, uint8_t ring, const *netc_rx_bdr_config_t* *bdrConfig)

Configure the Rx Buffer Descriptor Ring for specified SI.

Parameters

- base – SI base address.
- ring – BD ring index.
- bdrConfig – The BD ring configuration.

Returns
status_t

static inline void NETC_SIMapVlanToIpv(ENETC_SI_Type *base, uint8_t pcpDei, uint8_t ipv)

Enable the mapping of VLAN to IPV.

Parameters

- base – SI base address.
- pcpDei – The VLAN tag pcp dei value (use NETC_VLAN_PCP_DEI_VALUE macro).
- ipv – The IPV value for this VLAN mapping.

static inline void NETC_SIEnableVlanToIpv(ENETC_SI_Type *base, bool enable)

Enable the mapping of VLAN to IPV.

Parameters

- base – SI base address.
- enable – Whether enable mapping.

Returns
status_t

static inline void NETC_SIMapIpvToRing(ENETC_SI_Type *base, uint8_t ipv, uint8_t ring)

Set the IPV to ring mapping.

Parameters

- base – SI base address.
- ipv – IPV value to be mapped.
- ring – The Rx BD ring index to be mapped.

Returns
status_t

static inline void NETC_SISetRxBDRGroup(ENETC_SI_Type *base, uint8_t groupNum, uint8_t ringPerGroup)

Set the group of Rx BD ring.

Parameters

- base – SI base address.
- groupNum – Total group number.
- ringPerGroup – Rings per group.

Returns
status_t

static inline void NETC_SISetDefaultRxBDRGroup(ENETC_SI_Type *base, *netc_hw_enetc_si_rxr_group* groupIdx)

Set the default used receive Rx BD ring group.

Note: The IPV mapped ring index is the relative index inside the default used group.

Parameters

- base – SI base address.
- groupNum – The default Rx group index.

Returns
status_t

static inline void NETC_SICleanTxIntrFlags(ENETC_SI_Type *base, uint16_t txFrameIntrMask, uint16_t txThresIntrMask)

Clean the SI transmit interrupt flags.

Parameters

- base – SI base address.
- txFrameIntrMask – IPV value to be mapped, bit x represents ring x.
- txThresIntrMask – The Rx BD ring index to be mapped, bit x represents ring x.

static inline void NETC_SICleanRxIntrFlags(ENETC_SI_Type *base, uint32_t rxIntrMask)

Clean the SI receive interrupt flags.

Parameters

- base – SI base address.
- rxIntrMask – Rx interrupt bit mask, bit x represents ring x.

void NETC_SIPsiEnableInterrupt(ENETC_SI_Type *base, uint32_t mask, bool enable)

PSI enables/disables specified interrupt.

Parameters

- base – SI base address.
- mask – The interrupt mask, refer to netc_psi_msg_flags_t which should be OR'd together.
- enable – Enable/Disable the interrupt.

static inline uint32_t NETC_SIPsiGetStatus(ENETC_SI_Type *base)

PSI gets interrupt event flag status.

Parameters

- base – SI base address.

Returns

The interrupt mask, refer to netc_psi_msg_flags_t which should be OR'd together.

static inline void NETC_SIPsiClearStatus(ENETC_SI_Type *base, uint32_t mask)

PSI clears interrupt event flag.

Parameters

- base – SI base address.
- mask – The interrupt mask, refer to netc_psi_msg_flags_t which should be OR'd together.

status_t NETC_SIPsiSendMsg(ENETC_SI_Type *base, uint16_t msg, *netc_vsi_number_t* vsi)

PSI sends message to specified VSI(s)

Parameters

- base – SI base address.
- msg – The message to be sent.
- vsi – The VSI number.

Returns

status_t

static inline bool NETC_SIPsiCheckTxBusy(ENETC_SI_Type *base, *netc_vsi_number_t* vsi)

PSI checks Tx busy flag which should be cleaned when VSI receive the message data.

Parameters

- base – SI base address.
- vsi – The VSI number.

Returns

The busy status of specified VSI.

status_t NETC_SIPsiSetRxBuffer(ENETC_SI_Type *base, *netc_vsi_number_t* vsi, uint64_t buffAddr)

PSI sets Rx buffer to receive message from specified VSI.

Note: The buffer memory size should be big enough for the message data from VSI

Parameters

- base – SI base address.
- vsi – The VSI number.
- buffAddr – The buffer address to store message data from VSI, must be 64 bytes aligned.

Returns

status_t

status_t NETC_SIPsiGetRxMsg(ENETC_SI_Type *base, *netc_vsi_number_t* vsi, *netc_psi_rx_msg_t* *msgInfo)

PSI gets Rx message from specified VSI.

Parameters

- base – SI base address.
- vsi – The VSI number.
- msgInfo – The Rx message information.

void NETC_SIVsiEnableInterrupt(ENETC_SI_Type *base, uint32_t mask, bool enable)

Enable VSI interrupt.

Parameters

- base – SI base address.
- mask – The interrupt mask, see netc_vsi_msg_flags_t which should be OR'd together.
- enable – Enable/Disable interrupt.

static inline uint32_t NETC_SIVsiGetStatus(ENETC_SI_Type *base)

Get VSI interrupt status.

Parameters

- base – SI base address.

Returns

A bitmask composed of netc_vsi_msg_flags_t enumerators OR'd together.

static inline void NETC_SIVsiClearStatus(ENETC_SI_Type *base, uint32_t mask)

Clear VSI interrupt status.

Parameters

- base – SI base address.
- mask – The interrupt mask, see netc_vsi_msg_flags_t which should be OR'd together.

status_t NETC_SIVsiSendMsg(ENETC_SI_Type *base, uint64_t msgAddr, uint32_t msgLen)

VSI sends message to PSI.

Parameters

- base – SI base address.
- msgAddr – Address to store message ready to be sent, must be 64 bytes aligned.
- msgLen – The message length, must be 32 bytes aligned.

Returns

status_t

void NETC_SIVsiCheckTxStatus(ENETC_SI_Type *base, *netc_vsi_msg_tx_status_t* *status)

Check VSI Tx status.

Parameters

- base – SI base address.
- status – The VSI Tx status structure.

status_t NETC_SIVsiReceiveMsg(ENETC_SI_Type *base, uint16_t *msg)

VSI receives message from PSI.

Parameters

- base – SI base address.
- msg – The message from PSI.

Returns

status_t

void NETC_SIGetDiscardStatistic(ENETC_SI_Type *base, *netc_si_discard_statistic_t* *statistic)

Get the discard statistic from SI layer.

Parameters

- base – SI base address.
- statistic – The statistic data.

void NETC_SIGetTrafficStatistic(ENETC_SI_Type *base, *netc_si_traffic_statistic_t* *statistic)

Get the traffic statistic from SI layer.

Parameters

- base – SI base address.
- statistic – The statistic data.

NETC_VLAN_PCP_DEI_VALUE(pcp, dei)

Macro to cover VLAN PCP, DEI value to internal used pcpDei value.

struct _netc_hw_enetc_si_config

#include <fsl_netc.h> Station Interface configuration.

Public Members

uint32_t bandWeight

Station interface traffic class bandwidth weight

uint32_t vlanCtrl

VLAN Ethertypes can be inserted by the SI driver, set with OR of netc_hw_enetc_si_vlan_type.

uint32_t antiSpoofEnable

Anti-spoofing enable

uint32_t vlanInsertEnable

Software SI-based VLAN Insertion enable, avtive when enSIBaseVlan is true

uint32_t vlanExtractEnable

SI-based VLAN removed from frame enable, avtive when enSIBaseVlan is true

uint32_t sourcePruneEnable

Source pruning enable

uint32_t rxRingUse

Number of Rx Rings to be used, when enable Rx ring group, this equal to the sum of all Rx group rings.

uint32_t txRingUse

Number of Tx Rings to be used, note that when SI is Switch management ENETC SI, the number not include Tx ring 0.

uint32_t valnToIpvEnable

Enable the VLAN PCP/DEI value (use NETC_VLAN_PCP_DEI_VALUE marco) to internal priority value mapping.

uint32_t rxBdrGroupNum

Rx BD ring group number, range in 0 ~ 2.

uint32_t ringPerBdrGroup

The ring number in every Rx BD ring group, range in 1 ~ 8, active when rxBdrGroupNum not equal zero.

netc_hw_enetc_si_rxr_group defaultRxBdrGroup

The selected Rx BD ring group, active when rxBdrGroupNum not equal zero.

uint8_t vlanToIpvMap[16]

Frame VLAN pcp|dei to IPV mapping, active when valnToIpvEnable is true.

uint8_t ipvToRingMap[8]

BD ring used within the default Rx BD ring group for IPV n, active when rxBdrGroupNum not equal zero.

uint8_t vsiTcToTC[8]

Maps the VSI traffic class to transmit traffic class, done after the ENETC txPrio to TC mapping, only available for VSI.

bool enSIBaseVlan

Enable use SI-based VLAN information.

netc_enetc_vlan_tag_t siBaseVlan

SI-based VLAN information, active when enSIBaseVlan is true.

struct __netc_si_l2mf_config

#include <fsl_netc.h> L2 Mac Filter Configuration for SI.

Public Members

bool macUCPromis

Enable/Disable MAC unicast promiscuous.

bool macMCPromis

Enable/Disable MAC multicast promiscuous.

bool rejectUC

Reject Unicast.

bool rejectMC

Reject Multicast.

bool rejectBC

Reject Broadcast.

struct __netc_si_l2vf_config

#include <fsl_netc.h> L2 VLAN Filter Configuration for SI.

Public Members

bool acceptUntagged

Accept/Reject untagged frame.

bool enPromis
Enable/Disable VLAN promiscuous.

bool useOuterVlanTag
Use outer/inner VLAN tag for filtering.

struct __netc__si__discard__statistic
#include <fsl_netc.h> SI frame drop statistic struct.

Public Members

uint32_t programError
Due to programming error (non-existing BD ring or non-existing group, or SI disabled or BD ring disabled).

uint32_t busError
Due to system bus error.

uint32_t lackBD[14]
Due to lack of Rx BDs available.

struct __netc__si__traffic__statistic
#include <fsl_netc.h> SI traffic statistic struct.

struct __netc__si__config
#include <fsl_netc.h> SI Configuration.

Public Members

uint32_t tcBWWeight
SI traffic class bandwidth weight.

union __netc__tx__bd
#include <fsl_netc.h> Transmit Buffer Descriptor format.

A union type cover the BD used as Standard/Extended/WriteBack format.

Public Members

struct *_netc_tx_bd* standard

struct *_netc_tx_bd* ext

struct *_netc_tx_bd* writeback

uint64_t dword[2]

union __netc__rx__bd
#include <fsl_netc.h> Receive Buffer Descriptor format.

Public Members

struct *_netc_rx_bd* standard

struct *_netc_rx_bd* writeback

struct *_netc_rx_bd* ext

struct __netc__tx__bdr__config
#include <fsl_netc.h> Configuration for the SI Tx Buffer Descriptor Ring Configuration.

Public Members

uint32_t len
: Size of BD ring which shall be multiple of 8 BD.

netc_tx_bd_t *bdArray
: BDR base address which shall be 128 bytes aligned.

netc_tx_frame_info_t *dirtyArray
: Tx cleanup ring.

bool enIntr
: Enable/Disable completion interrupt.

bool enThresIntr
: Enable/Disable threshold interrupt.

bool enCoalIntr
: Enable/Disable interrupt coalescing.

uint32_t intrThreshold
: Interrupt coalescing packet threshold.

uint32_t intrTimerThres
: Interrupt coalescing timer threshold, specified in NETC clock cycles.

uint8_t msixEntryIdx
: MSIX entry index of Tx ring interrupt.

bool isVlanInsert
: Enable/Disable VLAN insert offload.

bool isUserCRC
: Enable/Disable user provided the CRC32 - FCS at end of frame.

uint8_t wrrWeight
: Weight used for arbitration when rings have same priority.

uint8_t priority
: Priority of the Tx BDR.

struct __netc_tx_bdr
: *#include <fsl_netc.h>* Transmit BD ring handler data structure.

Public Members

netc_tx_bd_t *bdBase
: Tx BDR base address.

netc_tx_frame_info_t *dirtyBase
: Tx cleanup ring base address.

uint16_t producerIndex
: Current index for transmit.

uint16_t cleanIndex
: Current index for tx cleaning.

uint32_t len
: Length of this BD ring.

struct __netc_rx_bdr_config
: *#include <fsl_netc.h>* Configuration for the SI Rx Buffer Descriptor Ring Configuration.

Public Members

bool extendDescEn
: False - Use 16Bytes standard BD. True - Use 32Bytes extended BD.

netc_rx_bd_t *bdArray
: BD ring base address which shall be 128 bytes aligned.

uint32_t len
: BD ring length in the unit of 16Bytes standard BD. Shall be multiple of 8/16 for standard/exteneded BD.

uint64_t *buffAddrArray
: Rx buffers array with BD length(half of BD length if use exteneded BD).

uint16_t buffSize
: Size of all Rx buffers in this BD ring.

bool enThresIntr
: Enable/Disable threshold interrupt.

bool enCoalIntr
: Enable/Disable interrupt coalescing.

uint32_t intrThreshold
: Interrupt coalescing packet threshold.

uint32_t intrTimerThres
: Interrupt coalescing timer threshold, specified in NETC clock cycles.

uint8_t msixEntryIdx
: MSIX entry index of Rx ring interrupt.

bool disVlanPresent
: Disable/Enable VLAN in BD.

bool enVlanExtract
: Enable/Disable VLAN extract.

bool isKeepCRC
: Whether user provided the CRC32 - FCS at end of frame.

bool congestionMode
: False - lossy. True - lossless.

bool enHeaderAlign
: Enable/disable +2B alignment to frame.

struct __netc__rx__bdr
: *#include <fsl_netc.h>* Receive BD ring handler data structure.

Public Members

netc_rx_bd_t *bdBase
: Rx BDR base address.

bool extendDesc
: Use extended buffer descriptor.

uint16_t index
: Current index for read.

uint32_t len
: Length of this BD ring, unit of 16Bytes standard BD.

uint64_t *buffArray
: Rx buffers array of this ring.

uint32_t buffSize
: Rx buffers size for all BDs in this ring.

struct __netc_bdr_config
: *#include <fsl_netc.h>* Configuration for the buffer descriptors ring.

Public Members

netc_rx_bdr_config_t *rxBdrConfig
: Receive buffer ring configuration array.

netc_tx_bdr_config_t *txBdrConfig
: Transmit buffer ring configuration array.

struct __netc_psi_rx_msg
: *#include <fsl_netc.h>* PSI receive message information.

Public Members

uint8_t *msgBuff
: The buffer address application set before receiving message.

uint32_t msgLen
: Received message length.

struct __netc_vsi_msg_tx_status
: *#include <fsl_netc.h>* VSI message transmit status.

Public Members

bool txBusy
: The VSI Tx busy flag, become idle when the PSI receive and clear the related status.

bool isTxErr
: Tx error flag.

uint16_t msgCode
: The error code or user-defined content.

struct standard

Public Members

uint64_t addr
: Address of the buffer. Little Endian.

uint16_t bufLen
: Length of buffer specifying effective number of bytes.

uint16_t frameLen
: Length of Frame.

uint32_t flags
: Flags qualified setting.

uint32_t enableInterrupt
: Whether enable interrupt on complete of BD.

uint32_t isExtended
: Extended BD format flag.

uint32_t isFinal
: Final BD flag.

struct ext

Public Members

uint32_t timestamp
: IEEE1588 PTP one-step timestamp.

uint32_t ___pad0___
: Ignore 2-bit MSB.

uint16_t tpid
: VLAN TPID type, see netc_vlan_tpid_select_t.

uint16_t vid
: VLAN ID.

uint16_t dei
: VLAN DEI.

uint16_t pcp
: VLAN PCP.

uint8_t eFlags
: Tx extension flags.

uint8_t isFinal
: Final BD flag.

struct writeback

Public Members

uint32_t timestamp
: Timestamp write back.

uint32_t status
: Status.

uint32_t written
: Write-back flag.

struct standard

Public Members

uint64_t addr
: Software write address.

struct writeback

Public Members

uint16_t internetChecksum
: Internet Checksum.

uint16_t parserSummary
: Parser Summary.

uint16_t bufLen
: Length of received buffer.

uint16_t vid
: VLAN ID.

uint16_t dei
: VLAN DEI.

uint16_t pcp
: VLAN PCP.

uint8_t tpid
: VLAN TPID.

uint8_t hr
: Host Reason.

uint8_t flags
: Rx information flags.

uint8_t error
: Rx error code.

uint8_t isReady
: Received data ready flag.

uint8_t isFinal
: Final BD flag.

union ___unnamed193___

Public Members

struct __netc_rx_bd

uint32_t rssHashSwt
: RSS hash while not used as switch management port.

struct ___unnamed195___

Public Members

uint32_t srcPort
: Source port received from switch management port.

uint32_t rssHash
: RSS Hash high field value.

struct ext

Public Members

uint32_t timestamp
Rx Timestamp.

2.79 Hardware Switch

enum __netc_swt_port_bitmap
The switch port bitmap.

Values:

enumerator kNETC_SWTPort0Bit
Switch port0 bitmap

enumerator kNETC_SWTPort1Bit
Switch port1 bitmap

enumerator kNETC_SWTPort2Bit
Switch port2 bitmap

enumerator kNETC_SWTPort3Bit
Switch port3 bitmap

enumerator kNETC_SWTPort4Bit
Switch port4 (internal port) bitmap

enum __netc_swt_imr_dest_port
The switch ingress mirror destination port.

Values:

enumerator kNETC_SWTPort0
Switch port0

enumerator kNETC_SWTPort1
Switch port1

enumerator kNETC_SWTPort2
Switch port2

enumerator kNETC_SWTPort3
Switch port3

enumerator kNETC_SWTPort4
Switch port4

enumerator kNETC_SWTMPort
Switch management port

enum __netc_swt_mac_forward_mode
The switch MAC forwarding options.

Values:

enumerator kNETC_NoFDBLookUp
No FDB lookup is performed, the frame is flooded.

enumerator kNETC_FDBLookUpWithFlood
FDB lookup is performed, and if there is no match, the frame is flooded to the port bitmap in VLAN filter entry.

enumerator kNETC_FDBLookUpWithDiscard
FDB lookup is performed, and if there is no match, the frame is discarded.

enum _netc_swt_mac_learn_mode
The switch MAC learning options.

Values:

enumerator kNETC_DisableMACLearn
Disable MAC learning. SMAC FDB lookup is by-passed.

enumerator kNETC_HardwareMACLearn
Hardware MAC learning is enabled.

enumerator kNETC_SeSoftwareMACLearn
Software MAC learning secure. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is redirected to the switch management port.

enumerator kNETC_UnseSoftwareMACLearn
Software MAC learning unsecure. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is copied to the switch management port.

enumerator kNETC_DisableMACLearnWithSMAC
Disable MAC learning with SMAC validation. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is discarded.

enum _netc_swt_port_tx_vlan_act
Switch transmit Bridge Port VLAN Tag Action.

Values:

enumerator kNETC_NoTxVlanModify
No egress VLAN modification performed

enumerator kNETC_TxDelOuterVlan
Delete outer VLAN tag

enumerator kNETC_TxReplOuterVlanVid
Replace outer VLAN tag's VID with 0; frame to be transmitted as a priority tag frame

enum _netc_swt_port_stg_mode
Switch port spanning tree group work mode.

Values:

enumerator kNETC_DiscardFrame
Tx or RX Frames on this port with current spanning tree group ID will be discarded

enumerator kNETC_LearnWithoutFowrad
RX Frames on this port with current spanning tree group ID will do Learn SMAC, but do not forward, Tx Frame will be discarded

enumerator kNETC_ForwardFrame
RX Frames on this port with current spanning tree group ID will do both MAC learning and forwarding, , Tx Frame will be forwarded.

typedef enum *_netc_swt_port_bitmap* netc_swt_port_bitmap_t
The switch port bitmap.

typedef enum *_netc_swt_imr_dest_port* netc_swt_imr_dest_port_t
The switch ingress mirror destination port.

typedef enum *_netc_swt_mac_forward_mode* netc_swt_mac_forward_mode_t
The switch MAC forwarding options.

typedef enum *_netc_swt_mac_learn_mode* netc_swt_mac_learn_mode_t
The switch MAC learning options.

typedef enum *_netc_swt_port_tx_vlan_act* netc_swt_port_tx_vlan_act_t
Switch transmit Bridge Port VLAN Tag Action.

typedef enum *_netc_swt_port_stg_mode* netc_swt_port_stg_mode_t
Switch port spanning tree group work mode.

typedef struct *_etc_swt_imr_config* netc_swt_imr_config_t
Switch Ingress mirror destination config.

typedef struct *_netc_swt_port_config* netc_swt_port_bridge_config_t
Switch port bridge configuration.

typedef struct *_netc_swt_port_fm_config* netc_swt_port_fm_config_t
Switch Port level Frame Modification configuration (PPCPDEIMR and PQOSMR[QVMP])

typedef struct *_netc_swt_default_vlan_filter* netc_swt_default_vlan_filter_t
Switch VLAN filter hash table default entry configuration, which determines the default entry when not found in VLAN filter lookup.

typedef struct *_netc_swt_bridge_config* netc_swt_bridge_config_t
Bridge config.

typedef struct *_netc_swt_psfp_config* netc_swt_psfp_config_t
Switch PSFP configuration.

typedef struct *_netc_qos_classify_config* netc_swt_qos_classify_config_t
Switch Qos Classification configuration (include two profiles)

typedef struct *_netc_swt_qos_to_vlan_config* netc_swt_qos_to_vlan_config_t
Switch QoS to PCP mapping and PCP to PCP mapping configuration when egress packet modification the VLAN tag.

typedef struct *_netc_switch_inuse_fdb_statistic* netc_switch_inuse_fdb_statistic_t
Switch static/dynamic FDB entries in-use statistic.

typedef struct *_netc_swt_port_sr_config* netc_swt_port_sr_config_t
Port seamless redundancy configuration.

struct _etc_swt_imr_config
#include <fsl_netc.h> Switch Ingress mirror destination config.

Public Members

bool enMirror
Enable ingress mirroring

netc_swt_imr_dest_port_t destPort
Port where ingress mirrored frames are sent

uint8_t dr
Mirrored packet's DR (drop resilience)

uint8_t ipv
Mirrored packet's IPV (internal priority value)

uint8_t efmLengthChange
Egress Frame Modification Frame Length change in 2s complement notation, Vaild if efmEntryID is noy null

uint16_t efmEntryID

Egress Frame Modification Entry Id, note 0xFFFF is a Null Frame Modification Entry, Only applicable if destPort != kNETC_SWTMPort

struct __netc_swt_port_config

#include <fsl_netc.h> Switch port bridge configuration.

Public Members

netc_swt_port_tx_vlan_act_t txVlanAction

Only applies for the frame outer VLAN tag's VID is equal to the port default VID

bool isRxVlanAware

Receive VLAN Aware Mode

bool acceptUntag

Accept untagged frame

bool acceptPriorityTag

Accept priority tagged frame (VID = 0)

bool acceptSingleTag

Accept single tagged frame

bool acceptDoubleTag

Accept double tagged frame (ounter and inner)

bool enSrcPortPrun

Enable/Disable received frame be transmitted to same port it was received

bool enMacStationMove

Enable/Disable received frame which ingress port not match the FDB entry Destination Port Bitmap

bool enBcastStormCtrl

Enable/Disable Storm control for broadcast frames

bool enMcastStormCtrl

Enable/Disable Storm control for multicast frames

bool enUnMcastStormCtrl

Enable/Disable Storm control for unknown multicast frames

bool enUnUcastStormCtrl

Enable/Disable Storm control for unknown unicast frames

uint32_t bcastRpEntryID

Broadcast rate policer entry ID. Valid if enBroadStormCtrl = true

uint32_t mcastEntryID

Known multicast rate policer entry ID. Valid if enBroadStormCtrl = true

uint32_t unMcastRpEntryID

Unknown multicast policer entry ID. Valid if enUnMultiStormCtrl = true

uint32_t unUcastRpEntryID

Unknown unicast rate policer entry ID. Valid if enUnUniStormCtrl = true

uint16_t maxDynaFDBEntry

The maximium number of dynamic entries in the FDB table, 0 means no limit

struct __netc_swt_port_fm_config

#include <fsl_netc.h> Switch Port level Frame Modification configuration (PPCPDEIMR and PQOSMR[QVMP])

Public Members

bool ignoreFMMiscfg

Enable/Disable ignore the Frame Modification Misconfiguration Action

bool enEgressPcpMap

Enable egress frame modification of outer VLAN tag's PCP value is mapped to a new value based on egressPcpMap, used for Frame Modification VLAN Outer PCP action

bool enIngressPcpMap

Enable ingress frame modification of outer VLAN tag's PCP value is mapped to a new value based on egressPcpMap, used for Frame Modification VLAN Outer PCP action

bool enUpdateVlanDei

Enable update DR value in the outer VLAN based on DEnDEI field, used for egress Frame Modification Outer DEI action

uint8_t drToDeiMap

Mapping of internal QoS's DR value n to VLAN DEI, The 4 bits correspond to the DR3 ~ DR0, and 1 means DRn mapping to DEI 1, 0 means DRn mapping to DEI 0

uint8_t egressPcpMap

Egress PCP to PCP Mapping Profile instance, active when enEgressPcpMap is true

uint8_t ingressPcpMap

Ingress PCP to PCP Mapping Profile instance, active when enIngressPcpMap is true

uint8_t qosVlanMap

Transmit QoS to VLAN PCP Mapping Profile index, used for egress Frame Modification VLAN Add/Replace Action

struct __netc_swt_default_vlan_filter

#include <fsl_netc.h> Switch VLAN filter hash table default entry configuration, which determines the default entry when not found in VLAN filter lookup.

Public Members

bool enIPMFlood

Enable IP Multicast Flooding

bool enIPMFilter

Enable IP Multicast Filtering

uint8_t stgID

Spanning Tree Group Member ID, range in 0 ~ 15

uint8_t portMembership

The bit 0 ~ 4 correspond to the 5 ports, When bit set (0b1), means the port is a member of this VLAN. Port membership is used for source/destination pruning

bool enUseFilterID

Enable use the specified filterID as FID, otherwise will use the frame VID

uint16_t filterID

Used as a key value to do FDB table and the L2 IPV4 Multicast Filter table lookup. Valid if enUseFilterID is true

netc_swt_mac_forward_mode_t mfo
MAC forwarding options

netc_swt_mac_learn_mode_t mlo
MAC learning options

uint16_t baseETEID
Base Egress Treatment Entry ID

uint8_t etaPortBitmap
Egress Treatment Applicability Port. Valid if baseETEID is not null.

struct _netc_swt_bridge_config
#include <fsl_netc.h> Bridge config.

Public Members

netc_swt_default_vlan_filter_t dVFCfg
Default VLAN filter entry configuration when not found in VLAN filter lookup

struct _netc_swt_psfp_config
#include <fsl_netc.h> Switch PSFP configuration.

Public Members

netc_isi_kc_rule_t kcRule[4]
Key construction rules

struct _netc_qos_classify_config
#include <fsl_netc.h> Switch Qos Classification configuration (include two profiles)

struct _netc_swt_qos_to_vlan_config
#include <fsl_netc.h> Switch QoS to PCP mapping and PCP to PCP mapping configuration when egress packet modification the VLAN tag.

struct fsl_netc
#include <fsl_netc.h>

Public Members

uint8_t qos[32]
Index is created from IPV (3 bits) + DR (2 bits) field. Value is the mapped PCP for VLAN tag.

uint8_t pcp[8]
Index is created from outer PCP (3 bits) field. Value is the mapped PCP for VLAN tag.

struct _netc_switch_inuse_fdb_statistic
#include <fsl_netc.h> Switch static/dynamic FDB entries in-use statistic.

Public Members

uint16_t camEntries
Number of FDB entries in-use in the CAM.

uint16_t staticEntries
Number of static FDB entries in-use (both hash-based and CAM-based entries).

uint16_t dynamicEntries
Number of dynamic FDB entries in-use (hash-based and CAM-based entries).

uint16_t dynamicEntriesHWM
High water mark of dynamic entries in-use in the FDB table.

struct __netc_swt_port_sr_config
#include <fsl_netc.h> Port seamless redundancy configuration.

struct defaultVlan

Public Members

uint32_t vid
Vlan Identifier.

uint32_t dei
Drop eligible indicator

uint32_t pcp
Priority code point.

uint32_t tpid
Tag protocol identifier, 0 = Standard C-VLAN 0x8100, 1 = Standard S-VLAN 0x88A8.

2.80 Hardware Table Access Functions

enum __netc_tb_index
Table index.

Values:

enumerator kNETC_TGSTable
Time Gate Scheduling table index

enumerator kNETC_RPTable
Rate Policer table index

enumerator kNETC_IPFTable
Ingress Port filter table index

enumerator kNETC_FDBTable
FDB table index

enumerator kNETC_L2MCFTable
L2 IPV4 Multicast Filter table index

enumerator kNETC_VFTable
VLAN Filter table index

enumerator kNETC_ECQTable
ETM Class Queue table index

enumerator kNETC_ECSTable
ETM Class Scheduler table index

enumerator kNETC_ISITable
Ingress Stream Identification table index

enumerator kNETC_ISTable
Ingress Stream table index

enumerator kNETC_ISFTable
Ingress Stream Filter table index

enumerator kNETC_ETTable
Egress Treatment table index

enumerator kNETC_ISGTable
Ingress Sequence Generation table index

enumerator kNETC_ESRTable
Egress Sequence Recovery table index

enumerator kNETC_SGITable
Stream Gate Instance table index

enumerator kNETC_SGCLTable
Stream Gate Control List table index

enumerator kNETC_ISCTable
Ingress Stream Count table index

enumerator kNETC_ECTable
Egress Count table index

enumerator kNETC_FMTable
Frame Modification table index

enumerator kNETC_BPTable
Buffer Pool table index

enumerator kNETC_SBPTable
Shared Buffer Pool table index

enumerator kNETC_ECGTable
ETM Class Group table index

enumerator kNETC_FMDTable
Frame Modification Data table index

enum _netc_tb_cmd
Table management command operations.

Values:

enumerator kNETC_DeleteEntry
Delete operation

enumerator kNETC_UpdateEntry
Update operation

enumerator kNETC_QueryEntry
Query operation

enumerator kNETC_QueryAndDeleteEntry
Query operation followed by a delete operation

enumerator kNETC_QueryAndUpdateEntry
Query operation followed by a update operation

enumerator kNETC_AddEntry
Add operation

enumerator kNETC_AddOrUpdateEntry
If the entry exists, is update operation, if not exist, is the Add operation

enumerator kNETC_AddAndQueryEntry
Add operation followed by a query operation

enumerator kNETC_AddQueryAndUpdateEntry
Add operation followed by a query operation, Then, if the entry existed prior to the Add operation of this command, the Update operation will be performed.

enum _netc_tb_access_mode
Table Access Method.

Values:

enumerator kNETC_EntryIDMatch
Entry ID Match

enumerator kNETC_ExactKeyMatch
Exact Match Key Element Match

enumerator kNETC_Search
Search with search criteria

enumerator kNETC_TernaryKeyMatch
Ternary Match Key Element Match

enum _netc_cbd_version
NTMP version.

Values:

enumerator kNETC_NtmpV1_0
NTMP Version 1.0

enumerator kNETC_NtmpV2_0
NTMP Version 2.0

enum _netc_cmd_error
Table command response error status.

Values:

enumerator kNETC_FormatError
Format error : 1. Illegal class or command. 2. Invalid SF bit setting. 3. LENGTH is zero for long format. 4. LENGTH is too small for buffer size.

enumerator kNETC_SizeError
Size error : 1. Invalid table index, out of range. 2. Table overflow, no additional entries available.

enumerator kNETC_AccessError
Access violation error, the entity is not allowed to perform the task requested

enumerator kNETC_ClassError
Class specific error

enumerator kNETC_IntegrityError
Integrity error, the command did not execute due to a data integrity error (ECC on internal memory or AXI read/write error)

enumerator kNETC_InvTableID
Invalid table ID

enumerator kNETC_InvAccMethod
Invalid Access method

enumerator kNETC_TableIdxOutRange
Table index out of range

enumerator kNETC_DBNotEnough
Request data buffer size or response data buffer size is not sufficient

enumerator kNETC_InvCmd
Invalid command

enumerator kNETC_ReqDBError
Request Data buffer error

enumerator kNETC_MultiBitError
Multi-bit ECC or parity error observed during command processing

enumerator kNETC_HashEntryLimit
Exceeded hash entry limit

enumerator kNETC_HashChainLimit
Exceeded maximum hash collision chain limit and the CAM if present is full

enumerator kNETC_InvHWGenEntryID
Invalid ENTRY_ID for ENTRY_ID generated by hardware

enumerator kNETC_SrchResDBNotEnough
Search command filled the response data buffer before completing the command

enumerator kNETC_CmdIdxTableWithITM
Command for index table before OSR[ITM_STATE]=0

enumerator kNETC_InvQueryAction
Invalid Query action

enumerator kNETC_InvTableAccPrivilege
Invalid table access privilege

enumerator kNETC_ReadSysBusErr
System Bus Read Error

enumerator kNETC_WriteSysBusErr
System Bus Write Error

enumerator kNETC_ClientErr
Client encountered a fault

enumerator kNETC_TGSCmdIssue
Command issued when time gating function is disabled for the port.

enumerator kNETC_TGSUpdateExistList
Update action attempted on an existing admin gate control list. (should delete admin gate control list first before creating a new admin list)

enumerator kNETC_TGSUpdateOverLength

Update action attempted exceeds TGSTCAPR[MAX_GCL_LEN]

enumerator kNETC_TGSUpdateOverSize

Update action attempted exceeds TGSTCAPR[NUM_WORDS].

enumerator kNETC_TGSEntryNotEnough

Insufficient resources to perform the requested operation (not enough free time gate list entries)

enumerator kNETC_TGSUpdateNSList

Update action attempted with ADMIN_CYCLE_TIME, ADMIN_TIME_INTERVAL_GE_i or truncated ADMIN_TIME_INTERVAL_GE_n due ADMIN_CYCLE_TIME specified is not sufficient to transmit 64 byte of frame data + header overhead.

enumerator kNETC_TGSUpdateEarlierStartTime

Update action attempted with ADMIN_BASE_TIME specified s more than one second in the past from tcs advance time.

enumerator kNETC_TGSUpdateOverflowCycle

Update action attempted with ADMIN_CYCLE_TIME + ADMIN_CYCLE_TIME_EXT is greater than 2^32-1.

enumerator kNETC_TGSQueryBeforeListActive

Query action issued when config change occurred. Retry query.

enumerator kNETC_TGSUpdateInvGateValue

Update action attempted with ADMIN_HR_CB_GE_i set to an invalid value

enumerator kNETC_RPSDUTypeOutRange

SDU_TYPE specified in entry CFGE_DATA is out of range

enumerator kNETC_IPFInvHR

HR value not valid. Only checked if command issued from the Switch and FLTFA=0x2 or FLTFA=0x3

enumerator kNETC_IPFEntryNotFit

Entry being added does not fit in table

enumerator kNETC_IPFWithoutSTSE

CFGE_DATA update without STSE_DATA update

enumerator kNETC_IPFInvRPP

RPR set to a reserved value. Only checked if FLTA=0x2.

enumerator kNETC_IPFFLTATGTOutRange

FLTA_TGT is outside valid range and not NULL. Only checked if FLTA>0x0

enumerator kNETC_IPFInvSwtFLTA

FLTA=0x3 when command issued from the Switch.

enumerator kNETC_IPFInvEnetcFLTA

FLTFA>0x1 when command issued from an ENETC PF.

enumerator kNETC_FDBReachPortLimit

Failed to add or update and entry because the Port BPCR[DYN_LIMIT] has been reached

enumerator kNETC_FDBReachSwtLimit

Failed to add entry because the Switch FDBHTMCR[DYN_LIMIT] has been reached.

enumerator kNETC_FDBInvEPORT

EPORT value not valid. Only checked if (OETEID=0x1 OR CTD=0x1)

enumerator kNETC_FDBETEIDOutRange

ET_EID is out of range and not NULL. Only checked if OETEID>0x0

enumerator kNETC_FDBParityErr

Parity error encountered when adding guaranteed entry

enumerator kNETC_L2MCFInvEPORT

EPORT value not valid. Only checked if (OETEID=0x1 OR CTD=0x1)

enumerator kNETC_L2MCFETEIDOutRange

ET_EID is not NULL or within the valid range. Only checked if OETEID>0x0.

enumerator kNETC_L2MCFInvKEYTYPE

KEY_TYPE value not valid

enumerator kNETC_VFBASEETEIDOutRange

BASE_ET_EID is out of range or MLO is not valid.

enumerator kNETC_ECQCQ2CGMAPOutRange

CQ2CG_MAP value out-of-range in update command.

enumerator kNETC_ISIPortIDOutRange

Port ID specified in KEYE_DATA is out of range.

enumerator kNETC_ISIInvISEID

IS_EID in invalid.

enumerator kNETC_ISInvOpt

Option specified in one or more of the following fields is not valid – FA, CTD or ISQA, SDU_TYPE.

enumerator kNETC_ISInvID

One or more of following : 1. Entry IDs are not in valid range or Entry ID is not Null. 2. Check valid ranges specified for these Entry IDs in Ingress Stream table entry – RP_EID, SGI_EID, ISQ_EID, ET_EID or EPORT. 3. ET_EID is checked if (FA =010b .. 101b) & (OETEID!=0). 4. EPORT is checked if (FA = 010b .. 101b) & (OETEID= 0x1 OR CTD= 0x1). 5. HR is chked if FA = 001b, 100b, or 101b. HR specified cannot be 0x0

enumerator kNETC_ISInvFMEID

FM_EID format or index is out of range : 1. FM_EID format option type is invalid. 2. FM_EID format is option 1 and the Index is out of range and not Null, or FM_EID format is option 2 and VUDA or SQTA is out of range.

enumerator kNETC_ISFInvISEID

IS_EID in KEYE_DATA is invalid.

enumerator kNETC_ISFInvCFGE

Any of the following in CFGE_DATA is invalid : 1. One or more of following Entry IDs are not in valid range or Entry ID specified is not Null. Checks are performed for following Entry IDs CFGE DATA – RP_EID, SGI_EID, ISC_EID. 2. SDU_TYPE is invalid

enumerator kNETC_ETInvOpt

Command option specified is invalid or not supported. ESQA is not 00 or 10 (others are reserved), or ECA > 1 (reserved).

enumerator kNETC_ETInvFMEID

FM_EID format or index is out of range. Check performed is as follows : 1. EFM_EID format option type is invalid, or EFM_EID format is option 1 and the Index is out of range and not Null, or EFM_EID format is option 2 and VUDA or SQTA is out of range . 2. the Egress Counter Table index EC_EID is out of range. 3. The Egress Sequence Actions Target Entry ID ESQA_TGT_EID is out of range

enumerator kNETC_ISGInvQSTAG

SQ_TAG specified is not valid

enumerator kNETC_SGISGCLEIDOutRange

SGCL_EID specified in out of range for Add or Update operation.

enumerator kNETC_SGIInvSDUTYPE

SDU_TYPE is specified is invalid for Add or Update operation.

enumerator kNETC_SGISGCLEIDNotAlloc

Either the SGCL_EID specified as admin gate control list in Add or Update operation has not been allocated or SGCL_EID is not the first entry in gate control list or the reference count in SGCL entry is not 0.

enumerator kNETC_SGIInvSGCLEID

SGCL_EID specified for Update operation is in invalid.

enumerator kNETC_SGIInvADMINBASETIME

ADMIN_BASE_TIME specified for Add or Update operation is more than 2^30ns in the past.

enumerator kNETC_SGIInvCYCLETIME

Cumulated time value of CYCLE_TIME in Stream gate Control list plus CYCLE_TIME_EXT specified in Add or Update operation is >=2^30ns or CYCLE_TIME specified is 0.

enumerator kNETC_SGCLOverLength

Number words required for the LIST_LENGTH specified for the Add operation exceeds the number of words allocated for SGCL table

enumerator kNETC_SGCLTimeIntervalZero

TIME_INTERVAL_GE_N specified in Add operation is 0. Note that upper 2 bits of TIME_INTERVAL_GE_N are ignored, TIME_INTERVAL_GE_N[29:0] must not be 0.

enumerator kNETC_SGCLTimeIntervalOverflow

Cumulated time value of TIME_INTERVAL_GE_N[29:0] for the gate list specified in Add operation is >= 2^30ns.

enumerator kNETC_FMInvEMEID

FM_EID format is invalid

enumerator kNETC_FMOptOutRange

Following fields specified are out of range - MAC_HDR_ACT, VLAN_HDR_ACT, SQT_ACT, OUTER_PCP_DEI_ACT, PLD_ACT.

enumerator kNETC_FMFMDOutRange

FMD_EID,FMD_BYTES specified is out of range. When FMD_EID is not set to Null, valid range is FMD_EID[15:0]*24 + FMD_BYTES <= (FMDITCAPR[NUM_WORDS]*24).

enumerator kNETC_BPSBPEIDOutRange

SBP_EN is 1 and SBP_EID value is out-of-range in update command

enum _netc_fm_vlan_ud_act

Frame Modification VLAN Update/Delete Action.

Note: Misconfiguration error if replace or delete action is specified and if VLAN tag is not present in frame.

Values:

enumerator kNETC_NoUDVlanAction

No Update/Delete VLAN action

enumerator kNETC_ReplVlanPcpAndDei

Replace outer VLAN's PCP/DEI based on the port's PPCPDEIMR. The tag's original VID and TPID are preserved

enumerator kNETC_DelVlan

Delete outer VLAN Tag

enum _netc_fm_sqt_act

Frame Modification Sequence Tag Action.

Note: Must be set to 000b for Ingress frame modification, otherwise misconfiguration error..

Values:

enumerator kNETC_NoSqtAction

No SQT action

enumerator kNETC_ReomveRTag

Remove R-TAG/draft 2.0 R-TAG/HSR tag, If R-TAG/HSR tag not present, misconfiguration error.

enum _netc_fm_vlan_ar_act

Frame Modification VLAN Add/Replace Action.

Note: For ingress frame modificaion with 00b or 01b, use the ingress port to select PCP and DEI from the Bridge port default VLAN register (BPDVR). For egress frame modification with 00b or 01b, use the internal QoS associated with the frame (IPV, DR) to access the QoS to PCP mapping profile (PQOSMR[QVMP] , QOSVLANMPaR0/1/2/3) to set the new PCP value. Use internal DR associated with frame to access the DR to DEI mapping profile (PPCPDEIMR[DRnDEI]) to set the new DEI value.

Values:

enumerator kNETC_AddCVlanPcpAndDei

Add outer VLAN with VID and PCP/DEI updated as described above. TPID=0x8100

enumerator kNETC_AddSVlanPcpAndDei

Add outer VLAN with VID and PCP/DEI updated as described above. TPID=0x88A8

enumerator kNETC_ReplVidOnly

Replace VLAN with VID. The tag's original PCP, DEI and TPID are preserved

enumerator kNETC_ReplVidPcpAndDei

Replace VLAN with VID and PCP/DEI updated by port's PPCPDEIMR. The tag's original TPID is preserved

enum _netc_tb_eteid_access_mode

Define FDB/L2MCF/IS table entry access the primary Egress Treatment table entry group mode.

Note: The FDB/L2 IPv4 Multicast filter table has precedence over any assignment made via the Ingress Stream table. For Mulit port mode, the index to access the Egress Treatment table is computed by adding an offset to the base index of the Egress Treatment group. That offset is derived from the applicability bitmap as follows: starting from the lowest

significant bit of the bitmap, the first encountered bit set to 1, corresponds to offset 0, and so on. This continues till the destination port location in the bitmap is reached

Values:

enumerator kNETC_NoETAccess
No Egress Treatment table access

enumerator kNETC_SinglePortETAccess
Only frame sent to a special port (define in ePort) can access a single Egress Treatment table entry, the applicability bitmap specified by FDB/L2MCF/IS ePort field

enumerator kNETC_MulitPortPackedETAccess
Only frames sent to a special set of ports (ports set to 1 in ePortBitmap) can access the Egress Treatment table, the applicability bitmap = IS ePortBitmap field or FDB/L2MCF portBitmap field

enumerator kNETC_MulitPortAbsETAccess
Frames sent to all of ports can access the Egress Treatment table, means the applicability bitmap is set with 1 for all ports

enum _netc_tb_ipf_update_action
Ingress Port Filter Table Update Actions.

Values:

enumerator kNETC_IPFCfgEUpdate
Configuration Element Update

enumerator kNETC_IPFStsEUpdate
Statistics Element Update

enum _netc_tb_ipf_attr_mask
Ingress Port Filter frame attribute mask.

Values:

enumerator kNETC_IPFSwtPortMasMask
Switch port masquerading Mask

enumerator kNETC_IPFEthernetMask
Ethernet type Mask

enumerator kNETC_IPFOuterVlanMask
Outer VLAN Mask

enumerator kNETC_IPFInnerVlanMask
Inner VLAN Mask

enumerator kNETC_IPFSeqTagMask
Sequence Tag Code Mask

enumerator kNETC_IPFIpHeaderMask
IP Header Mask

enumerator kNETC_IPFIpVersionMask
IP Version Mask

enumerator kNETC_IPFIpExtMask
IPv4 option / IPv6 extension Mask

enumerator kNETC_IPFL4HeaderMask
L4 Code Mask

enumerator kNETC_IPFWakeOnLanMask
Wake-on-LAN Magic Packet Mask

enum _netc_tb_ipf_seq_tag
Ingress Port Filter frame attribute Sequence Tag Code.

Values:

enumerator kNETC_IPFNoRtag
R-TAG/HSR tag is not present

enumerator kNETC_IPFDraftRtag
802.1CB draft 2.0 R-TAG is present

enumerator kNETC_IPFRtag
802.1CB R-TAG is present

enumerator kNETC_IPFHsrTag
HSR Tag is present

enum _netc_tb_ipf_l4_header
Ingress Port Filter frame attribute L4 Header Code.

Values:

enumerator kNETC_IPFOtherL4
The L4 Header is considered as other L4 if it is not one of the following L4 Headers

enumerator kNETC_IPFTcp
TCP header is present

enumerator kNETC_IPFUdp
UDP header is present

enumerator kNETC_IPFSctp
SCTP header is present

enum _netc_tb_ipf_forward_action
Ingress port filter forwarding Action.

Values:

enumerator kNETC_IPFForwardDiscard
Frame be discard

enumerator kNETC_IPFForwardPermit
Frame be permit

enumerator kNETC_IPFRedirectToMgmtPort
Redirect frame to switch management port without any frame modification, Switch only

enumerator kNETC_IPFCopyToMgmtPort
Copy frame to switch management port without any frame modification, Switch only

enum _netc_tb_ipf_filter_action
Ingress port filter Filter Action.

Values:

enumerator kNETC_IPFNoAction
No action

enumerator kNETC_IPFWithRatePolicer

Rate action with the Rate Policer Entry ID (RP_EID) set to the value configured in the fltaTgt field

enumerator kNETC_IPFWithIngressStream

Ingress stream identification action where the Ingress Stream Entry ID (IS_EID) is set to the value configured in the fltaTgt field

enumerator kNETC_IPFWithL2Filtering

Setting a pre L2 filtering SI bitmap (set to the value configured in the fltaTgt) that will be used by the L2 filtering function to determine the final SI bitmap, ENETC only

enum __netc_tb_isi_key_type

Stream identification table key type.

Values:

enumerator kNETC_KCRule0

Use key construction rule 0 (ISIDKC0CR0)

enumerator kNETC_KCRule1

Use key construction rule 1 (ISIDKC1CR0)

enumerator kNETC_KCRule2

Use key construction rule 2 (ISIDKC2CR0). Only for SWITCH

enumerator kNETC_KCRule3

Use key construction rule 3 (ISIDKC3CR0). Only for SWITCH

enum __netc_tb_is_isq_action

Ingress Stream table Ingress Sequence Action.

Values:

enumerator kNETC_ISNotPerformFRER

Not perform Frame FRER sequence generation function

enumerator kNETC_ISPerformFRER

Perform Frame FRER sequence generation function

enum __netc_tb_is_forward_action

Ingress Stream table forwarding Action.

Values:

enumerator kNETC_ISDiscard

Frame be discard

enumerator kNETC_ISRedirectToMgmtPort

Frame be Re-direct frame to switch management port, Switch only

enumerator kNETC_ISAllow

Frame is allow without setting the pre L2 filtering SI bitmap, ENETC only

enumerator kNETC_ISAllowWithSIMap

Frame is allow with setting the pre L2 filtering SI bitmap to the value configured in the SI_MAP field, ENETC only

enumerator kNETC_ISStreamForward

Frame is forwarded to the port(s) specified in the EGRESS_PORT_BITMAP field, Switch only

enumerator kNETC_ISBridgeForward

Frame is do 802.1Q Bridge forwarding (VLAN processing and L2 forwarding), Switch only

enumerator kNETC_ISCopyToMgmtPortAndStream

Copy frame to switch management port with specified HR and stream forwarding, Switch only

enumerator kNETC_ISCopyToMgmtPortAndBridge

Copy frame to switch management port with specified HR and Bridge forwarding, Switch only

enum _netc_tb_is_ctd_mode

Ingress Stream table Cut-Through Disable mode.

Values:

enumerator kNETC_ISNoCTD

Do not override cut-through state

enumerator kNETC_ISSinglePortCTD

Disable cut-through for the outgoing port specified in the cfge ePort field

enumerator kNETC_ISAllPortCTD

Disable cut-through for all ports specified in cfge portBitmap field

enum _netc_tb_rp_update_action

Rate Policer Table Update Actions.

Values:

enumerator kNETC_RPCfgEUpdate

Configuration Element Update

enumerator kNETC_RPFeEUpdate

Functional Enable Element Update

enumerator kNETC_RPPsEUpdate

Policer State Element Update Element Update

enumerator kNETC_RPStsEUpdate

Statistics Element Update

enum _netc_tb_sgi_update_action

Stream Gate Instance table Update Actions.

Values:

enumerator kNETC_SGIAcfEUpdate

Admin Configuration Element

enumerator kNETC_SGICfgEUpdate

Configuration Element Update

enumerator kNETC_SGISgisEUpdate

Stream Gate Instance State Element Update

enum _netc_tb_sgi_state

Stream Gate Instance State.

Values:

enumerator kNETC_GSNotOper

Gate instance is not operational or Gate instance and lists are not valid

enumerator kNETC_GSUseDefaultParam

Gate instance is operational but no stream gate control list specified, use default Gate Instance parameters

enumerator kNETC_GSUseDefUntilAdminAct

Use default Gate Instance parameters until administrative stream gate control list takes effect

enumerator kNETC_GSUseOperUntilAdminAct

Use Operational stream gate control list until new administrative stream gate control list takes effect

enumerator kNETC_GSUseOperList

Operational stream gate control list is in effect

enum _netc_tb_fm_layer2_act

Frame Modification table Layer 2 Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

Values:

enumerator kNETC_UseL2HeaderAct

L2 actions are specified in L2 header action fields macHdrAct, vlanHdrAct and sqtAct

enumerator kNETC_UseSpecPlayload

The entire L2 PDU is replaced with fmdBytes of data specified in fmdEID, not applicable for ingress frame modifications

enum _netc_tb_fm_mac_header_act

Frame Modification table Layer 2 Header MAC Actions.

Note: Ingress frame modifications only support kNETC_NoAction or kNETC_ReplDmac.

Values:

enumerator kNETC_NoMacAction

No Mac header action

enumerator kNETC_ReplSmac

Replace SMAC with the contents of the port's PMAR0/1 register, The port is specified by smacPort field

enumerator kNETC_ReplSmacAndDmacAct1

Replace SMAC and DMAC, The content of SMAC is the same as kNETC_ReplaceSMAC, the DMAC is specified by dmac[6] field

enumerator kNETC_ReplSmacAndDmacAct2

Replace SMAC and DMAC, The content of SMAC is the same as kNETC_ReplaceSMAC, the DMAC is specified by frame's SMAC

enumerator kNETC_ReplDmac

Replace DMAC with specified dmac[6] field value

enumerator kNETC_SwapDmacAndSmac

Swap DMAC and SMAC

enum _netc_tb_fm_vlan_header_act

Frame Modification table Layer 2 VLAN Actions.

Note: For use Delete or Replace action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_NoVlanAction

No VLAN header action

enumerator kNETC_DelOuterVlan

Delete outer VLAN header

enumerator kNETC_AddOuterVlan

Add outer VLAN header (new VLAN data will be inserted in the outer position), the VID, PCP, DEI and TPID values are specified by outerVidAct, outerPcpAct, outerDeiAct and outerTpidAct field

enumerator kNETC_ReplOuterVlan

Replace outer VLAN header, the VID, PCP, DEI and TPID values are specified by outerVidAct, outerPcpAct, outerDeiAct and outerTpidAct field

enum _netc_tb_fm_outer_vid_act

Frame Modification table Layer 2 outer VLAN VID Actions.

Note: For use kNETC_UseFrameVID action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFrameVid

Use the VID from the valid outer VLAN header of the received frame

enumerator kNETC_UseSpecVid

Use the VID specified in the outerVlanID field

enum _netc_tb_fm_outer_tpid_act

Frame Modification table Outer TPID action.

Note: For use kNETC_UseFrameTpid action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFrameTpid

Use TPID from outer VLAN header

enumerator kNETC_UseStdCVlan

Set TPID to Standard C-VLAN 0x8100

enumerator kNETC_UseStdSVlan

Set TPID to Standard S-VLAN 0x88A8

enumerator kNETC_UseCustomCVlan

Set TPID to Custom C-VLAN as defined by CVLANR1[ETYPE]

enumerator kNETC_UseCustomSVlan

Set TPID to Custom S-VLAN as defined by CVLANR2[ETYPE]

enum _netc_tb_fm_outer_pcp_act

Frame Modification table Outer PCP action.

Note: For use kNETC_UseFramePcp/kNETC_UseFramePcpMap action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFramePcp

Use PCP from frame outer VLAN header

enumerator kNETC_UseSpecPcp

Use the PCP specified in the outerVlanPcp field

enumerator kNETC_UseFramePcpMap

The PCP is mapping from frame outer VLAN PCP (do mapping according to the PCP to PCP mapping profile which specified in PPCPDEIMR[IPCPMP/EPCPMP])

enumerator kNETC_UseQosMap

The PCP is mapping from internal QoS (IPV, DR) (do mapping according to the QOS to PCP mapping profile which specified in QOSVLANMPaR0/1/2/3), not applicable for ingress frame modifications

enum _netc_tb_fm_outer_dei_act

Frame Modification table Outer DEI action.

Note: For use kNETC_UseFrameDei action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFrameDei

Use DEI from frame outer VLAN header

enumerator kNETC_UseSpecDei

Use the DEI specified in the outerVlanDei field

enumerator kNETC_UseDrMap

The DEI is mapping from internal DR (do mapping according to the DR to DEI mapping profile which specified in PPCPDEIMR[DRnDEI], not applicable for ingress frame modifications

enum _netc_tb_fm_payload_act

Frame Modification table Payload Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

Values:

enumerator kNETC_NoAction

No Action

enumerator kNETC_ReplAllEthPld
Remove entire Ethernet payload and insert with fmdBytes of data specified in fmdEID

enumerator kNETC_ReplPldWithOffset
Replace fmdBytes of raw data in the Ethernet payload starting at pldOffset, data specified in fmdEID

enum _netc_tb_fdb_update_action
FDB table Update Actions.

Values:

enumerator kNETC_FDBCfgEUpdate
Configuration Element Update

enumerator kNETC_FDBActEUpdate
Activity Element Update

enum _netc_tb_fdb_ctd_mode
FDB table Cut-Through Disable mode.

Values:

enumerator kNETC_FDBNoCTD
Do not override cut-through state

enumerator kNETC_FDBSinglePortCTD
Disable cut-through for the outgoing port specified in the cfge ePort field

enumerator kNETC_FDBAllPortCTD
Disable cut-through for all ports specified in cfge portBitmap field

enum _netc_tb_fdb_sc_keye_mc
FDB table search criteria Key Element Match Criteria.

Values:

enumerator kNETC_FDBKeyeMacthAny
Match any Key Element Criteria

enumerator kNETC_FDBKeyeMacthFID
Match Key Element FID

enumerator kNETC_FDBKeyeMacthMacMulticast
Match Key Element MAC Multicast bit (MAC_ADDR most significant byte's least significant bit)

enumerator kNETC_FDBKeyeMacthBoth
Match both FID field and MAC Multicast bit

enum _netc_tb_fdb_sc_cfge_mc
FDB table search criteria Configuration Element Match Criteria.

Values:

enumerator kNETC_FDBCfgeMacthAny
Match any Configuration Element Criteria

enumerator kNETC_FDBCfgeMacthDynamic
Match Configuration Element dynamic field

enumerator kNETC_FDBCfgeMacthPortBitmap
Match Configuration Element portBitmap field

enumerator kNETC_FDBCfgeMacthBoth
Match both dynamic field and portBitmap

enum __netc_tb_fdb_sc_acte_mc
FDB table search criteria Activity Element Match Criteria.

Values:

enumerator kNETC_FDBActeMacthAny
Match any Activity Element Criteria

enumerator kNETC_FDBActeMatchExact
Exact match with Activity Element

enum __netc_tb_l2mcf_key_type
L2 IPV4 Multicast Filter table key type.

Values:

enumerator kNETC_IPv4ASMKey
Key consists of a filtering ID (FID) and destination multicast IPv4 address

enumerator kNETC_IPv4SSMKey
Key consists of a filtering ID (FID), IPv4 source address and multicast IPv4 destination address

enum __etc_tb_l2mcf_sc_keye_mc
L2 IPV4 Multicast Filter table search criteria Key Element Match Criteria.

Values:

enumerator kNETC_L2MCFKeyeMacthAny
Match any Key Element Criteria

enumerator kNETC_L2MCFKeyeMacthFID
Match Key Element FID

enum __etc_tb_l2mcf_sc_cfge_mc
L2 IPV4 Multicast Filter table search criteria Configuration Element Match Criteria.

Values:

enumerator kNETC_L2MCFCfgeMacthAny
Match any Configuration Element Criteria

enumerator kNETC_L2MCFCfgeMacthDynamic
Match Configuration Element dynamic field

enumerator kNETC_L2MCFCfgeMacthPortBitmap
Match Configuration Element portBitmap field

enumerator kNETC_L2MCFCfgeMacthBoth
Match both dynamic field and portBitmap

enum __etc_tb_l2mcf_sc_acte_mc
FDB table search criteria Activity Element Match Criteria.

Values:

enumerator kNETC_L2MCFActeMacthAny
Match any Activity Element Criteria

enumerator kNETC_L2MCFActeMatchExact
Exact match with Activity Element

enum _netc_tb_iseqg_sqtag

Sequence Tag Type.

Values:

enumerator kNETC_SqDraftRTag

802.1CB draft 2.0 R-TAG.

enumerator kNETC_SqRTag

802.1CB R-TAG.

enumerator kNETC_SqHsrTag

HSR Tag.

enum _netc_tb_iseqg_update_action

Ingress Sequence Generation Table Update Actions.

Values:

enumerator kNETC_ISEQGCfgEUpdate

Configuration Element Update

enumerator kNETC_ISEQGSgsEUpdate

Sequence Generation Element Update

enum _netc_tb_eseqr_sqtag

Egress Sequence Recovery table Sequence Tag Type.

Values:

enumerator kNETC_AcceptAnyTag

Accept any incoming tag type (802.1CB draft 2.0 R-TAG, 802.1CB R-TAG or HSR Tag)

enumerator kNETC_AcceptDraftRTag

802.1CB draft 2.0 R-TAG.

enumerator kNETC_AcceptRTag

802.1CB R-TAG.

enumerator kNETC_AcceptHsrTag

HSR Tag.

enum _netc_tb_tgs_entry_id

Time Gate Scheduling table entry ID for switch and ENETC.

Values:

enumerator kNETC_TGSSwtPort0

Switch PORT 0 entry ID

enumerator kNETC_TGSSwtPort1

Switch PORT 1 entry ID

enumerator kNETC_TGSSwtPort2

Switch PORT 2 entry ID

enumerator kNETC_TGSSwtPort3

Switch PORT 3 entry ID

enumerator kNETC_TGSSwtPort4

Switch PORT 4 entry ID

enumerator kNETC_TGSEnetc0Port

ENETC 0 port entry ID

enumerator kNETC_TGSEnetc1Port
ENETC 1 port entry ID

enum __netc_tb_tgs_gate_type
Administrative gate operation type.

Values:

enumerator kNETC_SetGateStates
HoldRequest is unchanged

enumerator kNETC_SetAndHoldMac
HoldRequest is set to value hold, only active when enable preemption

enumerator kNETC_SetAndReleaseMac
HoldRequest is set to value release, only active when enable preemption

enum __netc_tb_et_efm_mode
Egress Frame Modification entry mode.

Values:

enumerator kNETC_NormalMode
Egress Frame Modification entry use normal/Default mode

enumerator kNETC_L2Act1
Egress Frame Modification entry l2Act = kNETC_UseSpecPlayload

enumerator kNETC_PldAct1
Egress Frame Modification entry pldAct = kNETC_ReplAllEthPld

enum __netc_tb_et_esq_act
Egress Sequence Actions.

Values:

enumerator kNETC_NoEsqAction
No Egress Sequence Action required

enumerator kNETC_HasEsqAction
Has Egress Sequence Recovery action

enum __netc_tb_et_ec_act
Egress Counter Action.

Values:

enumerator kNETC_NoEcCounter
Do not increment egress frame counter

enumerator kNETC_HasEcCounter
Increment egress frame counter

enum __netc_tb_etmcq_update_action
ETM Class Queue table Update Actions.

Values:

enumerator kNETC_CQCfgEUpdate
Configuration Element Update

enumerator kNETC_CQStsEUpdate
Statistics Element Update, all counters (except FRM_CNT) within the Statistics Element are reset

enum _netc_tb_etmcs_entry_id

ETM Class Scheduler table entry ID.

Values:

enumerator kNETC_CSSwtPort0

CS Switch PORT 0 entry ID

enumerator kNETC_CSSwtPort1

CS Switch PORT 1 entry ID

enumerator kNETC_CSSwtPort2

CS Switch PORT 2 entry ID

enumerator kNETC_CSSwtPort3

CS Switch PORT 3 entry ID

enumerator kNETC_CSSwtPort4

CS Switch PORT 4 entry ID

enum _netc_tb_etmcs_ca_assg

ETM Class Scheduler table Class queue assignment to scheduler inputs mode.

Values:

enumerator kNETC_CQ7AssignToSchedIn15

CQ 7 assignment to scheduler input 15, means all CQ use strict priority

enumerator kNETC_CQ7AssignToSchedIn14

CQ 7 assignment to scheduler input 14

enumerator kNETC_CQ7AssignToSchedIn13

CQ 7 assignment to scheduler input 13

enumerator kNETC_CQ7AssignToSchedIn12

CQ 7 assignment to scheduler input 12

enumerator kNETC_CQ7AssignToSchedIn11

CQ 7 assignment to scheduler input 11

enumerator kNETC_CQ7AssignToSchedIn10

CQ 7 assignment to scheduler input 10

enumerator kNETC_CQ7AssignToSchedIn9

CQ 7 assignment to scheduler input 9

enumerator kNETC_CQ7AssignToSchedIn8

CQ 7 assignment to scheduler input 8

enumerator kNETC_CQ7AssignToSchedIn7

CQ 7 assignment to scheduler input 7, means all CQ use weighted fair

enum _netc_tb_bp_fc_cfg

Buffer Pool Flow Control (FC) Configuration.

Values:

enumerator kNETC_FlowCtrlDisable

Flow Control disabled

enumerator kNETC_FlowCtrlWithBP

Flow Control enabled using only buffer pool FC state.

enumerator kNETC_FlowCtrlWithSBP

Flow Control enabled using only shared buffer pool FC state.

enumerator kNETC_FlowCtrlWithBPAndSBP

Flow Control enabled using both buffer pool and shared buffer pool FC state, only both 1 trigger the Flow Control ON

typedef enum *_netc_tb_index* netc_tb_index_t

Table index.

typedef enum *_netc_tb_cmd* netc_tb_cmd_t

Table management command operations.

typedef enum *_netc_tb_access_mode* netc_tb_access_mode_t

Table Access Method.

typedef enum *_netc_cbd_version* netc_cbd_version_t

NTMP version.

typedef enum *_netc_cmd_error* netc_cmd_error_t

Table command response error status.

typedef union *_netc_cmd_bd* netc_cmd_bd_t

The Switch/SI command BD data structure.

typedef struct *_netc_cmd_bdr_config* netc_cmd_bdr_config_t

Configuration for the Switch/SI command BD Ring Configuration.

typedef struct *_netc_cmd_bdr* netc_cmd_bdr_t

The Switch/SI command BD ring handle data structure.

typedef struct *_netc_tb_common_header* netc_tb_common_header_t

Table request data buffer common header.

typedef enum *_netc_fm_vlan_ud_act* netc_fm_vlan_ud_act_t

Frame Modification VLAN Update/Delete Action.

Note: Misconfiguration error if replace or delete action is specified and if VLAN tag is not present in frame.

---

typedef enum *_netc_fm_sqt_act* netc_fm_sqt_act_t

Frame Modification Sequence Tag Action.

Note: Must be set to 000b for Ingress frame modification, otherwise misconfiguration error..

---

typedef enum *_netc_fm_vlan_ar_act* netc_fm_vlan_ar_act_t

Frame Modification VLAN Add/Replace Action.

Note: For ingress frame modificaion with 00b or 01b, use the ingress port to select PCP and DEI from the Bridge port default VLAN register (BPDVR). For egress frame modification with 00b or 01b, use the internal QoS associated with the frame (IPV, DR) to access the QoS to PCP mapping profile (PQOSMR[QVMP] , QOSVLANMPaR0/1/2/3) to set the new PCP value. Use internal DR associated with frame to access the DR to DEI mapping profile (PPCPDEIMR[DRnDEI]) to set the new DEI value.

---

typedef enum *_netc_tb_eteid_access_mode* netc_tb_eteid_access_mode_t

Define FDB/L2MCF/IS table entry access the primary Egress Treatment table entry group mode.

Note: The FDB/L2 IPv4 Multicast filter table has precedence over any assignment made via the Ingress Stream table. For Mulit port mode, the index to access the Egress Treatment table is computed by adding an offset to the base index of the Egress Treatment group. That offset is derived from the applicability bitmap as follows: starting from the lowest significant bit of the bitmap, the first encountered bit set to 1, corresponds to offset 0, and so on. This continues till the destination port location in the bitmap is reached

---

typedef enum *_netc_tb_ipf_update_action* netc_tb_ipf_update_action_t

Ingress Port Filter Table Update Actions.

typedef enum *_netc_tb_ipf_attr_mask* netc_tb_ipf_attr_mask_t

Ingress Port Filter frame attribute mask.

typedef enum *_netc_tb_ipf_seq_tag* netc_tb_ipf_seq_tag_t

Ingress Port Filter frame attribute Sequence Tag Code.

typedef enum *_netc_tb_ipf_l4_header* netc_tb_ipf_l4_header_t

Ingress Port Filter frame attribute L4 Header Code.

typedef struct *_netc_tb_ipf_keye* netc_tb_ipf_keye_t

Ingress Port Filter key element.

typedef enum *_netc_tb_ipf_forward_action* netc_tb_ipf_forward_action_t

Ingress port filter forwarding Action.

typedef enum *_netc_tb_ipf_filter_action* netc_tb_ipf_filter_action_t

Ingress port filter Filter Action.

typedef struct *_netc_tb_ipf_cfge* netc_tb_ipf_cfge_t

Ingress port filter config element.

typedef struct *_netc_tb_ipf_stse* netc_tb_ipf_stse_t

Ingress port filter statistic element.

typedef struct *_netc_tb_ipf_req_data* netc_tb_ipf_req_data_t

Ingress port filter table entry config.

typedef struct *_netc_tb_ipf_rsp_data* netc_tb_ipf_rsp_data_t

Ingress port filter table response data.

typedef struct *_netc_tb_ipf_data* netc_tb_ipf_data_t

Ingress Port filter table data buffer.

typedef struct *_netc_tb_ipf_config* netc_tb_ipf_config_t

Ingress Port filter entry config.

typedef enum *_netc_tb_isi_key_type* netc_tb_isi_key_type

Stream identification table key type.

typedef struct *_netc_tb_isi_keye* netc_tb_isi_keye_t

Stream identification table key element.

typedef struct *_netc_tb_isi_cfge* netc_tb_isi_cfge_t

Stream identification table config element.

typedef struct *_netc_tb_isi_req_data* netc_tb_isi_req_data_t

Stream identification table request data buffer.

typedef struct *_netc_tb_isi_rsp_data* netc_tb_isi_rsp_data_t
Stream identification table request response data buffer.

typedef struct *_netc_tb_isi_data* netc_tb_isi_data_t
Stream identification table data buffer.

typedef struct *_netc_tb_isi_config* netc_tb_isi_config_t
Stream identification table entry config.

typedef enum *_netc_tb_is_isq_action* netc_tb_is_isq_action_t
Ingress Stream table Ingress Sequence Action.

typedef enum *_netc_tb_is_forward_action* netc_tb_is_forward_action_t
Ingress Stream table forwarding Action.

typedef enum *_netc_tb_is_ctd_mode* netc_tb_is_ctd_mode_t
Ingress Stream table Cut-Through Disable mode.

typedef *netc_tb_eteid_access_mode_t* netc_tb_is_oeteid_mode_t
Ingress Stream table Override ET_EID mode.

typedef struct *_netc_tb_is_cfge* netc_tb_is_cfge_t
Ingress Stream table config element.

typedef struct *_netc_tb_is_req_data* netc_tb_is_req_data_t
Ingress Stream table request data buffer.

typedef struct *_netc_tb_is_rsp_data* netc_tb_is_rsp_data_t
Ingress Stream table request response data buffer.

typedef struct *_netc_tb_is_data* netc_tb_is_data_t
Ingress Stream table data buffer.

typedef struct *_netc_tb_is_config* netc_tb_is_config_t
Ingress Stream table entry config.

typedef struct *_netc_tb_isf_keye* netc_tb_isf_keye_t
Ingress Stream Filter table key element.

typedef struct *_netc_tb_isf_cfge* netc_tb_isf_cfge_t
Ingress Stream Filter table config element.

typedef struct *_netc_tb_isf_req_data* netc_tb_isf_req_data_t
Ingress Stream Filter table request data buffer.

typedef struct *_netc_tb_isf_rsp_data* netc_tb_isf_rsp_data_t
Ingress Stream Filter table request response data buffer.

typedef struct *_netc_tb_isf_data* netc_tb_isf_data_t
Ingress Stream Filter table data buffer.

typedef struct *_netc_tb_isf_config* netc_tb_isf_config_t
Ingress Stream Filter table entry config.

typedef enum *_netc_tb_rp_update_action* netc_tb_rp_update_action_t
Rate Policer Table Update Actions.

typedef *netc_tc_sdu_type_t* netc_tb_rp_sdu_type_t
Rate Policer Table Protocol/Service Data Unit Type.

typedef struct *_netc_tb_rp_cfge* netc_tb_rp_cfge_t
Rate Policer table config element.

typedef struct *_netc_tb_rp_fee* netc_tb_rp_fee_t
Rate Policer table Function Enable element.

typedef struct *_netc_tb_rp_pse* netc_tb_rp_pse_t
Rate Policer table Policer State element.

typedef struct *_netc_tb_rp_stse* netc_tb_rp_stse_t
Rate Policer table statistic element.

typedef struct *_netc_tb_rp_req_data* netc_tb_rp_req_data_t
Rate Policer table request data buffer.

typedef struct *_netc_tb_rp_rsp_data* netc_tb_rp_rsp_data_t
Rate Policer table request response data buffer.

typedef struct *_netc_tb_rp_data* netc_tb_rp_data_t
Rate Policer table data buffer.

typedef struct *_netc_tb_rp_config* netc_tb_rp_config_t
Rate Policer table entry config.

typedef struct *_netc_tb_isc_stse* netc_tb_isc_stse_t
Ingress Stream Count table statistic element.

typedef struct *_netc_tb_isc_req_data* netc_tb_isc_req_data_t
Ingress Stream Count table request data buffer.

typedef struct *_netc_tb_isc_rsp_data* netc_tb_isc_rsp_data_t
Ingress Stream Count table request response data buffer.

typedef struct *_netc_tb_isc_data* netc_tb_isc_data_t
Ingress Stream Count table data buffer.

typedef enum *_netc_tb_sgi_update_action* netc_tb_sgi_update_action_t
Stream Gate Instance table Update Actions.

typedef *netc_tc_sdu_type_t* netc_tb_sgi_sdu_type_t
Stream Gate Instance table Protocol/Service Data Unit Type.

typedef enum *_netc_tb_sgi_state* netc_tb_sgi_state_t
Stream Gate Instance State.

typedef struct *_netc_tb_sgi_cfge* netc_tb_sgi_cfge_t
Stream Gate Instance table config element.

typedef struct *_netc_tb_sgi_acfge* netc_tb_sgi_acfge_t
Stream Gate Instance table Admin Configuration element.

typedef struct *_netc_tb_sgi_icfge* netc_tb_sgi_icfge_t
Stream Gate Instance table Initial Configuration element.

typedef struct *_netc_tb_sgi_sgise* netc_tb_sgi_sgise_t
Stream Gate Instance table stream gate instance state element.

typedef struct *_netc_tb_sgi_req_data* netc_tb_sgi_req_data_t
Stream Gate Instance table request data buffer.

typedef struct *_netc_tb_sgi_rsp_data* netc_tb_sgi_rsp_data_t
Stream Gate Instance table request response data buffer.

typedef struct *_netc_tb_sgi_data* netc_tb_sgi_data_t
Stream Gate Instance table data buffer.

typedef struct *_netc_tb_sgi_config* netc_tb_sgi_config_t

Stream Gate Instance table entry config.

typedef struct *_netc_sgcl_gate_entry* netc_sgcl_gate_entry_t

Defines the Stream Gate Control entry structure.

typedef struct *_netc_tb_sgcl_cfge* netc_tb_sgcl_cfge_t

Stream Gate Control List table config element.

typedef struct *_netc_tb_sgcl_sgclse* netc_tb_sgcl_sgclse_t

Stream Gate Control List table Stream Gate Control List State element.

typedef struct *_netc_tb_sgcl_req_data* netc_tb_sgcl_req_data_t

Stream Gate Control List table request data buffer.

typedef struct *_netc_tb_sgcl_rsp_data* netc_tb_sgcl_rsp_data_t

Stream Gate Control List table request response data buffer.

typedef struct *_netc_tb_sgcl_data* netc_tb_sgcl_data_t

Stream Gate Control List table data buffer.

typedef struct *_netc_tb_sgcl_gcl* netc_tb_sgcl_gcl_t

Stream Gate Control List table entry gate control list structure.

typedef enum *_netc_tb_fm_layer2_act* netc_tb_fm_layer2_act_t

Frame Modification table Layer 2 Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

---

typedef enum *_netc_tb_fm_mac_header_act* netc_tb_fm_mac_header_act_t

Frame Modification table Layer 2 Header MAC Actions.

Note: Ingress frame modifications only support kNETC_NoAction or kNETC_ReplDmac.

---

typedef enum *_netc_tb_fm_vlan_header_act* netc_tb_fm_vlan_header_act_t

Frame Modification table Layer 2 VLAN Actions.

Note: For use Delete or Replace action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

---

typedef enum *_netc_tb_fm_outer_vid_act* netc_tb_fm_outer_vid_act_t

Frame Modification table Layer 2 outer VLAN VID Actions.

Note: For use kNETC_UseFrameVID action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

---

typedef *netc_fm_sqt_act_t* netc_tb_fm_sqt_act_t

Frame Modification table Sequence Tag Action.

Note: For use kNETC_ReomveTag action, If R-TAG/draft 2.0 R-TAG/HSR tag not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

---

typedef enum *_netc_tb_fm_outer_tpid_act* netc_tb_fm_outer_tpid_act_t

Frame Modification table Outer TPID action.

Note: For use kNETC_UseFrameTpid action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

typedef enum *_netc_tb_fm_outer_pcp_act* netc_tb_fm_outer_pcp_act_t

Frame Modification table Outer PCP action.

Note: For use kNETC_UseFramePcp/kNETC_UseFramePcpMap action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

typedef enum *_netc_tb_fm_outer_dei_act* netc_tb_fm_outer_dei_act_t

Frame Modification table Outer DEI action.

Note: For use kNETC_UseFrameDei action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

typedef enum *_netc_tb_fm_payload_act* netc_tb_fm_payload_act_t

Frame Modification table Payload Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

typedef struct *_netc_tb_fm_cfge* netc_tb_fm_cfge_t

Frame Modification table config element.

typedef struct *_netc_tb_fm_req_data* netc_tb_fm_req_data_t

Frame Modification table request data buffer.

typedef struct *_netc_tb_fm_rsp_data* netc_tb_fm_rsp_data_t

Frame Modification table request response data buffer.

typedef struct *_netc_tb_fm_data* netc_tb_fm_data_t

Frame Modification table data buffer.

typedef struct *_netc_tb_fm_config* netc_tb_fm_config_t

Frame Modification table entry config.

typedef struct *_netc_tb_fmd_req_data* netc_tb_fmd_req_data_t

Frame Modification Data table request data buffer.

typedef struct *_netc_tb_fmd_rsp_data* netc_tb_fmd_rsp_data_t

Frame Modification Data table request response data buffer.

typedef struct *_netc_tb_fmd_data* netc_tb_fmd_data_t

Frame Modification Data table data buffer.

typedef struct *_netc_tb_fmd_update_config* netc_tb_fmd_update_config_t

Frame Modification data table entry update config.

typedef struct *_netc_tb_fmd_query_buffer* netc_tb_fmd_query_buffer_t

Frame Modification data table entry query data buffer.

typedef struct *_netc_tb_vf_keye* netc_tb_vf_keye_t
: Vlan Filter table key element.

typedef struct *_netc_tb_vf_cfge* netc_tb_vf_cfge_t
: Vlan Filter table config element.

typedef struct *_netc_tb_vf_search_criteria* netc_tb_vf_search_criteria_t
: Vlan Filter table search criteria format.

typedef struct *_netc_tb_vf_req_data* netc_tb_vf_req_data_t
: Vlan Filter table request data buffer.

typedef struct *_netc_tb_vf_rsp_data* netc_tb_vf_rsp_data_t
: Vlan Filter table request response data buffer.

typedef struct *_netc_tb_vf_data* netc_tb_vf_data_t
: Vlan Filter table data buffer.

typedef struct *_netc_tb_vf_config* netc_tb_vf_config_t
: Vlan Filter table entry config.

typedef enum *_netc_tb_fdb_update_action* netc_tb_fdb_update_action_t
: FDB table Update Actions.

typedef *netc_tb_eteid_access_mode_t* netc_tb_fdb_oeteid_mode_t
: FDB table entry defined the egress packet processing actions (will cover the actions which specified in the Egress Treatment table)

typedef enum *_netc_tb_fdb_ctd_mode* netc_tb_fdb_ctd_mode_t
: FDB table Cut-Through Disable mode.

typedef struct *_netc_tb_fdb_keye* netc_tb_fdb_keye_t

typedef struct *_netc_tb_fdb_cfge* netc_tb_fdb_cfge_t
: FDB table configuration element.

typedef struct *_netc_tb_fdb_acte* netc_tb_fdb_acte_t
: FDB table Activity element.

typedef enum *_netc_tb_fdb_sc_keye_mc* netc_tb_fdb_sc_keye_mc_t
: FDB table search criteria Key Element Match Criteria.

typedef enum *_netc_tb_fdb_sc_cfge_mc* netc_tb_fdb_sc_cfge_mc_t
: FDB table search criteria Configuration Element Match Criteria.

typedef enum *_netc_tb_fdb_sc_acte_mc* netc_tb_fdb_sc_acte_mc_t
: FDB table search criteria Activity Element Match Criteria.

typedef struct *_netc_tb_fdb_search_criteria* netc_tb_fdb_search_criteria_t
: FDB table search criteria format.

typedef struct *_netc_tb_fdb_req_data* netc_tb_fdb_req_data_t
: FDB table request data buffer.

typedef struct *_netc_tb_fdb_rsp_data* netc_tb_fdb_rsp_data_t
: FDB table request response data buffer.

typedef struct *_netc_tb_fdb_data* netc_tb_fdb_data_t
: FDB table data buffer.

typedef struct *_netc_tb_fdb_config* netc_tb_fdb_config_t
: FDB table entry config.

typedef enum *_netc_tb_l2mcf_key_type* netc_tb_l2mcf_key_type_t
L2 IPV4 Multicast Filter table key type.

typedef struct *_netc_tb_l2mcf_keye* netc_tb_l2mcf_keye_t
L2 IPV4 Multicast Filter table key element.

typedef enum *_etc_tb_l2mcf_sc_keye_mc* etc_tb_l2mcf_sc_keye_mc_t
L2 IPV4 Multicast Filter table search criteria Key Element Match Criteria.

typedef enum *_etc_tb_l2mcf_sc_cfge_mc* etc_tb_l2mcf_sc_cfge_mc_t
L2 IPV4 Multicast Filter table search criteria Configuration Element Match Criteria.

typedef enum *_etc_tb_l2mcf_sc_acte_mc* etc_tb_l2mcf_sc_acte_mc_t
FDB table search criteria Activity Element Match Criteria.

typedef *netc_tb_fdb_cfge_t* netc_tb_l2mcf_cfge_t
L2 IPV4 Multicast Filter table config element.

typedef *netc_tb_fdb_acte_t* netc_tb_l2mcf_acte_t
L2 IPV4 Multicast Filter table activity lement.

typedef struct *_etc_tb_l2mcf_search_criteria* netc_tb_l2mcf_search_criteria_t
L2 IPV4 Multicast Filter table search criteria format.

typedef struct *_netc_tb_l2mcf_req_data* netc_tb_l2mcf_req_data_t
L2 IPV4 Multicast Filter table request data buffer.

typedef struct *_netc_tb_l2mcf_rsp_data* netc_tb_l2mcf_rsp_data_t
L2 IPV4 Multicast Filter table request response data buffer.

typedef struct *_netc_tb_l2mcf_data* netc_tb_l2mcf_data_t
L2 IPV4 Multicast Filter table data buffer.

typedef struct *_netc_tb_l2mcf_config* netc_tb_l2mcf_config_t
L2 IPV4 Multicast Filter table entry config.

typedef enum *_netc_tb_iseqg_sqtag* netc_tb_iseqg_sqtag_t
Sequence Tag Type.

typedef struct *_netc_tb_iseqg_cfge* netc_tb_iseqg_cfge_t
Ingress Sequence Generation table config element.

typedef struct *_netc_tb_iseqg_sgse* netc_tb_iseqg_sgse_t
Ingress Sequence Generation table Sequence generation state element.

typedef struct *_netc_tb_iseqg_req_data* netc_tb_iseqg_req_data_t
Ingress Sequence Generation table request data buffer.

typedef struct *_netc_tb_iseqg_rsp_data* netc_tb_iseqg_rsp_data_t
Ingress Sequence Generation table request response data buffer.

typedef struct *_netc_tb_iseqg_data* netc_tb_iseqg_data_t
Ingress Sequence Generation table data buffer.

typedef struct *_netc_tb_iseqg_config* netc_tb_iseqg_config_t
Ingress Sequence Generation table entry config.

typedef enum *_netc_tb_iseqg_update_action* netc_tb_iseqg_update_action_t
Ingress Sequence Generation Table Update Actions.

typedef enum *_netc_tb_eseqr_sqtag* netc_tb_eseqr_sqtag_t
Egress Sequence Recovery table Sequence Tag Type.

typedef struct *_netc_tb_eseqr_cfge* netc_tb_eseqr_cfge_t
Egress Sequence Recovery table config element.

typedef struct *_netc_tb_eseqr_stse* netc_tb_eseqr_stse_t
Egress Sequence Recovery table statistic element.

typedef struct *_netc_tb_eseqr_srse* netc_tb_eseqr_srse_t
Egress Sequence Recovery table sequence recovery state element.

typedef struct *_netc_tb_eseqr_req_data* netc_tb_eseqr_req_data_t
Egress Sequence Recovery table request data buffer.

typedef struct *_netc_tb_eseqr_rsp_data* netc_tb_eseqr_rsp_data_t
Egress Sequence Recovery table request response data buffer.

typedef struct *_netc_tb_eseqr_data* netc_tb_eseqr_data_t
Egress Sequence Recovery table data buffer.

typedef struct *_netc_tb_eseqr_config* netc_tb_eseqr_config_t
Egress Sequence Recovery table entry config.

typedef enum *_netc_tb_tgs_entry_id* netc_tb_tgs_entry_id_t
Time Gate Scheduling table entry ID for switch and ENETC.

typedef enum *_netc_tb_tgs_gate_type* netc_tb_tgs_gate_type_t
Administrative gate operation type.

typedef struct *_netc_tgs_gate_entry* netc_tgs_gate_entry_t
Defines the Time Gate Scheduling gate control entry structure.

typedef struct *_netc_tb_tgs_cfge* netc_tb_tgs_cfge_t
Time Gate Scheduling table config element.

typedef struct *_netc_tb_tgs_olse* netc_tb_tgs_olse_t
Time Gate Scheduling table statistic element.

typedef struct *_netc_tb_tgs_req_data* netc_tb_tgs_req_data_t
Time Gate Scheduling table request data buffer.

typedef struct *_netc_tb_tgs_rsp_data* netc_tb_tgs_rsp_data_t
Time Gate Scheduling table request response data buffer.

typedef struct *_netc_tb_tgs_data* netc_tb_tgs_data_t
Time Gate Scheduling table data buffer, set with max size.

typedef struct *_netc_tb_tgs_gcl* netc_tb_tgs_gcl_t
Time Gate Scheduling table entry gate control list structure.

typedef enum *_netc_tb_et_efm_mode* netc_tb_et_efm_mode_t
Egress Frame Modification entry mode.

typedef enum *_netc_tb_et_esq_act* netc_tb_et_esq_act_t
Egress Sequence Actions.

typedef enum *_netc_tb_et_ec_act* netc_tb_et_ec_act_t
Egress Counter Action.

typedef struct *_netc_tb_et_cfge* netc_tb_et_cfge_t
Egress Treatment table config element.

typedef struct *_netc_tb_et_req_data* netc_tb_et_req_data_t
Egress Treatment table request data buffer.

typedef struct *_netc_tb_et_rsp_data* netc_tb_et_rsp_data_t
Egress Treatment table request response data buffer.

typedef struct *_netc_tb_et_data* netc_tb_et_data_t
Egress Treatment table data buffer.

typedef struct *_netc_tb_et_config* netc_tb_et_config_t
Egress Treatment table entry config.

typedef enum *_netc_tb_etmcq_update_action* netc_tb_etmcq_update_action_t
ETM Class Queue table Update Actions.

typedef struct *_netc_tb_etmcq_cfge* netc_tb_etmcq_cfge_t
ETM Class Queue table config element.

typedef struct *_netc_tb_etmcq_stse* netc_tb_etmcq_stse_t
ETM Class Queue table statistic element.

typedef struct *_netc_tb_etmcq_req_data* netc_tb_etmcq_req_data_t
ETM Class Queue table request data buffer.

typedef struct *_netc_tb_etmcq_rsp_data* netc_tb_etmcq_rsp_data_t
ETM Class Queue table request response data buffer.

typedef struct *_netc_tb_etmcq_data* netc_tb_etmcq_data_t
ETM Class Queue table data buffer.

typedef struct *_netc_tb_etmcq_config* netc_tb_etmcq_config_t
ETM Class Queue table entry config.

typedef enum *_netc_tb_etmcs_entry_id* netc_tb_etmcs_entry_id_t
ETM Class Scheduler table entry ID.

typedef enum *_netc_tb_etmcs_ca_assg* netc_tb_etmcs_ca_assg_t
ETM Class Scheduler table Class queue assignment to scheduler inputs mode.

typedef struct *_netc_tb_etmcs_cfge* netc_tb_etmcs_cfge_t
ETM Class Scheduler table config element.

typedef struct *_netc_tb_etmcs_req_data* netc_tb_etmcs_req_data_t
ETM Class Scheduler table request data buffer.

typedef struct *_netc_tb_etmcs_rsp_data* netc_tb_etmcs_rsp_data_t
ETM Class Scheduler table request response data buffer.

typedef struct *_netc_tb_etmcs_data* netc_tb_etmcs_data_t
ETM Class Scheduler table data buffer.

typedef struct *_netc_tb_etmcs_config* netc_tb_etmcs_config_t
ETM Class Scheduler table entry config.

typedef struct *_netc_tb_etmcg_cfge* netc_tb_etmcg_cfge_t
ETM Congestion Group table config element.

typedef struct *_netc_tb_etmcg_stse* netc_tb_etmcg_stse_t
ETM Congestion Group table statistic element.

typedef struct *_netc_tb_etmcg_req_data* netc_tb_etmcg_req_data_t
ETM Congestion Group table request data buffer.

typedef struct *_netc_tb_etmcg_rsp_data* netc_tb_etmcg_rsp_data_t
ETM Congestion Group table request response data buffer.

typedef struct *_netc_tb_etmcg_data* netc_tb_etmcg_data_t
ETM Congestion Group table data buffer.

typedef struct *_netc_tb_etmcg_config* netc_tb_etmcg_config_t
ETM Congestion Group table entry config.

typedef struct *_netc_tb_ec_stse* netc_tb_ec_stse_t
Egress Count table statistic element.

typedef struct *_netc_tb_ec_req_data* netc_tb_ec_req_data_t
Egress Count table request data buffer.

typedef struct *_netc_tb_ec_rsp_data* netc_tb_ec_rsp_data_t
Egress Count table request response data buffer.

typedef struct *_netc_tb_ec_data* netc_tb_ec_data_t
Egress Count table data buffer.

typedef enum *_netc_tb_bp_fc_cfg* netc_tb_bp_fc_cfg_t
Buffer Pool Flow Control (FC) Configuration.

typedef struct *_netc_tb_bp_cfge* netc_tb_bp_cfge_t
Buffer Pool table config element.

typedef struct *_netc_tb_bp_bpse* netc_tb_bp_bpse_t
Buffer Pool table State Element Data.

typedef struct *_netc_tb_bp_req_data* netc_tb_bp_req_data_t
Buffer Pool table request data buffer.

typedef struct *_netc_tb_bp_rsp_data* netc_tb_bp_rsp_data_t
Buffer Pool table request response data buffer.

typedef struct *_netc_tb_bp_data* netc_tb_bp_data_t
Buffer Pool table data buffer.

typedef struct *_netc_tb_bp_config* netc_tb_bp_config_t
Buffer Pool table entry config.

typedef struct *_netc_tb_sbp_cfge* netc_tb_sbp_cfge_t
Shared Buffer Pool table config element.

typedef struct *_netc_tb_sbp_sbpse* netc_tb_sbp_sbpse_t
Shared Buffer Pool table State Element Data.

typedef struct *_netc_tb_sbp_req_data* netc_tb_sbp_req_data_t
Shared Buffer Pool table request data buffer.

typedef struct *_netc_tb_sbp_rsp_data* netc_tb_sbp_rsp_data_t
Shared Buffer Pool table request response data buffer.

typedef struct *_netc_tb_sbp_data* netc_tb_sbp_data_t
Shared Buffer Pool table data buffer.

typedef struct *_netc_tb_sbp_config* netc_tb_sbp_config_t
Shared Buffer Pool table entry config.

typedef union *_netc_tb_data_buffer* netc_tb_data_buffer_t
Table common data buffer.

typedef struct *_netc_cbdr_hw* netc_cbdr_hw_t
Register group for SI/Switch command bd ring.

typedef struct *_netc_cbdr_handle* netc_cbdr_handle_t

Handle for common part of EP/Switch NTMP.

status_t NETC_CmdBDRInit(*netc_cbdr_hw_t* *base, const *netc_cmd_bdr_config_t* *config)

Initialize the command BD ring.

Parameters

- base –
- config –

Returns
kStatus_Success

Returns
kStatus_Fail

status_t NETC_CmdBDRDeinit(*netc_cbdr_hw_t* *base)

Deinitialize the command BD ring.

Parameters

- base –

Returns
kStatus_Success

status_t NETC_CmdBDSendCommand(*netc_cbdr_hw_t* *base, *netc_cmd_bdr_t* *cbdr, *netc_cmd_bd_t* *cbd, *netc_cbd_version_t* version)

Send the Command Buffer Descriptor to operate on a NTMP table.

Parameters

- base –
- cbdr –
- cbd –
- version –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddIPFTableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_ipf_config_t* *config, uint32_t *entryID)

Add entry into the ingress Port Filter Table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_UpdateIPFTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_ipf_cfge_t* *cfg)

Update entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_QueryIPFTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_ipf_config_t* *config)

Query entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_DelIPFTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete an entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_ResetIPFMatchCounter(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Reset the counter of an ingress port filter Table entry.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_GetIPFMatchedCount(*netc_cbdr_handle_t* *handle, uint32_t entryID, uint64_t *count)

Get the matched count of an ingress port filter Table entry.

Parameters

- handle –
- entryID –

- count –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddISITableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_isi_config_t* *config, uint32_t *entryID)

Add entry into Ingress Stream Identification table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_DelISITableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete an entry in Ingress stream identification table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_QueryISITableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_isi_config_t* *config)

Query Ingress Stream Identification table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_QueryISITableEntryWithKey(*netc_cbdr_handle_t* *handle, *netc_tb_isi_keye_t* *keye, *netc_tb_isi_rsp_data_t* *rsp)

Query Ingress Stream Identification table with key.

Parameters

- handle –
- keye –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddOrUpdateISTableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_is_config_t* *config, bool isAdd)

Add or update entry in Ingress Stream table.

Parameters

- handle –
- config –
- isAdd –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_QueryISTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_is_config_t* *config)

Query Ingress Stream table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_DelISTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete an entry in Ingress stream table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddISFTableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_isf_config_t* *config, uint32_t *entryID)

Add entry into ingress stream filter table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_UpdateISFTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_isf_cfge_t* *cfg)

Update entry into ingress stream filter table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_DelISFTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete an entry in Ingress stream filter table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_QueryISFTableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_isf_keye_t* *keye, *netc_tb_isf_rsp_data_t* *rsp)

Query entry from the Ingress stream filter table.

Parameters

- handle –
- keye –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddISCTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Add entry in ingress stream count table.

Parameters

- handle –
- entryID –

Returns
status_t

status_t NETC_GetISCStatistic(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_isc_stse_t* *statistic)

Get ingress stream count statistic.

Parameters

- handle –
- entryID –
- statistic –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_ResetISCStatistic(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Reset the count of the ingress stream count.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddOrUpdateSGITableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_sgi_config_t* *config, bool isAdd)

Add or update entry in stream gate instance table.

Parameters

- handle –
- config –
- isAdd –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_ResetIRXOEXSGITableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Reset IRX and OEX flags in stream gate instance entry.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_DelSGITableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete entry in the stream gate instance table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_GetSGIState(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_sgise_t* *statis)

Get statistic of specified stream gate instance table entry.

Parameters

- handle –
- entryID –
- statis –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_QuerySGITableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_config_t* *config)

Query entry from the stream gate instance table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddSGCLTableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_sgcl_gcl_t* *config)

Add entry into Stream Gate Control List Table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_DelSGCLTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete entry of Stream Gate Control List Table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_GetSGCLGateList(*netc_cbdr_handle_t* *handle, *netc_tb_sgcl_gcl_t* *gcl, uint32_t length)

Get Stream Gate Control List Table entry gate control list.

Parameters

- handle –
- gcl –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_GetSGCLState(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_sgcl_sgclse_t* *state)

Get state (ref count) for Stream Gate Control List table entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_QueryRPTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_rp_rsp_data_t* *rsp)

Query entry from the Rate Policer table.

Parameters

- handle –
- entryID –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_AddOrUpdateRPTableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_rp_config_t* *config, *netc_tb_cmd_t* cmd)

Add or update entry in Rate Policer table.

Parameters

- handle –
- config –

- cmd –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_DelRPTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete entry in the Rate Policer table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_GetRPStatistic(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_rp_stse_t* *statis)

Get statistic of specified Rate Policer table entry.

Parameters

- handle –
- entryID –
- statis –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_ResetMRRPTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Reset mark red parameter of specified Rate Policer table entry.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_ConfigTGSAdminList(*netc_cbdr_handle_t* *handle, *netc_tb_tgs_gcl_t* *config)

Config the QBV (Time Gate Scheduling)

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t NETC_GetTGSOperationList(*netc_cbdr_handle_t* *handle, *netc_tb_tgs_gcl_t* *gcl, uint32_t length)

Get time gate table operation list.

Parameters

- handle –
- gcl –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

NETC_FD_EID_ENCODE_OPTION_0(entryId)

Frame Modification Entry ID encode options.

Note: sqta should be netc_fm_sqt_act_t type, vuda should be netc_fm_vlan_ud_act_t type and vara should be netc_fm_vlan_ar_act_t type.

NETC_FD_EID_ENCODE_OPTION_1(sqta, vuda)

NETC_FD_EID_ENCODE_OPTION_2(vara, vid)

NETC_ISI_VLAN_FRAME_KEY(valid, pcp, vid)

2 Bytes VLAN field which may added to the frame Key

NETC_TB_SGCL_MAX_ENTRY

Stream Gate Control List Table maximum gate control list length.

NETC_TB_FMD_UPDATE_CONFIG_LENGTH(x)

TFrame Modification Data table update config Data Buffer length, x is the number of update data bytes.

NETC_TB_TGS_MAX_ENTRY

Time Gate Scheduling Table maximum gate control list length (TGSTCAPR[MAX_GCL_LEN])

NETC_TB_ETM_CQ_ENTRY_ID(portID, cqID)

ETM Class Queue table entry ID macro, cqID is represents the Class Queue ID ,rang in 0 ~ 7, portID is Switch ID, rang in 0 ~ 4.

NETC_TB_ETM_CG_ENTRY_ID(portID, cgID)

ETM Congestion Group table entry ID macro, cgID is represents the Congestion Group ID ,rang in 0 ~ 7, portID is Switch ID, rang in 0 ~ 4.

NETC_TB_BP_THRESH(mant, exp)

Buffer pool and shared buffer pool threshold macro, the threshold = MANT*2^EXP, uint is internal memory words (avergae of 20 bytes each)

union _netc_cmd_bd

#include <fsl_netc.h> The Switch/SI command BD data structure.

Public Members

struct *_netc_cmd_bd* req

struct *_netc_cmd_bd* resp

struct *_netc_cmd_bd* generic

struct __netc__cmd__bdr__config

#include <fsl_netc.h> Configuration for the Switch/SI command BD Ring Configuration.

Public Members

netc_cmd_bd_t *bdBase

BDR base address which shall be 128 bytes aligned

uint16_t bdLength

Size of BD ring which shall be multiple of 8 BD

bool enCompInt

Enable/Disable command BD completion interrupt

struct __netc__cmd__bdr

#include <fsl_netc.h> The Switch/SI command BD ring handle data structure.

Public Members

netc_cmd_bd_t *bdBase

BDR base address which shall be 128 bytes aligned

uint16_t bdLength

Size of BD ring

uint16_t producerIndex

Current index for execution.

uint16_t cleanIndex

Current index for cleaning.

bool bdrEnable

Current command BD ring is enable or not.

struct __netc__tb__common__header

#include <fsl_netc.h> Table request data buffer common header.

Public Members

uint32_t updateActions

Update Actions

uint32_t queryActions

Query Actions

struct __netc__tb__ipf__keye

#include <fsl_netc.h> Ingress Port Filter key element.

Public Members

uint16_t precedence

Precedence value of an entry

struct *_netc_tb_ipf_keye* frameAttr

Frame Attribute flags

uint16_t frameAttrMask
: Frame attribute mask, set with OR of netc_tb_ipf_attr_mask_t

uint16_t dscp
: Differentiated Services Code Point

uint16_t dscpMask
: Differentiated Services Code Point Mask

uint16_t srcPort
: Source Port ID

uint16_t srcPortMask
: Source Port ID Mask

uint16_t outerVlanTCI
: Outer VLAN Tag Control Information

uint16_t outerVlanTCIMask
: Outer VLAN Tag Control Information Mask

uint8_t dmac[6]
: Destination MAC Address

uint8_t dmacMask[6]
: Destination MAC Address Mask

uint8_t smac[6]
: Source MAC Address

uint8_t smacMask[6]
: Source MAC Address Mask

uint16_t innerVlanTCI
: Inner VLAN Tag Control Information

uint16_t innerVlanTCIMask
: Inner VLAN Tag Control Information Mask

uint16_t etherType
: 2-byte EtherType

uint16_t etherTypeMask
: EtherType Mask

uint8_t IPProtocol
: IP Protocol

uint8_t IPProtocolMask
: IP Protocol Mask

uint8_t srcIPAddr[16]
: IP Source Address, Bits 127-0: IPv6 source address, Bits 127-96: IPv4 source address

uint8_t srcIPAddrMask[16]
: IP Source Address Mask

uint16_t l4SrcPort
: L4 Source Port

uint16_t l4SrcPortMask
: L4 Source Port Mask

uint8_t destIPAddr[16]
: IP Destination Address, Bits 127-0: IPv6 source address, Bits 127-96: IPv4 source address

uint8_t destIPAddrMask[16]
: IP Destination Address Mask

uint16_t l4DestPort
: L4 Destination Port

uint16_t l4DestPortMask
: L4 Destination Port Mask

struct __netc_tb_ipf_cfge
: *#include <fsl_netc.h>* Ingress port filter config element.

Public Members

uint32_t ipv
: Internal Priority Value

uint32_t oipv
: Overwrite IPV

uint32_t dr
: Drop Resilience

uint32_t odr
: Overwrite DR

netc_tb_ipf_forward_action_t fltfa
: Filter Forwarding action.

uint32_t imire
: Ingress Mirroring Enable

uint32_t wolte
: Wake-onLAN trigger enable

netc_tb_ipf_filter_action_t flta
: FIlter Action.

uint32_t rpr
: Relative Precedent Resolution

uint32_t ctd
: Cut through disable.

netc_host_reason_t hr
: Host Reason metadata when frame is redirected/copied to the switch management port

uint32_t timecape
: Timestam capture enable

uint32_t rrt
: Report Receive Timestamp

uint32_t fltaTgt
: Target for selected switch forwarding action or filter action

struct __netc_tb_ipf_stse
: *#include <fsl_netc.h>* Ingress port filter statistic element.

Public Members

uint32_t matchCount[2]
A count of how many times this entry has been matched.

struct __netc_tb_ipf_req_data
#include <fsl_netc.h> Ingress port filter table entry config.

struct __netc_tb_ipf_rsp_data
#include <fsl_netc.h> Ingress port filter table response data.

Public Members

uint32_t entryID
Present only for commands which perform a query

netc_tb_ipf_keye_t keye
Present only for commands which perform a query

netc_tb_ipf_stse_t stse
Present only for commands which perform a query

netc_tb_ipf_cfge_t cfge
Present only for commands which perform a query

struct __netc_tb_ipf_data
#include <fsl_netc.h> Ingress Port filter table data buffer.

struct __netc_tb_ipf_config
#include <fsl_netc.h> Ingress Port filter entry config.

struct __netc_tb_isi_keye
#include <fsl_netc.h> Stream identification table key element.

Public Members

netc_tb_isi_key_type keyType
Define the key type used for the current isi entry

uint8_t srcPortID
Source Port ID, used when kc portp filed is 1. Only for SWITCH

uint8_t spm
Source Port Masquerading, used when kc spm filed is 1. Only for SWITCH

uint8_t framekey[16]
Frame portion of the key.

struct __netc_tb_isi_cfge
#include <fsl_netc.h> Stream identification table config element.

Public Members

uint32_t iSEID
Ingress stream entry ID, 0xFFFFFFFF means NULL

struct __netc_tb_isi_req_data
#include <fsl_netc.h> Stream identification table request data buffer.

struct __netc_tb_isi_rsp_data
#include <fsl_netc.h> Stream identification table request response data buffer.

Public Members

uint32_t entryID
: Only present for query command

netc_tb_isi_keye_t keye
: Only present for query command

netc_tb_isi_cfge_t cfge
: Only present for query command

struct __netc_tb_isi_data
: *#include <fsl_netc.h>* Stream identification table data buffer.

struct __netc_tb_isi_config
: *#include <fsl_netc.h>* Stream identification table entry config.

struct __netc_tb_is_cfge
: *#include <fsl_netc.h>* Ingress Stream table config element.

Public Members

uint32_t sfe
: Stream Filtering Enable

uint32_t ipv
: Internal Priority Value, active when opiv is set to 1

uint32_t oipv
: Override internal priority value

uint32_t dr
: Drop Resilience, active when odr is set to 1

uint32_t odr
: Overwrite DR

uint32_t imire
: Ingress Mirroring Enable, not applicable to ENETC

uint32_t timecape
: Timestamp Capture Enable, not applicable to ENETC

uint32_t sppd
: Source Port Pruning Disable, not applicable to ENETC

netc_tb_is_isq_action_t isqa
: Ingress Sequence Action, not applicable to ENETC

uint32_t orp
: Override Rate Policer ID

uint32_t osgi
: Override stream gate instance entry id (default is NULL)

netc_host_reason_t hr
: Host Reason when frame is redirected (fa = 01b) to the switch management port or copied to the switch management port (fa = 100b or 101b), value specified has to be a software defined Host Reason (8-15).

netc_tb_is_forward_action_t fa
: Forwad Option

netc_tc_sdu_type_t sduType

Service Data Unit Type to user for MSDU

uint32_t msdu

Maximum Service Data Unit

uint32_t ifmeLenChange

Ingress Frame Modification Entry Frame Length Change, specified in unit of bytes using a 2's complement notation

uint32_t eport

Egress Port which need do egress packet processing, active when oeteid is set to 1, not applicable to ENETC

netc_tb_is_oeteid_mode_t oETEID

Override ET_EID (Egress Treatment table entry, which specified egress packet processing actions)

netc_tb_is_ctd_mode_t ctd

Cut-Through Disable mode, valid if fa = 010b ~ 101b

uint32_t isqEID

Ingress Sequence Generation Entry ID, Valid when isqa is set to 1. 0xFFFF_FFFF is NULL. Not applicable to ENETC

uint32_t rpEID

Rate Policer Entry ID, Valid when orp =1. 0xFFFF_FFFF is NULL

uint32_t sgiEID

Stream Gate Instance Entry ID, Valid when osgi =1. 0xFFFF_FFFF is NULL

uint32_t ifmEID

Ingress Frame Modification Entry ID. 0xFFFF_FFFF is NULL

uint32_t etEID

Base Egress Treatment Entry ID for primary Egress Treatment group, Valid alid if fa = 010b ~ 101b. 0xFFFF_FFFF is NULL. Not applicable to ENETC

uint32_t iscEID

Ingress Stream counter Index. 0xFFFF_FFFF is NULL.

uint32_t ePortBitmap

Egress Port bitmap, identifies the ports to which the frame is to be forwarding or ET applicability port bitmap when oETEID = 10b. Not applicable to ENETC

uint32_t siMap

Station Interface Map, only valid for ENETC function when fa field is set to 10b

struct __netc_tb_is_req_data

#include <fsl_netc.h> Ingress Stream table request data buffer.

Public Members

netc_tb_is_cfge_t cfge

Only perform for update or add command

struct __netc_tb_is_rsp_data

#include <fsl_netc.h> Ingress Stream table request response data buffer.

Public Members

uint32_t entryID
: Only perform for query command

netc_tb_is_cfge_t cfge
: Only perform for query command

struct __netc_tb_is_data
: *#include <fsl_netc.h>* Ingress Stream table data buffer.

struct __netc_tb_is_config
: *#include <fsl_netc.h>* Ingress Stream table entry config.

struct __netc_tb_isf_keye
: *#include <fsl_netc.h>* Ingress Stream Filter table key element.

Public Members

uint32_t isEID
: Ingress Stream Entry ID

uint8_t pcp
: Priority Code Point, Outer VLAN TAG PCP of the received frame

struct __netc_tb_isf_cfge
: *#include <fsl_netc.h>* Ingress Stream Filter table config element.

Public Members

uint32_t ipv
: Internal Priority Value, active when opiv is set to 1

uint32_t oipv
: Override internal priority value

uint32_t dr
: Drop Resilience, active when odr is set to 1

uint32_t odr
: Overwrite DR

uint32_t imire
: Ingress Mirroring Enable, not applicable to ENETC

uint32_t timecape
: Timestamp Capture Enable, not applicable to ENETC

uint32_t osgi
: Override stream gate instance entry id

uint32_t ctd
: Cut-Through Disable, will disable cut-through for all destined ports when set 1, not applicable to ENETC

uint32_t orp
: Override Rate Policer (instance) ID

netc_tc_sdu_type_t sduType
: Service Data Unit Type to user for MSDU

uint32_t msdu
: Maximum Service Data Unit

uint32_t rpEID
: Rate Policer Entry ID, Valid when orp =1. 0xFFFF_FFFF is NULL

uint32_t sgiEID
: Stream Gate Instance Entry ID, Valid when osgi =1. 0xFFFF_FFFF is NULL

uint32_t iscEID
: Ingress Stream counter Index. 0xFFFF_FFFF is NULL.

struct __netc_tb_isf_req_data
: *#include <fsl_netc.h>* Ingress Stream Filter table request data buffer.

Public Members

netc_tb_isf_cfge_t cfge
: Only perform for update or add command

struct __netc_tb_isf_rsp_data
: *#include <fsl_netc.h>* Ingress Stream Filter table request response data buffer.

Public Members

uint32_t entryID
: Only perform for query command

netc_tb_isf_keye_t keye
: Only perform for query command

netc_tb_isf_cfge_t cfge
: Only perform for query command

struct __netc_tb_isf_data
: *#include <fsl_netc.h>* Ingress Stream Filter table data buffer.

struct __netc_tb_isf_config
: *#include <fsl_netc.h>* Ingress Stream Filter table entry config.

struct __netc_tb_rp_cfge
: *#include <fsl_netc.h>* Rate Policer table config element.

Public Members

uint32_t cir
: Committed information Rate

uint32_t cbs
: Commited Burst Size

uint32_t eir
: Excess information Rate

uint32_t ebs
: Excess Burst Size

uint32_t mren
: Mark All Frames Red Enable, Not valid when ndor=1

uint32_t doy
: Drop on Yellow enable

uint32_t cm
: Color mode, 0b = Color blind, 1b = Color aware

uint32_t cf
: Coupling flag, enables coupling the Committed (C) bucket and Excess (E) bucket

uint32_t ndor
: No drop on red

netc_tb_rp_sdu_type_t sduType
: Service Data Unit Type

uint32_t ___pad0___
: Reserved

struct __netc__tb__rp__fee
: *#include <fsl_netc.h>* Rate Policer table Function Enable element.

Public Members

uint8_t fen
: Function Enable

uint8_t ___pad0___
: Reserved

struct __netc__tb__rp__pse
: *#include <fsl_netc.h>* Rate Policer table Policer State element.

Public Members

uint8_t mr
: Mark Red Flag

uint8_t res0
: Reserved

struct __netc__tb__rp__stse
: *#include <fsl_netc.h>* Rate Policer table statistic element.

Public Members

uint32_t byteCount[2]
: Number of bytes received by the rate policer instance

uint32_t dropFrames[2]
: Number of frames dropped by the rate policer instance

uint32_t dr0GrnFrames[2]
: Number of frames marked green with DR=0 by the rate policer instance

uint32_t dr1GrnFrames[2]
: Number of frames marked green with DR=1 by the rate policer instance

uint32_t dr2GrnFrames[2]
: Number of frames marked yellow with DR=2 by the rate policer instance

uint32_t remarkYlwFrames[2]
Number of frames re-marked from green to yellow by the rate policer instance

uint32_t dr3RedFrames[2]
Number of frames marked red with DR=3 by the rate policer instance

uint32_t remarkRedFrames[2]
Number of frames re-marked from green or yellow to red by the rate policer instance

uint32_t lts
Last timestamp

uint32_t bci
Committed token bucket contents, integer portion (31 bits)

uint32_t bcs
Committed token bucket sign bit (1 bit)

uint32_t bcf
Committed token bucket contents, fractional portion (31 bits)

uint32_t bei
Excess token bucket contents, integer portion (32 bits)

uint32_t bef
Excess token bucket contents, fractional portion (31 bits)

uint32_t bes
Committed token bucket sign bit

struct __netc_tb_rp_req_data
#include <fsl_netc.h> Rate Policer table request data buffer.

struct __netc_tb_rp_rsp_data
#include <fsl_netc.h> Rate Policer table request response data buffer.

Public Members

uint32_t entryID
Present only for commands which perform a query

netc_tb_rp_stse_t stse
Present only for commands which perform a query

struct __netc_tb_rp_data
#include <fsl_netc.h> Rate Policer table data buffer.

struct __netc_tb_rp_config
#include <fsl_netc.h> Rate Policer table entry config.

struct __netc_tb_isc_stse
#include <fsl_netc.h> Ingress Stream Count table statistic element.

Public Members

uint32_t rxCount
Receive Count

uint32_t msduDropCount
MSDU Drop Count

uint32_t policerDropCount
: Policer Drop Count

uint32_t sgDropCount
: Stream Gating Drop Count

struct __netc_tb_isc_req_data
: *#include <fsl_netc.h>* Ingress Stream Count table request data buffer.

struct __netc_tb_isc_rsp_data
: *#include <fsl_netc.h>* Ingress Stream Count table request response data buffer.

struct __netc_tb_isc_data
: *#include <fsl_netc.h>* Ingress Stream Count table data buffer.

struct __netc_tb_sgi_cfge
: *#include <fsl_netc.h>* Stream Gate Instance table config element.

Public Members

uint8_t oexen
: Octets Exceeded (Gate Closed Due To Octets Exceeded function) Enable

uint8_t irxen
: Invalid Receive (Gate Closed Due To Invalid Rx) Enable

netc_tb_sgi_sdu_type_t sduType
: The type of PDU/SDU for Interval Octets Maximum check for Gate Entry

struct __netc_tb_sgi_acfge
: *#include <fsl_netc.h>* Stream Gate Instance table Admin Configuration element.

Public Members

uint32_t adminSgclEID
: Administrative Stream Gate Control List Entry ID, 0xFFFFFFFF is NULL

uint32_t adminBaseTime[2]
: Admin Base Time

uint32_t adminCycleTimeExt
: Admin Cycle Time Extension

struct __netc_tb_sgi_icfge
: *#include <fsl_netc.h>* Stream Gate Instance table Initial Configuration element.

Public Members

uint8_t ipv
: Internal Priority Value (IPV), Valid if oipv is 1

uint8_t oipv
: Override frame IPV, otherwise the IPV value is determined by the stream gate control list entry

uint8_t gst
: Specifies Gate State before the administrative stream gate control list takes affect, 0b = Closed; 1b = Open

uint8_t ctd
: Specifies Cut Through disable status before the administrative stream gate control list takes affect , Not applicable to ENETC function

struct __netc_tb_sgi_sgise
: *#include <fsl_netc.h>* Stream Gate Instance table stream gate instance state element.

Public Members

uint32_t operSgclEID
: Operational Stream Gate Control List Entry ID

uint32_t configChangeTime[2]
: Configuration Change Time

uint32_t operBaseTime[2]
: Operational Base Time

uint32_t operCycleTimeExt
: Oper Cycle Time Extension

uint32_t oex
: Octets Exceeded Flag

uint32_t irx
: Invalid Receive Flag

netc_tb_sgi_state_t state
: Current Gate Instance State

struct __netc_tb_sgi_req_data
: *#include <fsl_netc.h>* Stream Gate Instance table request data buffer.

struct __netc_tb_sgi_rsp_data
: *#include <fsl_netc.h>* Stream Gate Instance table request response data buffer.

struct __netc_tb_sgi_data
: *#include <fsl_netc.h>* Stream Gate Instance table data buffer.

struct __netc_tb_sgi_config
: *#include <fsl_netc.h>* Stream Gate Instance table entry config.

struct __netc_sgcl_gate_entry
: *#include <fsl_netc.h>* Defines the Stream Gate Control entry structure.

Public Members

uint32_t timeInterval
: Time Interval for Gate Entry

uint32_t iom
: Interval Octets Maximum for Gate Entry, specifies the maximum bytes (octets) allowed to pass (open), valid if iomen = 1

uint32_t ipv
: Internal Priority Value for Gate Entry

uint32_t oipv
: Override Internal Priority Value for Gate Entry

uint32_t ctd
: Cut Through Disable for Gate Entry

uint32_t iomen
: Interval Octet Maximum Enabled for Gate Entry, 0b = Don'track count, 1b = Track count

uint32_t gtst
: Gate State for Gate Entry, 0b = Closed; 1b = Open

struct _netc_tb_sgcl_cfge
: *#include <fsl_netc.h>* Stream Gate Control List table config element.

Public Members

uint32_t cycleTime
: Cycle Time

uint8_t listLength
: List Length

uint16_t extOipv
: Extension (means the stream gate control list ends and before cycleTime restarts) Override Internal Priority Value

uint16_t extIpv
: List Extension Internal Priority Value, valid if extOipv = 1

uint16_t extCtd
: Extension Cut Through Disabled, 0b = No action, 1b = Disabled

uint16_t extGtst
: Extension Gate State, 0b = closed, 1b = Open

struct _netc_tb_sgcl_sgclse
: *#include <fsl_netc.h>* Stream Gate Control List table Stream Gate Control List State element.

Public Members

uint8_t refCount
: Reference Count, 1 indicates that the gate control list is an administrative or an operational gate control list in a stream gate instance

struct _netc_tb_sgcl_req_data
: *#include <fsl_netc.h>* Stream Gate Control List table request data buffer.

struct _netc_tb_sgcl_rsp_data
: *#include <fsl_netc.h>* Stream Gate Control List table request response data buffer.

struct _netc_tb_sgcl_data
: *#include <fsl_netc.h>* Stream Gate Control List table data buffer.

struct _netc_tb_sgcl_gcl
: *#include <fsl_netc.h>* Stream Gate Control List table entry gate control list structure.

Public Members

uint16_t extOipv
: Extension (means the stream gate control list ends and before cycleTime restarts) Override Internal Priority Value

uint16_t extIpv
: List Extension Internal Priority Value, valid if extOipv = 1

uint16_t extCtd
: Extension Cut Through Disabled, 0b = No action, 1b = Disabled

uint16_t extGtst
: Extension Gate State, 0b = closed, 1b = Open

uint32_t cycleTime
: Cycle Time

uint32_t numEntries
: Control List entry numbers

netc_sgcl_gate_entry_t *gcList
: Pointer to stream gate control list array

struct __netc_tb_fm_cfge
: *#include <fsl_netc.h>* Frame Modification table config element.

Public Members

netc_tb_fm_layer2_act_t l2Act
: Layer 2 Actions

netc_tb_fm_mac_header_act_t macHdrAct
: Layer 2 Header MAC Actions

netc_tb_fm_vlan_header_act_t vlanHdrAct
: Layer 2 VLAN Actions

netc_tb_fm_outer_vid_act_t outerVidAct
: Outer VID Actions

netc_tb_fm_sqt_act_t sqtAct
: Sequence Tag Action, Not applicable for ingress frame modifications

uint16_t smacPort
: Source MAC Address Register Port, valid if macHdrAct=010b,011b,100b

uint8_t dmac[6]
: Destination MAC Address, valid if macHdrAct = 011b,101b

uint32_t outerVlanID
: Outer VLAN VID, valid if outerVidAct = 01b

uint32_t outerVlanPcp
: Outer VLAN PCP, valid if outerPcpAct = 01b

uint32_t outerVlanDei
: Outer VLAN DEI, valid if outerDeiAct = 01b

netc_tb_fm_outer_tpid_act_t outerTpidAct
: Outer TPID action

netc_tb_fm_outer_pcp_act_t outerPcpAct
: Outer PCP action

netc_tb_fm_outer_dei_act_t outerDeiAct
: Outer DEI action

netc_tb_fm_payload_act_t pldAct
: Payload Actions, Not applicable for ingress frame modifications

uint8_t pldOffset
: Payload Offset, valid if outerPldAct = 010b

uint16_t fmdBytes
: Frame Modification Bytes, valid if outerPldAct = 001b,010b or l2Act = 1b

uint32_t fmdEID
: Frame Modification Data Entry ID, valid if outerPldAct = 001b,010b or l2Act = 1b. 0xFFFF is null pointer

struct __netc_tb_fm_req_data
: *#include <fsl_netc.h>* Frame Modification table request data buffer.

struct __netc_tb_fm_rsp_data
: *#include <fsl_netc.h>* Frame Modification table request response data buffer.

struct __netc_tb_fm_data
: *#include <fsl_netc.h>* Frame Modification table data buffer.

struct __netc_tb_fm_config
: *#include <fsl_netc.h>* Frame Modification table entry config.

struct __netc_tb_fmd_req_data
: *#include <fsl_netc.h>* Frame Modification Data table request data buffer.

Public Members

uint8_t cfge[]
: Configuration Element Data size is variable

struct __netc_tb_fmd_rsp_data
: *#include <fsl_netc.h>* Frame Modification Data table request response data buffer.

Public Members

uint8_t cfge[]
: Configuration Element Data size is variable

struct __netc_tb_fmd_data
: *#include <fsl_netc.h>* Frame Modification Data table data buffer.

struct __netc_tb_fmd_update_config
: *#include <fsl_netc.h>* Frame Modification data table entry update config.

Public Members

uint32_t res
: Hold for request->commonHeader

uint8_t cfge[]
: Configuration Element Data size is variable

struct __netc_tb_fmd_query_buffer
: *#include <fsl_netc.h>* Frame Modification data table entry query data buffer.

Public Members

uint32_t entryID
: EntryID of the queried entry

uint8_t cfge[]
: Configuration Element Data size is variable

struct __netc_tb_vf_keye
: *#include <fsl_netc.h>* Vlan Filter table key element.

struct __netc_tb_vf_cfge
: *#include <fsl_netc.h>* Vlan Filter table config element.

Public Members

uint32_t portMembership
: Port Membership Bitmap

uint32_t stgID
: Spanning Tree Group Member ID

uint32_t fid
: Filtering ID

uint32_t mlo
: MAC Learning Options

uint32_t mfo
: MAC Forwarding Options

uint32_t ipmfe
: IP Multicast Filtering Enable

uint32_t ipmfle
: IP Multicast Flooding Enable

uint32_t etaPortBitmap
: Egress Treatment Applicability Port Bitmap for the secondary Egress Treatment group

uint32_t baseETEID
: Base Egress Treatment Entry ID for the secondary Egress Treatment group

struct __netc_tb_vf_search_criteria
: *#include <fsl_netc.h>* Vlan Filter table search criteria format.

Public Members

uint32_t resumeEntryId
: Resume Entry ID, when starting a search, pass the NULL Entry ID.

struct __netc_tb_vf_req_data
: *#include <fsl_netc.h>* Vlan Filter table request data buffer.

Public Members

netc_tb_vf_cfge_t cfge
: Present only for update or add commands

struct __netc_tb_vf_rsp_data
: *#include <fsl_netc.h>* Vlan Filter table request response data buffer.

Public Members

uint32_t status
: Present only for query command with search access method

uint32_t entryID
: Present only for query command

netc_tb_vf_keye_t keye
: Present only for query command

netc_tb_vf_cfge_t cfge
: Present only for query command

struct __netc_tb_vf_data
: *#include <fsl_netc.h>* Vlan Filter table data buffer.

struct __netc_tb_vf_config
: *#include <fsl_netc.h>* Vlan Filter table entry config.

struct __netc_tb_fdb_keye
: *#include <fsl_netc.h>*

Public Members

uint8_t macAddr[6]
: Destination MAC address of the frame for MAC forwarding lookups and the source MAC address of the frame for MAC learning lookups

uint32_t fid
: Filtering ID, is obtained from an ingress lookup into the VLAN Filter table

struct __netc_tb_fdb_cfge
: *#include <fsl_netc.h>* FDB table configuration element.

Public Members

uint32_t portBitmap
: Forwarding destination Port Bitmap and ET applicability port bitmap when oETEID = 10b

netc_tb_fdb_oeteid_mode_t oETEID
: Override ET_EID option

uint32_t ePort
: Egress Ports, active when oETEid = 01b or ctd = 01b

uint32_t iMirE
: Ingress Mirroring Enable

netc_tb_fdb_ctd_mode_t ctd
: Cut-Through Disable

uint32_t dynamic
: Static or Dynamic Entry, 0b = Static entry, 1b = Dynamic entry

uint32_t timeCapE
: Timestamp Capture Enable when set

uint32_t etEID

Base egress treatment table entry id for primary Egress Treatment group, is valid if the oETEID field is set to value other than kNETC_FDBNoEPP. 0xFFFFFFFF is NULL.

struct __netc_tb_fdb_acte

#include <fsl_netc.h> FDB table Activity element.

Public Members

uint8_t actCnt

Activity Counter

uint8_t actFlag

Activity Flag

struct __netc_tb_fdb_search_criteria

#include <fsl_netc.h> FDB table search criteria format.

Public Members

uint32_t resumeEntryId

Resume Entry ID, pass the NULL Entry ID when starting a search

netc_tb_fdb_keye_t keye

Key Element data which used to match against the table entries

netc_tb_fdb_cfge_t cfge

Configuration Element data which used to match against the table entries

struct __netc_tb_fdb_req_data

#include <fsl_netc.h> FDB table request data buffer.

Public Members

netc_tb_common_header_t commonHeader

Define update actions (use netc_tb_fdb_update_action_t) and query actions

netc_tb_fdb_cfge_t cfge

Present only for commands which perform an update or add

struct __netc_tb_fdb_rsp_data

#include <fsl_netc.h> FDB table request response data buffer.

Public Members

uint32_t status

RESUME_ENTRY_ID, valid only in responses for commands which use the Search Access Method

uint32_t entryID

Present only for query command

netc_tb_fdb_keye_t keye

Present only for query command

netc_tb_fdb_cfge_t cfge

Present only for query command

netc_tb_fdb_acte_t acte
Present only for query command

struct __netc_tb_fdb_data
#include <fsl_netc.h> FDB table data buffer.

struct __netc_tb_fdb_config
#include <fsl_netc.h> FDB table entry config.

struct __netc_tb_l2mcf_keye
#include <fsl_netc.h> L2 IPV4 Multicast Filter table key element.

Public Members

netc_tb_l2mcf_key_type_t keyType
Key Type

uint32_t fid
Filtering ID

uint32_t ipv4DestAddr
IPv4 Destination Address

uint32_t ipv4SrcAddr
IPv4 Source Address

struct __etc_tb_l2mcf_search_criteria
#include <fsl_netc.h> L2 IPV4 Multicast Filter table search criteria format.

Public Members

uint32_t resumeEntryId
Resume Entry ID, pass the NULL Entry ID when starting a search

netc_tb_l2mcf_keye_t keye
Key Element data which used to match against the table entries

netc_tb_l2mcf_cfge_t cfge
Configuration Element data which used to match against the table entries

struct __netc_tb_l2mcf_req_data
#include <fsl_netc.h> L2 IPV4 Multicast Filter table request data buffer.

struct __netc_tb_l2mcf_rsp_data
#include <fsl_netc.h> L2 IPV4 Multicast Filter table request response data buffer.

struct __netc_tb_l2mcf_data
#include <fsl_netc.h> L2 IPV4 Multicast Filter table data buffer.

struct __netc_tb_l2mcf_config
#include <fsl_netc.h> L2 IPV4 Multicast Filter table entry config.

struct __netc_tb_iseqg_cfge
#include <fsl_netc.h> Ingress Sequence Generation table config element.

Public Members

netc_tb_iseqg_sqtag_t sqTag
: Sequence Tag Type.

uint8_t ___pad0___
: Reserved.

struct __netc_tb_iseqg_sgse
: *#include <fsl_netc.h>* Ingress Sequence Generation table Sequence generation state element.

Public Members

uint16_t sqgNum
: Sequence Generation Number

struct __netc_tb_iseqg_req_data
: *#include <fsl_netc.h>* Ingress Sequence Generation table request data buffer.

struct __netc_tb_iseqg_rsp_data
: *#include <fsl_netc.h>* Ingress Sequence Generation table request response data buffer.

struct __netc_tb_iseqg_data
: *#include <fsl_netc.h>* Ingress Sequence Generation table data buffer.

struct __netc_tb_iseqg_config
: *#include <fsl_netc.h>* Ingress Sequence Generation table entry config.

struct __netc_tb_eseqr_cfge
: *#include <fsl_netc.h>* Egress Sequence Recovery table config element.

Public Members

netc_tb_eseqr_sqtag_t sqTag
: Sequence Tag, specify the expected sequence tag type in the frame

uint32_t sqrTnsq
: Sequence Recovery Take No Sequence

uint32_t sqrAlg
: Sequence Recovery Algorithm, 0b = Vector algorithm, 1b = Match algorithm

uint32_t sqrType
: Sequence Recovery Function type, 0b = Sequence recovery function, 1b = Individual recovery function

uint32_t sqrHl
: Sequence Recovery History Length, valid if sqrAlg = 0b

uint32_t sqrFwl
: Sequence Recovery Future Window Length, valid if sqrAlg = 0b

uint32_t sqrTp
: Sequence Timeout Period, the unit is 1.048576 milliseconds

struct __netc_tb_eseqr_stse
: *#include <fsl_netc.h>* Egress Sequence Recovery table statistic element.

Public Members

uint32_t inOrderPackets[2]
In Order Packets

uint32_t outOfOrderPackets[2]
Out of Order Packets

uint32_t roguePackets[2]
Rogue Packets

uint32_t duplicatePackets[2]
Duplicate Packets

uint32_t lostPackets[2]
Lost Packets

uint32_t taglessPackets[2]
Tag-Less Packets

uint32_t esqrResetCounts
Sequence Recovery Resets

struct __netc_tb_eseqr_srse
#include <fsl_netc.h> Egress Sequence Recovery table sequence recovery state element.

Public Members

uint32_t sqrNum
Sequence Recovery Number

uint32_t takeAny
Take Any

uint32_t lce
Lost Count Enable

uint32_t sqrTs
Sequence Recovery Timestamp

uint32_t sqrHistory[4]
Recovery History bit vector, each bit corresponding to sequence numbers, bit 1 means a packet with that sequence number has been previously received

struct __netc_tb_eseqr_req_data
#include <fsl_netc.h> Egress Sequence Recovery table request data buffer.

struct __netc_tb_eseqr_rsp_data
#include <fsl_netc.h> Egress Sequence Recovery table request response data buffer.

struct __netc_tb_eseqr_data
#include <fsl_netc.h> Egress Sequence Recovery table data buffer.

struct __netc_tb_eseqr_config
#include <fsl_netc.h> Egress Sequence Recovery table entry config.

struct __netc_tgs_gate_entry
#include <fsl_netc.h> Defines the Time Gate Scheduling gate control entry structure.

Public Members

uint32_t interval
: Entry Time Interval

struct __netc_tb_tgs_cfge
: *#include <fsl_netc.h>* Time Gate Scheduling table config element.

Public Members

uint64_t adminBaseTime
: Administrative Base Time

uint32_t adminCycleTime
: Administrative Cycle Time

uint32_t adminCycleTimeExt
: Administrative Cycle Time Extension

uint32_t adminControlListLength
: Administrative Control List Length

netc_tgs_gate_entry_t adminGcl[]
: Administrative Gate control list

struct __netc_tb_tgs_olse
: *#include <fsl_netc.h>* Time Gate Scheduling table statistic element.

Public Members

uint64_t configChangeTime
: The time at which this operational gate control list became active

uint64_t configChangeError
: Count of error configuration changes

uint64_t operBaseTime
: Operational Base Time

uint32_t operCycleTime
: Operational Cycle Time

uint32_t operCycleTimeExt
: Operational Cycle Time Extension

uint32_t operControlListLength
: Operational Control List Length

netc_tgs_gate_entry_t operGcl[]
: Operational Gate control list

struct __netc_tb_tgs_req_data
: *#include <fsl_netc.h>* Time Gate Scheduling table request data buffer.

Public Members

netc_tb_tgs_cfge_t cfge
: Present only for commands which perform a update

struct __netc_tb_tgs_rsp_data
: *#include <fsl_netc.h>* Time Gate Scheduling table request response data buffer.

Public Members

uint32_t entryID
: Present only for commands which perform a query

netc_tb_tgs_cfge_t cfge
: Present only for commands which perform a query

netc_tb_tgs_olse_t olse
: Present only for commands which perform a query

struct __netc_tb_tgs_data
: *#include <fsl_netc.h>* Time Gate Scheduling table data buffer, set with max size.

struct __netc_tb_tgs_gcl
: *#include <fsl_netc.h>* Time Gate Scheduling table entry gate control list structure.

Public Members

uint64_t baseTime
: Base Time

uint32_t cycleTime
: Cycle Time

uint32_t extTime
: Cycle Time Extension

uint32_t numEntries
: Control List entry numbers

netc_tgs_gate_entry_t *gcList
: Pointer to time gate control list array

struct __netc_tb_et_cfge
: *#include <fsl_netc.h>* Egress Treatment table config element.

Public Members

netc_tb_et_efm_mode_t efmMode
: Egress Frame Modification mode

netc_tb_et_esq_act_t esqa
: Egress Sequence Actions

netc_tb_et_ec_act_t eca
: Egress Counter Action

uint8_t ___pad1___
: Reserve for data align

uint8_t efmLenChange
: Egress Frame Modification Length Change, specified in units of bytes using a 2's complement notation

uint16_t efmDataLen
: Egress Frame Modification Data Length

uint32_t efmEID
: Egress Frame Modification Entry Id

uint32_t ecEID
Egress Count Table Entry ID

uint32_t esqaTgtEID
Egress Sequence Actions Target Entry ID, active when esqa = 10b

struct __netc_tb_et_req_data
#include <fsl_netc.h> Egress Treatment table request data buffer.

struct __netc_tb_et_rsp_data
#include <fsl_netc.h> Egress Treatment table request response data buffer.

struct __netc_tb_et_data
#include <fsl_netc.h> Egress Treatment table data buffer.

struct __netc_tb_et_config
#include <fsl_netc.h> Egress Treatment table entry config.

struct __netc_tb_etmcq_cfge
#include <fsl_netc.h> ETM Class Queue table config element.

Public Members

netc_hw_etm_class_queue_idx_t cq2cgMap
Class Queue to Congestion Group Mapping

struct __netc_tb_etmcq_stse
#include <fsl_netc.h> ETM Class Queue table statistic element.

Public Members

uint32_t rejByteCnt[2]
Reject Byte Count

uint32_t rejFrameCnt[2]
Reject Frame Count

uint32_t deqByteCnt[2]
Dequeue Byte Count

uint32_t deqFrameCnt[2]
Dequeue Frame Count

uint32_t dropByteCnt[2]
Dropped Frames, Memory Lost

uint32_t dropFrameCnt[2]
Dropped Frames, Memory Recovered

uint32_t frmCnt
Frame Count

struct __netc_tb_etmcq_req_data
#include <fsl_netc.h> ETM Class Queue table request data buffer.

struct __netc_tb_etmcq_rsp_data
#include <fsl_netc.h> ETM Class Queue table request response data buffer.

struct __netc_tb_etmcq_data
#include <fsl_netc.h> ETM Class Queue table data buffer.

struct _netc_tb_etmcq_config

#include <fsl_netc.h> ETM Class Queue table entry config.

Public Members

uint32_t entryID

Need use NETC_TB_ETM_CQ_ENTRY_ID macro to create entry ID

struct _netc_tb_etmcs_cfge

#include <fsl_netc.h> ETM Class Scheduler table config element.

Public Members

netc_tb_etmcs_ca_assg_t cqAssg

Class Queue Assignment, input 0 to 7 are weighted fair whereby input 8 to 15 are strict priority

uint32_t oal

Overead accounting length

struct *_netc_tb_etmcs_cfge* wbfsWeight[8]

Weight for scheduler input 0 ~ 7, effective weight is: (2^x)/(1-(y/64))

struct _netc_tb_etmcs_req_data

#include <fsl_netc.h> ETM Class Scheduler table request data buffer.

Public Members

netc_tb_etmcs_entry_id_t entryID

One class scheduler entry per port

struct _netc_tb_etmcs_rsp_data

#include <fsl_netc.h> ETM Class Scheduler table request response data buffer.

struct _netc_tb_etmcs_data

#include <fsl_netc.h> ETM Class Scheduler table data buffer.

struct _netc_tb_etmcs_config

#include <fsl_netc.h> ETM Class Scheduler table entry config.

Public Members

netc_tb_etmcs_entry_id_t entryID

One class scheduler entry per port

struct _netc_tb_etmcg_cfge

#include <fsl_netc.h> ETM Congestion Group table config element.

Public Members

uint16_t tdDr0En

Tail drop enable for DR0 Frame

uint16_t tdDr1En

Tail drop enable for DR1 Frame

uint16_t tdDr2En
: Tail drop enable for DR2 Frame

uint16_t tdDr3En
: Tail drop enable for DR3 Frame

uint16_t oal
: Overhead accounting length, 2's complement value (range -2048 to +2047)

struct *_netc_tb_etmcg_cfge* tdDRThresh[4]
: Tail Drop Threshold (TA * 2^Tn) for DR0 ~ DR3 Frames, valid if tdDrnEn = 1b

struct _netc_tb_etmcg_stse
: *#include <fsl_netc.h>* ETM Congestion Group table statistic element.

Public Members

uint32_t byteCount[2]
: Number of bytes currently in use in all class queues that are members of this group.

struct _netc_tb_etmcg_req_data
: *#include <fsl_netc.h>* ETM Congestion Group table request data buffer.

struct _netc_tb_etmcg_rsp_data
: *#include <fsl_netc.h>* ETM Congestion Group table request response data buffer.

struct _netc_tb_etmcg_data
: *#include <fsl_netc.h>* ETM Congestion Group table data buffer.

struct _netc_tb_etmcg_config
: *#include <fsl_netc.h>* ETM Congestion Group table entry config.

Public Members

uint32_t entryID
: Need use NETC_TB_ETM_CG_ENTRY_ID macro to create entry ID

struct _netc_tb_ec_stse
: *#include <fsl_netc.h>* Egress Count table statistic element.

Public Members

uint32_t enqFrmCnt[2]
: Enqueued Frame Count

uint32_t rejFrmCnt[2]
: Rejected Frame Count

struct _netc_tb_ec_req_data
: *#include <fsl_netc.h>* Egress Count table request data buffer.

struct _netc_tb_ec_rsp_data
: *#include <fsl_netc.h>* Egress Count table request response data buffer.

struct _netc_tb_ec_data
: *#include <fsl_netc.h>* Egress Count table data buffer.

struct _netc_tb_bp_cfge
: *#include <fsl_netc.h>* Buffer Pool table config element.

Public Members

bool sbpEn

Shared Buffer Pool Enable, set true measn a shared buffer pool is associated with this buffer pool

netc_tb_bp_fc_cfg_t gcCfg

Flow Control (FC) Configuration

uint8_t pfcVector

Priority Flow Control (PFC) Vector, not support in NETC 3.0 and 3.1 version

uint16_t maxThresh

Maximum Threshold, value 0 means disable maximum threshold checking, use NETC_TB_BP_THRESH macro to set this value

uint16_t fcOnThresh

Flow Control On Threshold, If the buffer pool usage crosses this threshold, and if fcOnThresh is greater than fcOffThresh, the flow control state of the buffer pool is set to 1, use NETC_TB_BP_THRESH macro to set this value.

uint16_t fcOffThresh

Flow Control Off Threshold, If buffer pool usage drops to this threshold or below, the flow control state of the buffer pool is set to 0, , use NETC_TB_BP_THRESH macro to set this value

uint32_t sbpThresh

Shared Buffer Pool Threshold, use NETC_TB_BP_THRESH macro to set this value

uint32_t sbpEid

Shared Buffer Pool Entry ID, valid if sbpEn is true

uint32_t fcPorts

Flow Control Port bitmap, indicates which ports are to be flow controlled for this buffer pool

struct __netc_tb_bp_bpse

#include <fsl_netc.h> Buffer Pool table State Element Data.

Public Members

uint32_t amountUsed

Amount Used, number of internal memory words (average of 20 bytes each) currently in use in this buffer pool.

uint32_t amountUsedHWM

Amount Used High Watermark, value sticks at the highest AMOUNT_USED seen since the last watermark reset

uint32_t fcState

Flow Control (FC) State, ON (1) or OFF (0)

uint32_t bpd

Buffer Pool Disabled, 1 means the buffer pool has been disabled due to an uncorrectable ECC error

struct __netc_tb_bp_req_data

#include <fsl_netc.h> Buffer Pool table request data buffer.

struct __netc_tb_bp_rsp_data

#include <fsl_netc.h> Buffer Pool table request response data buffer.

struct _netc_tb_bp_data
#include <fsl_netc.h> Buffer Pool table data buffer.

struct _netc_tb_bp_config
#include <fsl_netc.h> Buffer Pool table entry config.

Public Members

uint32_t entryID
Buffer pool ID, range in 0 ~ (SWT_GetBPTableEntryNum() - 1)

netc_tb_bp_cfge_t cfge
Buffer Pool table config element

struct _netc_tb_sbp_cfge
#include <fsl_netc.h> Shared Buffer Pool table config element.

Public Members

uint32_t maxThresh
Maximum Threshold, If shared buffer pool usage is greater than or equal to this threshold, use NETC_TB_BP_THRESH macro to set this value

uint16_t fcOnThresh
Flow Control On Threshold, If the shared buffer pool usage crosses this threshold, and if fcOnThresh is greater than fcOffThresh, the flow control state of the buffer pool is set to 1, use NETC_TB_BP_THRESH macro to set this value.

uint16_t fcOffThresh
Flow Control Off Threshold, If shared buffer pool usage drops to this threshold or below, the flow control state of the buffer pool is set to 0, use NETC_TB_BP_THRESH macro to set this value

struct _netc_tb_sbp_sbpse
#include <fsl_netc.h> Shared Buffer Pool table State Element Data.

Public Members

uint32_t amountUsed
Amount Used, number of internal memory words (average of 20 bytes each) currently in use in this buffer pool.

uint32_t amountUsedHWM
Amount Used High Watermark, value sticks at the highest AMOUNT_USED seen since the last watermark reset

uint32_t fcState
Flow Control (FC) State, ON (1) or OFF (0)

struct _netc_tb_sbp_req_data
#include <fsl_netc.h> Shared Buffer Pool table request data buffer.

struct _netc_tb_sbp_rsp_data
#include <fsl_netc.h> Shared Buffer Pool table request response data buffer.

struct _netc_tb_sbp_data
#include <fsl_netc.h> Shared Buffer Pool table data buffer.

struct _netc_tb_sbp_config
#include <fsl_netc.h> Shared Buffer Pool table entry config.

Public Members

uint32_t entryID
: Shared Buffer pool ID, range in 0 ~ (SWT_GetSBPTableEntryNum() - 1)

netc_tb_sbp_cfge_t cfge
: Shared Buffer Pool table config element

union __netc_tb_data_buffer
: *#include <fsl_netc.h>* Table common data buffer.

Public Members

netc_tb_tgs_data_t tgs
: Time Gate Scheduling table data buffer

netc_tb_rp_data_t rp
: Rate Policer table data buffer

netc_tb_ipf_data_t ipf
: Ingress Port filter table data buffer

netc_tb_fdb_data_t fdb
: FDB table data buffer

netc_tb_l2mcf_data_t l2mcf
: L2 IPV4 Multicast Filter table data buffer

netc_tb_vf_data_t vf
: VLAN Filter table data buffer

netc_tb_isi_data_t isi
: Ingress Stream Identification table data buffer

netc_tb_is_data_t is
: Ingress Stream table data buffer

netc_tb_isf_data_t isf
: Ingress Stream Filter table data buffer

netc_tb_isc_data_t isc
: Ingress Stream Count table data buffer

netc_tb_sgi_data_t sgi
: Stream Gate Instance table data buffer

netc_tb_sgcl_data_t sgcl
: Stream Gate Control List table data buffer

netc_tb_fm_data_t fm
: Frame Modification table data buffer

netc_tb_fmd_data_t fmd
: Frame Modification Data table data buffer

netc_tb_et_data_t et
: Egress Treatment table data buffer

netc_tb_ec_data_t ec
: Egress Count table data buffer

netc_tb_etmcq_data_t cq
ETM Class Queue table data buffer

netc_tb_etmcs_data_t cs
ETM Class Scheduler table data buffer

netc_tb_etmcg_data_t cg
ETM Class Group table data buffer

netc_tb_iseqg_data_t iseqg
Ingress Sequence Generation table data buffer

netc_tb_eseqr_data_t eseqr
Egress Sequence Recovery table data buffer

netc_tb_bp_data_t bp
Buffer Pool table data buffer

netc_tb_sbp_data_t sbp
Shared Buffer Pool table data buffer

struct _netc_cbdr_hw
#include <fsl_netc_hw.h> Register group for SI/Switch command bd ring.

Public Members

___IO uint32_t CBDRMR
Command BDR mode register.

___I uint32_t CBDRSR
Command BDR status register.

___IO uint32_t CBDRBAR0
Command BDR base address register 0

___IO uint32_t CBDRBAR1
Command BDR base address register 1

___IO uint32_t CBDRPIR
Command BDR producer index register

___IO uint32_t CBDRCIR
Command BDR consumer index register

___IO uint32_t CBDRLENR
Command BDR length register

struct _netc_cbdr_handle
#include <fsl_netc_hw.h> Handle for common part of EP/Switch NTMP.

Public Members

netc_cbdr_hw_t *base
Point to hardware command bd ring register group.

netc_cmd_bdr_t *cmdr
Point to command BD ring handle.

netc_tb_data_buffer_t *buffer
Point to table common data buffer.

struct req

Public Members

uint64_t addr
: The request and response data buffers address

struct ___unnamed91___

Public Members

uint32_t resLength
: The length of the Response Data Buffer

uint32_t reqLength
: The length of the Request Data Buffer

struct ___unnamed93___

Public Members

netc_tb_cmd_t cmd
: Access table entry command, see netc_tb_cmd_t .

netc_tb_access_mode_t accessType
: Access table entry method, see netc_tb_access_mode_t.

uint32_t ___pad1___
: RSS Hash high field value.

uint32_t version
: Protocol Version.

uint32_t enCompInt
: Command Completion Interrupt.

uint32_t resReady
: Response Ready.

struct ___unnamed95___

Public Members

uint32_t npf
: NTMP Protocol Format.

struct resp

struct ___unnamed97___

Public Members

uint32_t numMatched
: Number of Entries Matched.

uint32_t error
: Error status.

uint32_t resReady
: Response Ready.

struct generic

Public Members

uint64_t addr
: Data.

uint32_t en
: Enable entry.

uint32_t siBitMap
: Station interfaces 15-0 for which this filter applies.

uint32_t index
: The index refers to an entry location within a table.

uint32_t length
: NA

uint32_t cmd
: Command.

uint32_t ___pad2___
: < Class of command.

uint32_t status
: Status.

uint32_t ci
: Completion interrupt.

uint32_t sf
: Short format.

struct frameAttr

Public Members

uint16_t swtPortMas
: Switch port masquerading, applicable only if the incoming port is designated as a switch management port

uint16_t ethernet
: Ethernet type Present

uint16_t outerVlan
: Outer VLAN Present

uint16_t innerVlan
: Inner VLAN Present

netc_tb_ipf_seq_tag_t seqTag
: Sequence Tag Code

uint16_t ipHeader
: IP Header Present

uint16_t ipVersion
: 0b = IPv4, 1b = IPv6

uint16_t ipExt
: IPv4 option / IPv6 extension present

netc_tb_ipf_l4_header_t l4Header
L4 Header code

uint16_t wakeOnLan
Wake-on-LAN Magic Packet Present

struct payload

Public Members

uint8_t data
Payload Byte n

uint8_t mask
Payload Byte n Mask

union ___unnamed101___

Public Members

netc_tb_ipf_keye_t keye

uint32_t entryID

uint32_t sCriteria

union ___unnamed103___

Public Members

netc_tb_ipf_req_data_t request

netc_tb_ipf_rsp_data_t response

union ___unnamed105___

Public Members

uint32_t entryID

uint32_t sCriteria

netc_tb_isi_keye_t keye

union ___unnamed107___

Public Members

netc_tb_isi_req_data_t request

netc_tb_isi_rsp_data_t response

union ___unnamed109___

Public Members

netc_tb_is_req_data_t request

netc_tb_is_rsp_data_t response

union ___unnamed111___

Public Members

uint32_t entryID

uint32_t sCriteria

netc_tb_isf_keye_t keye

union ___unnamed113___

Public Members

netc_tb_isf_req_data_t request

netc_tb_isf_rsp_data_t response

union ___unnamed115___

Public Members

netc_tb_rp_cfge_t cfge

Present only for commands which perform an update or add

struct __netc_tb_rp_req_data

struct ___unnamed117___

Public Members

netc_tb_rp_fee_t fee

Present only for commands which perform an update or add

union ___unnamed119___

Public Members

netc_tb_rp_cfge_t cfge

Present only for commands which perform a query

struct __netc_tb_rp_rsp_data

struct __netc_tb_rp_rsp_data

struct ___unnamed121___

Public Members

netc_tb_rp_fee_t fee

Present only for commands which perform a query

struct ___unnamed123___

Public Members

netc_tb_rp_pse_t pse

Present only for commands which perform a query

union ___unnamed125___

Public Members

netc_tb_rp_req_data_t request

netc_tb_rp_rsp_data_t response

union ___unnamed127___

Public Members

netc_tb_isc_req_data_t request

netc_tb_isc_rsp_data_t response

union ___unnamed129___

Public Members

netc_tb_sgi_sgise_t sgise

struct __netc_tb_sgi_rsp_data

struct ___unnamed131___

union ___unnamed133___

Public Members

netc_tb_sgi_req_data_t request

netc_tb_sgi_rsp_data_t response

union ___unnamed135___

Public Members

netc_tb_sgcl_req_data_t request

netc_tb_sgcl_rsp_data_t response

struct __netc_tb_sgcl_data

struct ___unnamed137___

union ___unnamed139___

Public Members

netc_tb_fm_req_data_t request

netc_tb_fm_rsp_data_t response

union ___unnamed141___

Public Members

netc_tb_fmd_req_data_t request

netc_tb_fmd_rsp_data_t response

union ___unnamed143___

Public Members

uint32_t entryID

netc_tb_vf_search_criteria_t sCriteria
Active when access method is kNETC_Search

netc_tb_vf_keye_t keye

union ___unnamed145___

Public Members

netc_tb_vf_req_data_t request

netc_tb_vf_rsp_data_t response

struct ___unnamed147___

Public Members

netc_tb_fdb_acte_t acte
Activity Element data which used to match against the table entries

netc_tb_fdb_sc_keye_mc_t keyeMc
Key Element data match criteria

netc_tb_fdb_sc_cfge_mc_t cfgeMc
Configuration Element data match criteria

netc_tb_fdb_sc_acte_mc_t acteMc
Activity Element data match criteria

union ___unnamed149___

Public Members

uint32_t entryID
Active when access method is kNETC_EntryIDMatch

netc_tb_fdb_keye_t keye
Active when access method is kNETC_ExactKeyMatch

netc_tb_fdb_search_criteria_t sCriteria
Active when access method is kNETC_Search

union ___unnamed151___

Public Members

netc_tb_fdb_req_data_t request

netc_tb_fdb_rsp_data_t response

struct ___unnamed153___

Public Members

netc_tb_l2mcf_acte_t acte
Activity Element data which used to match against the table entries

etc_tb_l2mcf_sc_keye_mc_t keyeMc
Key Element data match criteria

etc_tb_l2mcf_sc_cfge_mc_t cfgeMc
Configuration Element data match criteria

etc_tb_l2mcf_sc_acte_mc_t acteMc
Activity Element data match criteria

union ___unnamed155___

Public Members

uint32_t entryID

netc_tb_l2mcf_search_criteria_t sCriteria

netc_tb_l2mcf_keye_t keye

union ___unnamed157___

Public Members

netc_tb_l2mcf_req_data_t request

netc_tb_l2mcf_rsp_data_t response

union ___unnamed159___

Public Members

netc_tb_iseqg_req_data_t request

netc_tb_iseqg_rsp_data_t response

union ___unnamed161___

Public Members

netc_tb_eseqr_req_data_t request

netc_tb_eseqr_rsp_data_t response

union ___unnamed163___

Public Members

struct __netc_tgs_gate_entry

uint32_t gate

Entry Gate Mask

struct ___unnamed165___

Public Members

uint32_t tcGateState

Traffic Class Gate States for Gate Entry, 8 bits for 8 Traffic Class , 0b means Gate closed, 1b means Gate open

netc_tb_tgs_gate_type_t operType

Gate operation type (IEEE 802.1Q-2018) field for gate control list entry i

union ___unnamed167___

Public Members

netc_tb_tgs_req_data_t request

netc_tb_tgs_rsp_data_t response

struct __netc_tb_tgs_data

struct ___unnamed169___

union ___unnamed171___

Public Members

netc_tb_et_req_data_t request

netc_tb_et_rsp_data_t response

union ___unnamed173___

Public Members

netc_tb_etmcq_req_data_t request

netc_tb_etmcq_rsp_data_t response

struct wbfsWeight

Public Members

uint8_t xCode
: Weight code x value

uint8_t yCode
: Weight code y value

union ___unnamed176___

Public Members

netc_tb_etmcs_req_data_t request

netc_tb_etmcs_rsp_data_t response

struct tdDRThresh

Public Members

uint16_t tn
: TA

uint16_t ta
: Tn

union ___unnamed179___

Public Members

netc_tb_etmcg_req_data_t request

netc_tb_etmcg_rsp_data_t response

union ___unnamed181___

Public Members

netc_tb_ec_req_data_t request

netc_tb_ec_rsp_data_t response

union ___unnamed183___

Public Members

netc_tb_bp_req_data_t request

netc_tb_bp_rsp_data_t response

union ___unnamed185___

Public Members

netc_tb_sbp_req_data_t request

netc_tb_sbp_rsp_data_t response

2.81 NETC MDIO Driver

2.82 MDIO initialization module

enum _netc_mdio_type

Enumeration for the MAC port MDIO type.

Values:

enumerator kNETC_EMdio

Bound handle to EMDIO access, submodule of NETC.

enumerator kNETC_InternalMdio

Bound handle to MAC port internal MDIO access, submodule of EP/Switch.

enumerator kNETC_ExternalMdio

Bound handle to MAC port external MDIO access, submodule of EP/Switch.

typedef enum *_netc_mdio_type* netc_mdio_type_t

Enumeration for the MAC port MDIO type.

typedef struct *_netc_mdio* netc_mdio_t

Structure to choose MDIO entity(Internal/external MDIO for specified EP/Switch port)

typedef struct *_netc_mdio_handle* netc_mdio_handle_t

MDIO handle.

typedef struct *_netc_mdio_config* netc_mdio_config_t

MDIO configuration structure.

status_t NETC_MDIOInit(*netc_mdio_handle_t* *handle, *netc_mdio_config_t* *config)

Initialize the MDIO.

Note: The EMDIO can be used independently. The port internal/external MDIO is a part of EP/Switch, should be initialized and used after EP/Switch is enabled.

Parameters

- handle – MDIO handle.
- config – MDIO configuration.

Returns

status_t

struct _netc_mdio

#include <fsl_netc_mdio.h> Structure to choose MDIO entity(Internal/external MDIO for specified EP/Switch port)

Public Members

netc_mdio_type_t type

Internal or external MAC port MDIO.

netc_hw_eth_port_idx_t port

MDIO port index, only meaningful when port MDIO type is used.

struct _netc_mdio_handle

#include <fsl_netc_mdio.h> MDIO handle.

Public Members

netc_mdio_t mdio
MDIO identificator.

struct __netc_mdio_config
#include <fsl_netc_mdio.h> MDIO configuration structure.

Public Members

netc_mdio_t mdio
MDIO identificator.

uint32_t srcClockHz
MDIO reference clock for MDC frequency calculation.

bool isNegativeDriven
MDIO driven at positive(false)/negative(true) of MDC edge.

bool isPreambleDisable
Enable/Disable generation of MDIO preamble.

2.83 MDIO PHY status module

typedef struct *_netc_mdio_phy_status* netc_mdio_phy_status_t
PHY auto status check configuration structure.

status_t NETC_MDIOSetPhyStatusCheck(*netc_mdio_handle_t* *handle, *netc_mdio_phy_status_t* *config)
Setup the mechanism to check PHY status automatically This is a hardware mechanism to read specified PHY register in a configured time interval instead of polling the PHY in software.

Parameters

- handle – MDIO handle.
- config – The configuration of the PHY status automatical check.

Returns
status_t

void NETC_MDIOPhyStatusGetFlags(*netc_mdio_handle_t* *handle, uint16_t *low2HighMask, uint16_t *high2LowMask)
Get the PHY register bit status transition interrupt flag(s).

Parameters

- handle – MDIO handle.
- low2HighMask – The interrupt flag of a 0->1 transition on a corresponding bit of PHY register.
- high2LowMask – The interrupt flag of a 1->0 transition on a corresponding bit of PHY register.

void NETC_MDIOPhyStatusClearFlags(*netc_mdio_handle_t* *handle, uint16_t low2HighMask, uint16_t high2LowMask)
Clear the PHY register bit status transition interrupt flag(s).

Parameters

- handle – MDIO handle.
- low2HighMask – Clear the interrupt flag of a 0->1 transition on a corresponding bit of PHY register.
- high2LowMask – Clear the interrupt flag of a 1->0 transition on a corresponding bit of PHY register.

struct _netc_mdio_phy_status

#include <fsl_netc_mdio.h> PHY auto status check configuration structure.

Public Members

uint16_t interval

PHY status read interval in units of 1-2 ms. A value of 0 indicates disable.

bool isC45Used

PHY status read with Clause 22/45 MDIO access.

uint8_t phyOrPortAddr

MDIO PHY address(Clause 22) / port address(Clause 45).

uint8_t regiOrDevAddr

MDIO register address(Clause 22) / device address(Clause 45).

uint16_t c45RegiAddr

MDIO register address(Clause 45).

uint16_t enableIntrHigh2Low

Bit high-to-low event interrupt enable.

uint16_t enableIntrLow2High

Bit low-to-high event interrupt enable.

2.84 MDIO write/read module

status_t NETC_MDIOWrite(*netc_mdio_handle_t* *handle, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

IEEE802.3 Clause 22 MDIO write data.

Parameters

- handle – MDIO handle.
- phyAddr – The PHY address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns

status_t

status_t NETC_MDIORead(*netc_mdio_handle_t* *handle, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

IEEE802.3 Clause 22 MDIO read data.

Parameters

- handle – MDIO handle.
- phyAddr – The PHY address.

- regAddr – The PHY register address.
- pData – The received data from PHY.

Returns
status_t

status_t NETC_MDIOC45Write(*netc_mdio_handle_t* *handle, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)

IEEE802.3 Clause 45 MDIO write data.

Parameters

- handle – MDIO handle.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns
status_t

status_t NETC_MDIOC45Read(*netc_mdio_handle_t* *handle, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)

IEEE802.3 Clause 45 MDIO read data.

Parameters

- handle – MDIO handle.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The received data from PHY.

Returns
status_t

2.85 NETC SWITCH (SWT) Driver

2.86 Switch (SWT) data path

2.87 Switch (SWT) Generic Configuration

typedef struct *_swt_handle* swt_handle_t

typedef struct *_swt_transfer_config* swt_transfer_config_t

typedef *status_t* (*swt_reclaim_cb_t)(*swt_handle_t* *handle, *netc_tx_frame_info_t* *frameInfo, void *userData)

Callback for reclaimed tx frames.

typedef void *(*swt_rx_alloc_cb_t)(*swt_handle_t* *handle, uint32_t length, void *userData)

Defines the Switch Rx memory buffer alloc function pointer.

typedef void (*swt_rx_free_cb_t)(*swt_handle_t* *handle, void *address, void *userData)

Defines the Switch Rx memory buffer free function pointer.

typedef struct *_swt_config* swt_config_t

Configuration structure for Switch.

typedef struct *_swt_config_const* swt_config_const_t

Configuration constant structure for Switch in handle.

status_t SWT_Init(*swt_handle_t* *handle, const *swt_config_t* *config)

Init SWITCH.

Parameters

- handle –
- config – The user configuration

Returns

status_t

status_t SWT_Deinit(*swt_handle_t* *handle)

Deinit switch.

Parameters

- handle –

Returns

status_t

status_t SWT_GetDefaultConfig(*swt_config_t* *config)

Get deafult configuration for switch.

Parameters

- config –

Returns

status_t

status_t SWT_SetEthPortMII(*swt_handle_t* *handle, uint8_t ethPort, *netc_hw_mii_speed_t* speed, *netc_hw_mii_duplex_t* duplex)

Set MII speed and duplex for switch ethernet port.

Parameters

- handle –
- ethPort –
- speed –
- duplex –

Returns

status_t

static inline void SWT_SetPortSpeed(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, uint16_t pSpeed)

Set switch port speed.

Parameters

- handle –
- portIdx –
- pSpeed –

status_t SWT_PortStop(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx)

Stop Switch Port receive/transmit (disable both ETH and port)

Parameters

- handle –
- portIdx –

Returns

status_t

status_t SWT_SetPortMaxFrameSize(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, uint16_t size)

Set switch port maximum supported received frame size.

Parameters

- handle –
- portIdx –
- size –

Returns

status_t

struct _swt_config

#include <fsl_netc_switch.h> Configuration structure for Switch.

Public Members

netc_hw_switch_idx_t switchIdx

Switch index

struct *_swt_config* ports[5]

The common configuration required for all ports

uint16_t dynFDBLimit

Maximum number of entries which can be dynamically learned by FDB table, a value of 0 implies no limit.

netc_swt_psfp_config_t psfpCfg

The switch PSFP configuration, cover the ISI key construction profiles configuration

netc_ipf_config_t ipfCfg

The switch ingress port filter configuration

netc_vlan_classify_config_t vlanCfg

The switch ingress vlan classify configuration

netc_swt_qos_classify_config_t rxqosCfg

The switch ingress VLAN to QoS mapping configuration

netc_swt_qos_to_vlan_config_t txqosCfg

The switch egress QoS to VLAN mapping configuration

netc_swt_imr_config_t imrCfg

The switch Ingress mirror configuration

netc_swt_bridge_config_t bridgeCfg

The switch bridge configuration

uint8_t cmdRingUse
: Number of command BD rings to be used

netc_cmd_bdr_config_t cmdBdrCfg[2]
: The switch command BD rings configuration

netc_msix_entry_t *msixEntry
: MSIX table entry array.

uint8_t entryNum
: MSIX entry number.

uint8_t cmdBdrEntryIdx[2]
: MSIX entry index of command BD ring interrupt.

uint8_t timeCaptureEntryIdx
: MSIX entry index of time capture interrupt.

struct _swt_config_const
: *#include <fsl_netc_switch.h>* Configuration constant structure for Switch in handle.

Public Members

uint8_t cmdRingUse
: Number of command BD rings to be used

uint8_t entryNum
: MSIX entry number.

bool rxCacheMaintain
: Enable/Disable Rx buffer cache maintain in driver.

bool txCacheMaintain
: Enable/Disable Tx buffer cache maintain in driver.

bool enUseMgmtRxBdRing
: Enable/Disable use Switch management Rx BD ring.

bool rxZeroCopy
: Enable zero-copy receive mode.

swt_rx_alloc_cb_t rxBuffAlloc
: Callback function to alloc memory, must be provided for zero-copy Rx.

swt_rx_free_cb_t rxBuffFree
: Callback function to free memory, must be provided for zero-copy Rx.

swt_reclaim_cb_t reclaimCallback
: Callback for reclaimed Tx Switch management frames.

void *userData
: User data, return in callback.

struct _swt_handle
: *#include <fsl_netc_switch.h>* Handle for SWITCH.

Public Members

netc_switch_hw_t hw
: Hardware register map resource.

swt_config_const_t cfg
: Switch configuration constant.

netc_cmd_bdr_t cmdBdRing[2]
: Command BD ring handle for switch.

ep_handle_t *epHandle
: The EP handle to send/receive management frame.

netc_rx_bdr_t mgmtRxBdRing
: Management Receive BD ring for frames with Host Reason filed not zero.

netc_tx_bdr_t mgmtTxBdRing
: Management Transmit BD ring for Direct Switch Enqueue frames.

struct ports

Public Members

netc_port_ethmac_t ethMac
: Ethernet MAC configuration, vaild only on port 0 ~ 3

netc_port_common_t commonCfg
: Port common configuration

netc_swt_port_bridge_config_t bridgeCfg
: Port bridge configuration

netc_swt_port_fm_config_t fmCfg
: Port Frame Modification configuration

netc_port_tx_tc_config_t txTcCfg[8]
: Port Tx traffic class related configuration, vaild only on port 0 ~ 3

uint8_t ipvToTC[8]
: Port IPV to Tx traffic class queue index mapping, value means TC index, range in 0 ~ 7

uint8_t ipvToBP[8]
: Port IPV to buffer pool ID mapping, value means BP ID, range in 0 ~ (SWT_GetBPTableEntryNum() - 1)

netc_port_psfp_isi_config isiCfg
: Port Ingress stream identification

bool enMirror
: Enable/Disable Ingress Mirroring on this port

uint8_t lanID
: The HSR's LANID this port belong to

bool inCutThrough
: Enable/Disable Ingress Cut Through on this port, vaild only on port 0 ~ 3

bool outCutThrough
: Enable/Disable Egress Cut Through on this port, vaild only on port 0 ~ 3

bool enableTg
: Enable port time gate scheduling.

bool enTxRx
: Enable port transmit/receive path, if disabled, the tx/rx of the mac corresponding to the port will also be closed.

2.88 Switch (SWT) Interrupt Module

status_t SWT_MsixSetGlobalMask(*swt_handle_t* *handle, bool mask)

Set the global MSIX mask status.

This function masks/unmasks global MSIX message. Mask - All of the vectors are masked, regardless of their per-entry mask bit states. Unmask - Each entry's mask status determines whether the vector is masked or not.

Parameters

- handle – The EP handle
- mask – The mask state. True: Mask, False: Unmask.

Returns

status_t

status_t SWT_MsixSetEntryMask(*swt_handle_t* *handle, uint8_t entryIdx, bool mask)

Set the MSIX entry mask status for specified entry.

This function masks/unmasks MSIX message for specified entry.

Parameters

- handle – SWT handle.
- entryIdx – The entry index in the table.
- mask – The mask state. True: Mask, False: Unmask.

Returns

status_t

status_t SWT_MsixGetPendingStatus(*swt_handle_t* *handle, uint8_t pbaIdx, uint64_t *status)

Get the MSIX pending status in MSIX PBA table.

This function is to get the entry pending status from MSIX PBA table. If interrupt occurs but masked by vector control of entry, pending bit in PBA will be set.

Parameters

- handle – SWT handle.
- pbaIdx – The index of PBA array with 64-bit unit.
- status – Pending status bit mask, bit n for entry n.

Returns

status_t

2.89 Switch (SWT) management

status_t SWT_BridgeInit(*swt_handle_t* *handle, const *netc_swt_bridge_config_t* *config)

Config the Bridge.

Cover configuration below

- Vlan filter table default VFHTDRCR0-2

Not Cover

- Port specific Vlan config
- Port Storm Control BPSCR0/R1
- FDB entry

- Vlan Filter Table entry
- Multicast filter entry

Parameters

- handle –
- config –

Returns
status_t

status_t SWT_BridgeConfigPort(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, const *netc_swt_port_bridge_config_t* *config)

Config bridge port vlan Tx/Rx.

Cover configuration below

- Port specific Vlan config
- Port Storm Control BPSCR0/R1

Parameters

- handle –
- portIdx –
- config –

Returns
status_t

status_t SWT_BridgeConfigPortDefaultVid(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, uint16_t vid)

Config bridge port default VID.

Parameters

- handle –
- portIdx –
- vid –

Returns
status_t

static inline uint32_t SWT_BridgeGetVFTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of bridge vlan filter table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification, Ingress Stream Filter, FDB, L2 IPV4 Multicast Filter table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_BridgeAddVFTableEntry(*swt_handle_t* *handle, *netc_tb_vf_config_t* *config, uint32_t *entryID)

Add entry into bridge vlan filter table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeUpdateVFTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_vf_cfge_t* *cfg)

Update entry in bridge vlan filter table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeQueryVFTableEntry(*swt_handle_t* *handle, *netc_tb_vf_keye_t* *keye, *netc_tb_vf_rsp_data_t* *rsp)

Query an entry in bridge VF Table.

Parameters

- handle –
- keye –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeDelVFTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete entry in bridge vlan filter table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeSearchVFTableEntry(*swt_handle_t* *handle, *netc_tb_vf_search_criteria_t* *sCriteria, *netc_tb_vf_rsp_data_t* *rsp)

Search entry in bridge vlan filter table.

Note: Only support search and return one entry at a time.

Parameters

- handle –
- sCriteria –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_BridgeGetFDBTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of bridge FDB table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification, Ingress Stream Filter, VLAN Filter, L2 IPV4 Multicast Filter table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_BridgeAddFDBTableEntry(*swt_handle_t* *handle, *netc_tb_fdb_config_t* *config, uint32_t *entryID)

Add entry into bridge FDB table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeUpdateFDBTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_fdb_cfge_t* *cfg)

Update entry in bridge FDB table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeQueryFDBTableEntry(*swt_handle_t* *handle, *netc_tb_fdb_keye_t* *keye, *netc_tb_fdb_rsp_data_t* *rsp)

Query an entry in bridge FDB Table.

Parameters

- handle –
- keye –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeSearchFDBTableEntry(*swt_handle_t* *handle, *netc_tb_fdb_search_criteria_t* *sCriteria, *netc_tb_fdb_rsp_data_t* *rsp)

Search entry in bridge FDB table.

Note: Only support search and return one entry at a time.

Parameters

- handle –
- sCriteria –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeDelFDBTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete an entry in bridge FDB Table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeGetFDBActivityState(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_fdb_acte_t* *state)

Get the activity data for FDB table entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_BridgeGetL2MCFTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of bridge L2 IPV4 multicast filter table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification, Ingress Stream Filter, VLAN Filter, FDB table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_BridgeAddL2MCFTableEntry(*swt_handle_t* *handle, *netc_tb_l2mcf_config_t* *config, uint32_t *entryID)

Add entry into Bridge L2 IPV4 multicast filter table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeUpdateL2MCFTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_l2mcf_cfge_t* *cfg)

Update entry in bridge L2 IPV4 multicast filter table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeSearchL2MCFTableEntry(*swt_handle_t* *handle, *netc_tb_l2mcf_search_criteria_t* *sCriteria, *netc_tb_l2mcf_rsp_data_t* *rsp)

Search entry in bridge L2 IPV4 multicast filter table.

Note: Only support search and return one entry at a time.

Parameters

- handle –

- sCriteria –
- rsp –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeGetL2MCFActivityState(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_l2mcf_acte_t* *state)

Get the activity data for bridge L2 IPV4 multicast filter table entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_BridgeDelL2MCFTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete entry in bridge L2 IPV4 multicast filter table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline *status_t* SWT_FMConfigPort(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, const *netc_swt_port_fm_config_t* *config)

Config Port level Frame Modification.

Parameters

- handle –
- portIdx –
- config –

Returns
status_t

static inline uint32_t SWT_FMGetTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number of Frame Modification table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

static inline uint32_t SWT_FMGetTableMaxEntryNum(*swt_handle_t* *handle)

Get maximum entry number of Frame Modification table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_FMAddTableEntry(*swt_handle_t* *handle, *netc_tb_fm_config_t* *config)

Add entry into the Frame Modification table.

Add entry of the frame modification table. Frame modification can be encoded into the table id directly or provided as index of the table entry. refer to the frame modification entry definition.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FMUpdateTableEntry(*swt_handle_t* *handle, *netc_tb_fm_config_t* *config)

Update entry in the Frame Modification table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FMDelTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete table entry in Frame Modification Table.

The provided ID must be table index. Error return if the id is action encoded id.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FMQueryTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_fm_config_t* *config)

Query table entry in Frame Modification Table.

The provided ID must be table index. Error return if the id is action encoded id.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_FMDGetTableWordsNum(*swt_handle_t* *handle)

Get word number (word size is 24 bytes) of Frame Modification data table.

Note: This is a static bounded index table, when update or query table, should satisifed: numBytes + ENTRY_ID <= 24 * wordNumber .

Parameters

- handle –

Returns
uint32_t

status_t SWT_FMDUpdateTableEntry(*swt_handle_t* *handle, *netc_tb_fmd_update_config_t* *config, uint32_t length)

Update Frame Modification data table contents.

Parameters

- handle –
- config –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FMDQueryTableEntry(*swt_handle_t* *handle, *netc_tb_fmd_query_buffer_t* *query, uint32_t length)

Query Frame Modification data table contents.

Parameters

- handle –
- query –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_FRERGetISEQGTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number of ingress Sequence Generation table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

status_t SWT_FRERAddISEQGTableEntry(*swt_handle_t* *handle, *netc_tb_iseqg_config_t* *config)

Add entry into FRER sequence number generation table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FRERUpdateISEQGTableEntry(*swt_handle_t* *handle, *netc_tb_iseqg_config_t* *config)

Update entry in FRER sequence number generation table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FRERResetSEQGTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Reset sequence number generation in FRER sequence generation entry.

Parameters

- handle –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FRERDelISEQGTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete entry in FRER sequence number generation table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FRERGetISEQGState(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_iseqg_sgse_t* *state)

Get the Sequence generation number form sequence generation table for specified entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_FRERGetESEQRTableEntryNum(*swt_handle_t* *handle)

Get the number of entries assigned to FRER sequence number recovery table.

Note: This is a static bounded index table, so all entries in the table are available for SWT_FRERConfigESEQRTableEntry() API

Parameters

- handle –

Returns
uint32_t

status_t SWT_FRERConfigESEQRTableEntry(*swt_handle_t* *handle, *netc_tb_eseqr_config_t* *config)

Config FRER sequence number recovery table entry.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FRERQueryESEQRTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_eseqr_stse_t* *statistic, *netc_tb_eseqr_cfge_t* *config, *netc_tb_eseqr_srse_t* *state)

Query FRER sequence recovery table entry.

Parameters

- handle –
- entryID –
- statistic –
- config –

- state –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_FRERGetESEQRStatistic(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_eseqr_stse_t* *statistic, *netc_tb_eseqr_srse_t* *state)

Get FRER sequence recorvery table state and statistic.

Parameters

- handle –
- entryID –
- statistic –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_GetBPTableEntryNum(*swt_handle_t* *handle)

Get Buffer Pool table entry number.

Note: This is a static bounded index table, so all entries in the table are available for SWT_UpdateBPTableEntry() API

Parameters

- handle –

Returns
uint32_t

status_t SWT_UpdateBPTableEntry(*swt_handle_t* *handle, *netc_tb_bp_config_t* *config)

Update Buffer Pool table entry.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_GetBPEntryState(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_bp_bpse_t* *state)

Get Buffer Pool table entry State.

Parameters

- handle –
- entryID –

- state –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_GetSBPTableEntryNum(*swt_handle_t* *handle)

Get Shared Buffer Pool table entry number.

Note: This is a static bounded index table, so all entries in the table are available for SWT_UpdateSBPTableEntry() API

Parameters

- handle –

Returns
uint32_t

status_t SWT_UpdateSBPTableEntry(*swt_handle_t* *handle, *netc_tb_sbp_config_t* *config)

Update Shared Buffer Pool table entry.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_GetSBPEntryState(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_sbp_sbpse_t* *state)

Get Shared Buffer Pool table entry State.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

2.90 Switch (SWT) Table Management Module

status_t SWT_CmdBDRInit(*swt_handle_t* *handle, *netc_hw_swt_cbdr_idx_t* ring, const *netc_cmd_bdr_config_t* *config)

Init SWITCH command BD ring.

Parameters

- handle –

- ring – Command BD ring index
- config – The user configuration

Returns
status_t

status_t SWT_CmdBDRDeinit(*swt_handle_t* *handle, *netc_hw_swt_cbdr_idx_t* ring)

Deinit switch command BD ring.

Parameters

- handle –
- ring – Command BD ring index

Returns
status_t

2.91 Switch (SWT) Ingress data path configuration

status_t SWT_RxMirrorConfig(*swt_handle_t* *handle, const *netc_swt_imr_config_t* *config)

Configure switch ingress mirror.

Parameters

- handle –
- config –

Returns
status_t

static inline *status_t* SWT_RxParserConfig(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, const *netc_port_parser_config_t* *config)

Configure Parser in Receive Data Path.

Parameters

- handle –
- portIdx –
- config –

Returns
status_t

static inline *status_t* SWT_RxVlanCInit(*swt_handle_t* *handle, const *netc_vlan_classify_config_t* *config)

Configure the customer vlan type.

Parameters

- handle –
- config –

Returns
status_t

static inline *status_t* SWT_RxVlanCConfigPort(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_port_vlan_classify_config_t* *config)

Configure the Accepted Vlan for PORT.

Parameters

- handle –
- portIdx –
- config –

Returns
status_t

static inline *status_t* SWT_RxQosCInit(*swt_handle_t* *handle, const *netc_swt_qos_classify_config_t* *config)

Init the ingress QoS classification for Switch.

Parameters

- handle –
- config –

Returns
status_t

static inline *status_t* SWT_RxQosCConfigPort(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_port_qos_classify_config_t* *config)

Configure the QoS Classification for Switch Port.

Parameters

- handle –
- portIdx –
- config –

Returns
status_t

static inline *status_t* SWT_RxIPFInit(*swt_handle_t* *handle, const *netc_ipf_config_t* *config)

Enable/Disable Ingress Port Filtering.

Applied for both Switch and ENETC

Parameters

- handle –
- config –

Returns
status_t

static inline *status_t* SWT_RxIPFConfigPort(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, const *netc_port_ipf_config_t* *config)

Config the IPF for specified Port.

Parameters

- handle –
- portIdx –
- config –

Returns
status_t

static inline uint32_t SWT_RxIPFGetTableRemainWordNum(*swt_handle_t* *handle)

Get remaining available word number (words size is 6 bytes) of the ingress Port Filter Table.

Note: This is a ternary match table, and the entries can vary in size, from 2 to 14 words.

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxIPFAddTableEntry(*swt_handle_t* *handle, *netc_tb_ipf_config_t* *config, uint32_t *entryID)

Add an entry for the ingress Port Filter Table.

This function do an add & query with return hardware id which can be used as future query / delete / update.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxIPFUpdateTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_ipf_cfge_t* *cfg)

Update entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxIPFQueryTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_ipf_config_t* *config)

Query an entry for the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxIPFDelTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete an entry for the ingress Port Filter Table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxIPFResetMatchCounter(*swt_handle_t* *handle, uint32_t entryID)

Reset the counter of an ingress port filter entry.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxIPFGetMatchedCount(*swt_handle_t* *handle, uint32_t entryID, uint64_t *count)

Get the matched count for entry in IPF.

Parameters

- handle –
- entryID –
- count –

Returns
status_t

Returns
See netc_cmd_error_t

static inline *status_t* SWT_RxPSFPInit(*swt_handle_t* *handle, const *netc_swt_psfp_config_t* *config)

Init the Switch PSFP Cover configuration below.

- ISI key construction profiles (ISIDKC0CR0 - ISIDKC3CR0)

Not Cover

- Port specific isi config

Parameters

- handle –
- config –

Returns
status_t

static inline *status_t* SWT_RxPSFPConfigPortISI(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, const *netc_port_psfp_isi_config* *config)

Configure Ingress stream identification for specified port.

Parameters

- handle –
- portIdx –
- config –

Returns
status_t

static inline uint32_t SWT_RxPSFPGetISITableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of stream identification table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Filter, VLAN Filter, FDB, L2 IPV4 Multicast Filter table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxPSFPAddISITableEntry(*swt_handle_t* *handle, *netc_tb_isi_config_t* *config, uint32_t *entryID)

Add an entry into the stream identification table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPQueryISITableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_isi_config_t* *config)

Query an entry in ingress stream identification table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

status_t SWT_RxPSFPQueryISITableEntryWithKey(*swt_handle_t* *handle, *netc_tb_isi_keye_t* *keye, *netc_tb_isi_rsp_data_t* *rsp)

Query an entry in ingress stream identification table using key match.

Parameters

- handle –
- keye –
- rsp –

Returns
status_t

status_t SWT_RxPSFPDelISITableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete an entry in the stream identification table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_RxPSFPGetISTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number of ingress stream table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxPSFPAddISTableEntry(*swt_handle_t* *handle, *netc_tb_is_config_t* *config)

Add an entry into the ingress stream table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPUpdateISTableEntry(*swt_handle_t* *handle, *netc_tb_is_config_t* *config)

Updtae an entry in the ingress stream table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPQueryISTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_is_config_t* *config)

Query an entry in ingress stream table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPDelISTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete an entry in the stream identification table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_RxPSFPGetISFTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of ingress stream filter table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification, VLAN Filter, FDB, L2 IPV4 Multicast Filter table.

Parameters

- handle –

Returns
uint32_t

static inline uint32_t SWT_RxPSFPGetISFTableMaxEntryNum(*swt_handle_t* *handle)

Get maximum available entry number (entry size is 24 bytes) of ingress stream filter table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification, VLAN Filter, FDB, L2 IPV4 Multicast Filter table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxPSFPAddISFTableEntry(*swt_handle_t* *handle, *netc_tb_isf_config_t* *config, uint32_t *entryID)

Add an entry into the ingress stream filter table.

Parameters

- handle –
- config –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPUpdateISFTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_isf_cfge_t* *cfg)

Update an entry in the ingress stream filter table.

Parameters

- handle –
- entryID –
- cfg –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPDelISFTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete an entry in the ingress stream filter table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_RxPSFPGetRPTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number of Rate Policer table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

static inline uint32_t SWT_RxPSFPGetRPTableMaxEntryNum(*swt_handle_t* *handle)

Get maximum available entry number (entry size is 24 bytes) of Rate Policer table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification, VLAN Filter, FDB, L2 IPV4 Multicast Filter table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxPSFPAddRPTableEntry(*swt_handle_t* *handle, *netc_tb_rp_config_t* *config)

Add entry to Rate Policer table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPUpdateRPTableEntry(*swt_handle_t* *handle, *netc_tb_rp_config_t* *config)

Update entry in Rate Policer table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPAddOrUpdateRPTableEntry(*swt_handle_t* *handle, *netc_tb_rp_config_t* *config)

Add or update entry in Rate Policer table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPDelRPTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete entry in the Rate Policer table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPQueryRPTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_rp_rsp_data_t* *rsp)

Query entry in the stream Rate Policer table.

Parameters

- handle –
- entryID –
- rsp –

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPGetRPStatistic(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_rp_stse_t* *statis)

Get statistic of specified Rate Policer entry.

Parameters

- handle –
- entryID –
- statis –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPResetMRRPTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Reset mark red parameter of specified Rate Policer entry.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_RxPSFPGetISCTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number of ingress stream count table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxPSFPQueryISFTableEntry(*swt_handle_t* *handle, *netc_tb_isf_keye_t* *keye, *netc_tb_isf_rsp_data_t* *rsp)

Query entry from the Ingress Stream Filter table.

Parameters

- handle –
- keye –
- rsp –

Returns
status_t

status_t SWT_RxPSFPAddISCTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Add entry in ingress stream count table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPGetISCStatistic(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_isc_stse_t* *statistic)

Get ingress stream count statistic.

Parameters

- handle –
- entryID –
- statistic –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPResetISCStatistic(*swt_handle_t* *handle, uint32_t entryID)

Reset the count of the ingress stream count.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_RxPSFPGetSGITableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number of stream gate instance table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

static inline uint32_t SWT_RxPSFPGetSGITableMaxEntryNum(*swt_handle_t* *handle)

Get maximum entry number of stream gate instance table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxPSFPAddSGITableEntry(*swt_handle_t* *handle, *netc_tb_sgi_config_t* *config)

Add entry in stream gate instance table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPResetIRXOEXSGITableEntry(*swt_handle_t* *handle, uint32_t entryID)

Reset IRX and OEX flag in stream gate instance entry.

Parameters

- handle –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPUpdateSGITableEntry(*swt_handle_t* *handle, *netc_tb_sgi_config_t* *config)

Update entry in stream gate instance table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPDelSGITableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete entry in stream gate instance table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPGetSGIState(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_sgise_t* *state)

Get state of the stream gate instance for specified entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPQuerySGITableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_config_t* *config)

Query entry in the stream gate instance table.

Parameters

- handle –
- entryID –
- config –

Returns
See netc_cmd_error_t

static inline uint32_t SWT_RxPSFPGetSGCLTableRemainWordNum(*swt_handle_t* *handle)

Get remaining available words number of Stream Gate Control List table.

Note: This is a dynamic bounded index table, and number of words required for a stream gate control list is 1+N/2 where N is number of gate time slots in the stream gate control list. The remaining word should be greater than the want added entry size

---

Parameters

- handle –

Returns
uint32_t

static inline uint32_t SWT_RxPSFPGetSGCLTableMaxWordNum(*swt_handle_t* *handle)

Get maximum words number of Stream Gate Control List table.

Parameters

- handle –

Returns
uint32_t

status_t SWT_RxPSFPAddSGCLTableEntry(*swt_handle_t* *handle, *netc_tb_sgcl_gcl_t* *config)

Add entry into Stream Gate Control List Table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPDelSGCLTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete entry of Stream Gate Control List Table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPGetSGCLGateList(*swt_handle_t* *handle, *netc_tb_sgcl_gcl_t* *gcl, uint32_t length)

Get Stream Gate Control List Table entry gate control list.

Parameters

- handle –
- gcl –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_RxPSFPGetSGCLState(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_sgcl_sgclse_t* *state)

Get state (ref count) for Stream Gate Control List table entry.

Parameters

- handle –
- entryID –
- state –

Returns
status_t

Returns
See netc_cmd_error_t

2.92 Switch (SWT) Statistic Module

static inline *status_t* SWT_GetCutThroughForwardStatistic(*swt_handle_t* *handle, uint32_t *statistic)

Get Switch cut through forwarding count.

Get the Cut-through frames counter of frames forwarded to at least one egress port.

Parameters

- handle –
- statistic – pointer to the statistic count data

Returns
kStatus_Success

status_t SWT_GetETMCongestionGroupStatistic(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_etmcg_stse_t* *statistic)

Get statistic of the congestion group.

Parameters

- handle –
- entryID –
- statistic –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_GetETMClassQueueStatistic(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_etmcq_stse_t* *statistic)

Get the statistic for the class queue in specified switch port.

Parameters

- handle –
- entryID –
- statistic –

Returns
status_t

Returns
See netc_cmd_error_t

static inline uint32_t SWT_GetECEntryNum(*swt_handle_t* *handle)

Get egress count table entry number.

Note: This is a static bounded index table, so all entries in the table are available for SWT_GetECStatistic()/SWT_ResetECStatistic() API

Parameters

- handle –

Returns
uint32_t

status_t SWT_GetECStatistic(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_ec_stse_t* *statistic)

Get statistic for specified egress count table entry.

Parameters

- handle –
- entryID –
- statistic –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_ResetECStatistic(*swt_handle_t* *handle, uint32_t entryID)

Reset statistic for specified egress count table entry.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

static inline *status_t* SWT_GetPortDiscardStatistic(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_port_discard_tpye_t* discardType, *netc_port_discard_statistic_t* *statistic)

Get the port discard statistic and reason.

Get the discarded count of frames and its reasons.

Parameters

- handle –
- portIdx – port index
- discardType – Port discard type.
- statistic – pointer to the statistic data

Returns
kStatus_Success

static inline *status_t* SWT_ClearPortDiscardReason(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_port_discard_tpye_t* discardType, uint32_t reason0, uint32_t reason1)

Clean the Port Rx discard reason. Set the related bits to 1 to clear the specific reasons.

Parameters

- handle –
- portIdx – port index
- discardType – Port discard type.
- reason0 –
- reason1 –

Returns
kStatus_Success

static inline *status_t* SWT_GetFDBInUseEntriesNumber(*swt_handle_t* *handle, *netc_switch_inuse_fdb_statistic_t* *statistic)

Clean the Port Rx discard reason. Set the related bits to 1 to clear the specific reasons.

Parameters

- handle –
- statistic –

Returns
kStatus_Success

static inline uint32_t SWT_GetPortTGSListStatus(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx)

Get Switch port time gate scheduling gate list status.

Parameters

- handle –
- portIdx – port index

Returns
Port status flags which are ORed by the enumerators in the netc_port_tgsl_status_t

static inline *status_t* SWT_SetPortIPV2QMR(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, const uint8_t *ipvToTC)

Set IPV to queue mapping on Switch port.

Parameters

- handle –
- portIdx – port index
- ipvToTC[] – queue number of each ipv value

Returns
kStatus_Success

static inline *status_t* SWT_EnablePortMacStationMove(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, bool enable)

Enable MAC station move on Switch port.

Parameters

- handle –
- portIdx – port index
- enable – Allow MAC station move

Returns
kStatus_Success

static inline *status_t* SWT_SetPortSR(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_swt_port_sr_config_t* *sr)

Set the port seamless redundancy configuration.

Set the port seamless redundancy configuration.

Parameters

- handle –
- portIdx – port index
- sr – netc_swt_port_sr_config_t

Returns
kStatus_Success

static inline *status_t* SWT_SetPortGroup(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, uint8_t pgid)

Set the port group configuration.

Set the port group configuration.

Parameters

- handle –
- portIdx – port index
- pgid – Port Group ID

Returns
kStatus_Success

2.93 Switch (SWT) Egress data path configuration

static inline *status_t* SWT_TxSDUConfigPort(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_hw_tc_idx_t* tcIdx, const *netc_port_tc_sdu_config_t* *config)

Configure the transmit maximum SDU for port Traffic Class.

Parameters

- handle –
- portIdx –
- tcIdx –
- config –

Returns
status_t

static inline *status_t* SWT_TxCBSConfigPort(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_hw_tc_idx_t* tcIdx, const *netc_port_tc_cbs_config_t* *config)

Configure the Credit Based Shaper for port Traffic Class.

Parameters

- handle –
- portIdx –
- tcIdx –
- config –

Returns
status_t

status_t SWT_TxTGSConfigAdminGcl(*swt_handle_t* *handle, *netc_tb_tgs_gcl_t* *config)

Config the Time Gate Scheduling entry admin gate control list.

This function is used to program the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxtTGSGetOperGcl(*swt_handle_t* *handle, *netc_tb_tgs_gcl_t* *gcl, uint32_t length)

Get Time Gate Scheduling entry operation gate control list.

This function is used to read the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

Parameters

- handle –
- gcl –
- length –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxTrafficClassConfig(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_hw_tc_idx_t* tcIdx, const *netc_port_tx_tc_config_t* *config)

Config the TC (traffic class) property.

Parameters

- handle –
- portIdx –
- tcIdx –
- config –

Returns
status_t

status_t SWT_TxPortTGSEnable(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, bool enable, uint8_t gateState)

Enable the switch port time gate scheduling.

Parameters

- handle –
- portIdx –
- enable –
- gateState –

Returns
status_t

static inline uint32_t SWT_TxEPPGetETTableRemainEntryNum(*swt_handle_t* *handle)

Get remaining available entry number of egress treatment table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

---

Parameters

- handle –

Returns
uint32_t

status_t SWT_TxEPPAddETTableEntry(*swt_handle_t* *handle, *netc_tb_et_config_t* *config)

Add entry in the egress treatment table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxEPPUpdateETTableEntry(*swt_handle_t* *handle, *netc_tb_et_config_t* *config)

Update entry in the egress treatment table.

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxEPPQueryETTableEntry(*swt_handle_t* *handle, uint32_t entryID, *netc_tb_et_config_t* *config)

Query an entry in the egress treatment table.

Parameters

- handle –
- entryID –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxEPPDelETTableEntry(*swt_handle_t* *handle, uint32_t entryID)

Delete entry for the egress treatment table.

Parameters

- handle –
- entryID –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxEPPQosToVlanConfig(*swt_handle_t* *handle, const *netc_swt_qos_to_vlan_config_t* *config)

Configure switch egress QOS to VLAN map classification.

Parameters

- handle –
- config –

Returns
status_t

status_t SWT_TxETMConfigClassScheduler(*swt_handle_t* *handle, *netc_tb_etmcs_config_t* *config)

Configure the WBFS and Strict priority for each PORT by using ETM Class Scheduler table.

Note: There is one class queue scheduler per switch port, and there is limitation between TSD and class queue scheduler configuration

---

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxETMConfigClassQueue(*swt_handle_t* *handle, *netc_tb_etmcq_config_t* *config)

Configure the class queue to congestion group map by using ETM class queue table.

Note: The number of class queue table entry is equal to the number of class queues in the switch

---

Parameters

- handle –
- config –

Returns
status_t

Returns
See netc_cmd_error_t

status_t SWT_TxETMConfigCongestionGroup(*swt_handle_t* *handle, *netc_tb_etmcg_config_t* *config)

Config congestion group map by using ETM congestion group table.

Note: The number of congestion groups table entry is equal to the number of class queues in the switch

---

Parameters

- handle –

- config –

Returns
status_t

Returns
See netc_cmd_error_t

static inline void SWT_TxTcConfigPreemption(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_hw_tc_idx_t* tcIdx, const bool enable)

Config Preemption for each port TC (traffic class)

Parameters

- handle –
- portIdx –
- tcIdx –
- enable –

static inline void SWT_TxGetTcPreemption(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_hw_tc_idx_t* tcIdx, bool *enabled)

Get Preemption configuration for each port TC (traffic class)

Parameters

- handle –
- portIdx –
- tcIdx –
- enabled –

static inline void SWT_TxPortEthMacConfigPreemption(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, const *netc_port_preemption_config* *config)

Configure Preemption control configuration for an ethernet MAC.

Parameters

- handle –
- portIdx –
- config –

static inline void SWT_TxPortGetEthMacPreemption(*swt_handle_t* *handle, *netc_hw_port_idx_t* portIdx, *netc_port_preemption_config* *config, *netc_port_phy_mac_preemption_status_t* *status)

Get Preemption control configuration from ethernet MAC port.

Parameters

- handle –
- portIdx –
- config –

2.94 Switch (SWT) Transmit/Receive

enum _swt_mgmt_tx_opt_flags

Switch management Tx Option flags.

Values:

enumerator kSWT_TX_OPT_VLAN_INSERT

Enable VLAN insert, only active when use Switch Port masquerading Tx option.

enumerator kSWT_TX_OPT_OFFLOAD

Tx offload.

typedef union *_swt_mgmt_tx_arg_t* swt_mgmt_tx_arg_t

Switch management Tx parameter.

typedef enum *_swt_mgmt_tx_opt_flags* swt_mgmt_tx_opt_flags

Switch management Tx Option flags.

typedef struct *_swt_tx_opt* swt_tx_opt

status_t SWT_ManagementTxRxConfig(*swt_handle_t* *handle, *ep_handle_t* *epHandle, const *swt_transfer_config_t* *txRxConfig)

Set the EP handle used by switch frame management.

Parameters

- handle –
- epHandle – The EP handle
- txRxConfig – Switch transfer config

Return values

status_t –

status_t SWT_SendFrame(*swt_handle_t* *handle, *netc_frame_struct_t* *frame, void *context, *swt_tx_opt* *opt)

Transmits a frame on management port.

Parameters

- handle – The SWT handle
- frame – The frame descriptor pointer
- context – Private context provided back on reclaim
- opt – Tx options.

Return values

status_t –

static inline void SWT_WaitUnitilTxComplete(*swt_handle_t* *handle, bool enMasquerade, uint8_t ring)

Wait until the EP Tx ring has completed the transfer.

Note: Only call after EP_SendFrame() to do a no-interrupt transfer

Parameters

- handle –
- ring – Tx ring.

void SWT_ReclaimTxDescriptor(*swt_handle_t* *handle, uint8_t ring)

Reclaim tx descriptors. This function is used to update the tx descriptor status. For each reclaimed transmit frame the ep_reclaim_cb_t is called.

This is called after being notified of a transmit completion from ISR. It runs until there are no more frames to be reclaimed in the BD ring.

Parameters

- handle –
- ring – Tx ring.

status_t SWT_ReceiveFrame(*swt_handle_t* *handle, *netc_frame_struct_t* *frame, *netc_frame_attr_t* *attr)

Receives management frames (host reason not zero) with zero copy.

Parameters

- handle –
- frame – The frame descriptor pointer
- attr – Frame attribute pointer

Returns
kStatus_Success Successfully receive a management (host reason not zero) frame frame

Returns
kStatus_NETC_RxHRZeroFrame Frame in Rx BD ring is regular frame, need call EP_ReceiveFrame()

Returns
kStatus_NETC_RxTsrResp Frame in Rx BD ring is Transmit Timestamp Reference Response messages, need call SWT_GetTimestampRefResp() to get Transmit Timestamp Reference Response

Returns
kStatus_NETC_RxFrameEmpty Rx BD ring is empty

Returns
kStatus_NETC_RxFrameError Frame in Rx BD ring has error, need be dropped

Returns
kStatus_InvalidArgument No Rx BD ring is available

Returns
kStatus_NETC_LackOfResource Appliction provided buffer is not enough

status_t SWT_ReceiveFrameCopy(*swt_handle_t* *handle, void *buffer, uint32_t length, *netc_frame_attr_t* *attr)

Receives management frame (host reason not zero) which will be copied in the provided buffer.

Note: The buffer size MUST be queried using SWT_GetRxFrameSize() beforehand.

Parameters

- handle –
- buffer – Buffer address
- length – Buffer length
- attr – Frame attribute pointer

Returns
kStatus_Success Successfully receive a management (host reason not zero) frame frame

Returns
kStatus_InvalidArgument No Rx BD ring is available

status_t SWT_GetRxFrameSize(*swt_handle_t* *handle, uint32_t *length)

brief Gets the size of the pending frame in the specified receive ring buffer.

Note: Frame size without FCS, and the size will be zero when Rx BD is Timestamp Reference Response message.

Parameters

- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- length – The length of the valid frame received.

Returns
kStatus_Success Successfully get the length of a management (host reason not zero) frame

Returns
kStatus_NETC_RxHRZeroFrame Frame in Rx BD ring is regular frame, need call EP_GetRxFrameSize() to get frame size

Returns
kStatus_NETC_RxTsrResp Frame in Rx BD ring is Transmit Timestamp Reference Response messages, need call SWT_GetTimestampRefResp() to get Transmit Timestamp Reference Response

Returns
kStatus_NETC_RxFrameEmpty Rx BD ring is empty

Returns
kStatus_NETC_RxFrameError Frame in Rx BD ring has error, need be dropped

Returns
kStatus_InvalidArgument No Rx BD ring is available

union __swt_mgmt_tx_arg_t

#include <fsl_netc_switch.h> Switch management Tx parameter.

Public Members

uint8_t ring

Tx ring index, use for Switch Port masquerading Tx option

struct __swt_mgmt_tx_arg_t

struct __swt_tx_opt

#include <fsl_netc_switch.h>

Public Members

uint8_t ring

Tx ring index.

uint32_t flags
: A bitmask of @swt_mgmt_tx_opt_flags.

netc_enetc_vlan_tag_t vlan
: VLAN tag which will be inserted, used if enVlanInsert is set.

netc_tx_offload_t offload
: Offload parameters.

struct __swt__transfer__config
: *#include <fsl_netc_switch.h>* Transfer configuration structure for Switch.

Note: When need use Switch/EP receive APIs, recommended enable mgmtRxBdrConfig to use special MgmtRxBdRing rings to receive these host reason not-zero frames

---

Public Members

bool rxCacheMaintain
: Enable/Disable Rx buffer cache maintain in driver.

bool txCacheMaintain
: Enable/Disable Tx buffer cache maintain in driver.

bool rxZeroCopy
: Enable zero-copy receive mode.

swt_rx_alloc_cb_t rxBuffAlloc
: Callback function to alloc memory, must be provided for zero-copy Rx.

swt_rx_free_cb_t rxBuffFree
: Callback function to free memory, must be provided for zero-copy Rx.

bool enUseMgmtRxBdRing
: Enable/Disable use Switch management Rx BD ring, if disabled, the Switch/EP receive APIs (SWT_GetRxFrameSize()/SWT_ReceiveFrameCopy()/SWT_GetTimestampRefResp()/SWT_ReceiveF will use EP Rx BD ring 0 to receive frames.

netc_rx_bdr_config_t mgmtRxBdrConfig
: Switch management Rx BD ring configuration.

bool enUseMgmtTxBdRing
: Enable/Disable use Switch management Tx BD ring, if disabled, can't use Switch transfer API (SWT_SendFrame()) to send frames.

netc_tx_bdr_config_t mgmtTxBdrConfig
: Switch management Rx BD ring configuration.

swt_reclaim_cb_t reclaimCallback
: Callback for reclaimed Tx Switch management frames.

void *userData
: User data, return in callback.

struct ___unnamed204___

Public Members

uint8_t ipv
Internal Priority Value, use for Direct Switch Enqueue Tx option

uint8_t dr
Discard Resiliance, use for Direct Switch Enqueue Tx option

2.95 NETC Timer Driver

2.96 Timer adjustment module

void NETC_TimerGetTime(ENETC_PF_TMR_Type *base, uint64_t *nanosecond)
Get the timer's current or default time.

Parameters

- base – NETC timer base address.
- nanosecond – Time in nanosecond.

void NETC_TimerGetCurrentTime(*netc_timer_handle_t* *handle, uint64_t *nanosecond)
Get the timer's current time.

Parameters

- handle – NETC timer handle.
- nanosecond – Time in nanosecond.

void NETC_TimerGetFreeRunningTime(*netc_timer_handle_t* *handle, uint64_t *nanosecond)
Get the timer's freerunning time.

Parameters

- handle – NETC timer handle.
- nanosecond – Time in nanosecond.

void NETC_TimerAddOffset(*netc_timer_handle_t* *handle, int64_t nanosecond)
Correct the current timer.

Note: Need stop all mechanims based on 1588 timers during call this API. To take time gate scheduling as example, if the port is enabled with TGS and it contains an operational list, user need to call SWT_TxPortTGSEnable()/SWT_TxPortTGSEnable() to disable the port time gate before call this API, and re-enable and configure the port TGS after the execution of this API.

Parameters

- handle – NETC timer handle.
- nanosecond – Time in nanosecond.

void NETC_TimerAdjustFreq(*netc_timer_handle_t* *handle, int32_t ppb)
Adjust the timer frequency.

Parameters

- handle – NETC timer handle.
- ppb – Parts per billion.

void NETC_TimerGetFrtSrtTime(*netc_timer_handle_t* *handle, uint64_t *frt, uint64_t *srt)

Get the free running and synchronized current time with an atomic read.

Parameters

- handle – NETC timer handle.
- frt – The free-running time in nanosecond.
- srt – The synchronized current time in nanosecond.

2.97 Timer initialization module

enum __netc_timer_irq_flags

Timer interrupt flags.

Values:

enumerator kNETC_TimerFiper1IrqFlag

Periodic pulse 1 interrupt.

enumerator kNETC_TimerFiper2IrqFlag

Periodic pulse 2 interrupt.

enumerator kNETC_TimerFiper3IrqFlag

Periodic pulse 3 interrupt.

enumerator kNETC_TimerAlarm1IrqFlag

Alarm 1 interrupt.

enumerator kNETC_TimerAlarm2IrqFlag

Alarm 2 interrupt.

enumerator kNETC_TimerExtTrig1ThresholdIrqFlag

External trigger 1 timestamp FIFO threshold hit interrupt.

enumerator kNETC_TimerExtTrig2ThresholdIrqFlag

External trigger 2 timestamp FIFO threshold hit interrupt.

enumerator kNETC_TimerExtTrig1TsAvailIrqFlag

External trigger 1 new timestamp available interrupt.

enumerator kNETC_TimerExtTrig2TsAvailIrqFlag

External trigger 2 new timestamp available interrupt.

enumerator kNETC_TimerExtTrig1OverflowIrqFlag

External trigger 1 timestamp FIFO overflow interrupt.

enumerator kNETC_TimerExtTrig2OverflowIrqFlag

External trigger 2 timestamp FIFO overflow interrupt.

typedef enum *_netc_timer_irq_flags* netc_timer_irq_flags_t

Timer interrupt flags.

typedef struct *_netc_timer_handle* netc_timer_handle_t

typedef struct *_netc_timer_config* netc_timer_config_t

Structure to configure timer.

status_t NETC_TimerInit(*netc_timer_handle_t* *handle, const *netc_timer_config_t* *config)

Initialize the NETC PTP1588 timer.

Parameters

- handle – NETC timer handle.
- config – The configuration of the timer.

Returns

status_t

void NETC_TimerDeinit(*netc_timer_handle_t* *handle)

Deinitialize the NETC PTP1588 timer.

Parameters

- handle – NETC timer handle.

void NETC_TimerInitHandle(*netc_timer_handle_t* *handle)

Initialize a NETC PTP1588 timer handle.

Parameters

- handle – NETC timer handle.

void NETC_TimerEnable(*netc_timer_handle_t* *handle, bool enable)

Enable/Disable the NETC PTP1588 timer.

Parameters

- handle – NETC timer handle.
- enable – Whether enable the PTP1588 timer.

struct _netc_timer_handle

#include <fsl_netc_timer.h> Timer handler structure.

Public Members

netc_timer_hw_t hw

Hardware register map resource.

uint32_t timerFreq

Timer clock frequency(Hz).

uint8_t entryNum

MSIX entry number.

struct _netc_timer_config

#include <fsl_netc_timer.h> Structure to configure timer.

Public Members

bool clkOutputPhase

True: Inverted divided clock is output, False: Non-inverted divided clock is output.

bool clkInputPhase

True: Inverted frequency tuned timer input clock, False: Non-inverted frequency tuned timer input clock.

bool enableTimer

True: Enable 1588 timer, False: Disable 1588 timer, use default counter.

netc_timer_ref_clk_t clockSelect
Timer reference clock.

uint32_t refClkHz
Timer reference clock frequency in Hz.

int32_t defaultPpb
Default ppb.

netc_msix_entry_t *msixEntry
MSIX table entry array.

uint8_t entryNum
MSIX entry number.

2.98 Local time synchronization module

enum _netc_timer_alarm_index
Enumeration for NETC timer alarm index.

Values:

enumerator kNETC_TimerAlarm1

enumerator kNETC_TimerAlarm2

enum _netc_timer_fiper_index
Enumeration for NETC timer FIPER index.

Values:

enumerator kNETC_TimerFiper1

enumerator kNETC_TimerFiper2

enumerator kNETC_TimerFiper3

typedef enum *_netc_timer_alarm_index* netc_timer_alarm_index_t
Enumeration for NETC timer alarm index.

typedef struct *_netc_timer_alarm_t* netc_timer_alarm_t
Structure to configure timer alarm.

typedef enum *_netc_timer_fiper_index* netc_timer_fiper_index_t
Enumeration for NETC timer FIPER index.

typedef struct *_netc_timer_fiper_config* netc_timer_fiper_config_t
Structure to configure timer FIPER.

typedef struct *_netc_timer_fiper* netc_timer_fiper_t
Structure to set and start timer FIPER.

typedef struct *_netc_timer_ext_pulse_trig* netc_timer_ext_trig_t
Structure to configure external pulse trigger timestamp.

void NETC_TimerConfigureAlarm(*netc_timer_handle_t* *handle, *netc_timer_alarm_index_t* alarmId, const *netc_timer_alarm_t* *alarm)
Configure the timer alarm feature.

Parameters

- handle – NETC timer handle.

- alarmId – The alarm index.
- alarm – The timer alarm configuration structure.

void NETC_TimerStartAlarm(*netc_timer_handle_t* *handle, *netc_timer_alarm_index_t* alarmId, uint64_t nanosecond)

Start the alarm with specified time after alarm feature is configured This function can generate a pulse on a GPIO and/or an interrupt at specified future time. It also can trigger FIPER at specified time.

Parameters

- handle – NETC timer handle.
- alarmId – The alarm index.
- nanosecond – The time in nanosecond to generate alarm pulse.

void NETC_TimerStopAlarm(*netc_timer_handle_t* *handle, *netc_timer_alarm_index_t* alarmId)

Stop the alarm before/after it's fired This function can deactivate alarm.

Parameters

- handle – NETC timer handle.
- alarmId – The alarm index.

void NETC_TimerConfigureFIPER(*netc_timer_handle_t* *handle, const *netc_timer_fiper_config_t* *config)

Configure the timer FIPER feature.

Parameters

- handle – NETC timer handle.
- config – The timer FIPER configuration structure.

void NETC_TimerStartFIPER(*netc_timer_handle_t* *handle, *netc_timer_fiper_index_t* fiperId, const *netc_timer_fiper_t* *fiper)

Start the timer FIPER to generate pulse This function can generate a periodic(Fixed Period-FIPER) pulse on a GPIO pin and/or an interrupt to the host.

Parameters

- handle – NETC timer handle.
- fiperId – The timer FIPER index.
- fiper – The timer FIPER configuration structure.

void NETC_TimerStopFIPER(*netc_timer_handle_t* *handle, *netc_timer_fiper_index_t* fiperId)

Stop the timer FIPER to generate pulse.

Parameters

- handle – NETC timer handle.
- fiperId – The timer FIPER index.

void NETC_TimerConfigureExtPulseTrig(*netc_timer_handle_t* *handle, netc_timer_exttrig_index_t extTrigId, const *netc_timer_ext_trig_t* *extTrig)

Configure the external pulse trigger timestamp capture.

Parameters

- handle – NETC timer handle.
- extTrigId – The timer FIPER index.

- extTrig – The external pulse trigger configuration structure.

status_t NETC_TimerSetTsFifoThreshold(*netc_timer_handle_t* *handle, uint8_t threshold)

Set timestamp FIFO threshold of the external pulse trigger timestamp capture.

Parameters

- handle – NETC timer handle.
- threshold – Timestamp FIFO threshold.

Returns

status_t

status_t NETC_TimerReadExtPulseCaptureTime(*netc_timer_handle_t* *handle, netc_timer_exttrig_index_t extTrigId, uint64_t *nanosecond)

Read the timestamp captured by external pulse trigger in FIFO.

Parameters

- handle – NETC timer handle.
- extTrigId – The timer FIPER index.
- nanosecond – Timestamp in nanosecond.

Returns

status_t

static inline void NETC_TimerClearInterruptStatus(*netc_timer_handle_t* *handle, uint32_t flags)

Clear timer interrupt flags.

Parameters

- handle – NETC timer handle.
- flags – Timer interrupt flags. This is a logical OR of enumeration :: netc_timer_irq_flags_t.

status_t NETC_TimerMsixSetGlobalMask(*netc_timer_handle_t* *handle, bool mask)

Set the global MSIX mask status.

This function masks/unmasks global MSIX message. Mask - All of the vectors are masked, regardless of their per-entry mask bit states. Unmask - Each entry's mask status determines whether the vector is masked or not.

Parameters

- handle – The timer handle
- mask – The mask state. True: Mask, False: Unmask.

Returns

status_t

status_t NETC_TimerMsixSetEntryMask(*netc_timer_handle_t* *handle, uint8_t entryIdx, bool mask)

Set the MSIX entry mask status for specified entry.

This function masks/unmasks MSIX message for specified entry.

Parameters

- handle – NETC timer handle.
- entryIdx – The entry index in the table.
- mask – The mask state. True: Mask, False: Unmask.

Returns
status_t

status_t NETC_TimerMsixGetPendingStatus(*netc_timer_handle_t* *handle, uint8_t pbaIdx, uint64_t *status)

Get the MSIX pending status in MSIX PBA table.

This function is to get the entry pending status from MSIX PBA table. If interrupt occurs but masked by vector control of entry, pending bit in PBA will be set.

Parameters

- handle – NETC timer handle.
- pbaIdx – The index of PBA array with 64-bit unit.
- status – Pending status bit mask, bit n for entry n.

Returns
status_t

struct _netc_timer_alarm_t

#include <fsl_netc_timer.h> Structure to configure timer alarm.

Public Members

bool enableInterrupt

Enable/Disable ALARM interrupt enable.

bool polarity

True: Active low output, False: Active high output.

bool pulseGenSync

True: ALARM output asserted synchronous to timer generated clock, False: ALARM output asserted immediately.

uint8_t pulseWidth

Pulse width in number of timer generated clocks the alarm will be active for.

struct _netc_timer_fiper_config

#include <fsl_netc_timer.h> Structure to configure timer FIPER.

Public Members

bool startCondition

True: FIPER is enabled through timer enable and alarm getting set, False: FIPER is enabled through timer enable.

bool fiper1Loopback

True: FIPER1 pulse is looped back into Trigger1 input, False: Trigger1 input is based upon normal external trigger input.

bool fiper2Loopback

True: FIPER2 pulse is looped back into Trigger2 input, False: Trigger2 input is based upon normal external trigger input.

uint16_t prescale

Output FIPER pulse clock is generated by dividing the timer input clock by this number. Must be an even value.

struct _netc_timer_fiper

#include <fsl_netc_timer.h> Structure to set and start timer FIPER.

Public Members

bool enableInterrupt
: Enable/Disable FIPER interrupt interrupt.

bool pulseGenSync
: True: FIPER output asserted synchronous to timer generated clock, False: FIPER output asserted immediately.

uint8_t pulseWidth
: FIPER pulse width.

uint32_t pulsePeriod
: Interval of FIPER pulses.

struct __netc_timer_ext_pulse_trig
: *#include <fsl_netc_timer.h>* Structure to configure external pulse trigger timestamp.

Public Members

bool polarity
: Time stamp on the falling(true)/rising(false) edge of the external trigger.

bool enableFifoOverflowInterrupt
: Enable/Disable FIFO Overflow interrupt.

bool enableFifoThresholdHitInterrupt
: Enable/Disable FIFO Threshold Hit interrupt.

bool enableTsAvailInterrupt
: Enable/Disable timestamp capture interrupt.

2.99 OTFAD: On The Fly AES-128 Decryption Driver

void OTFAD_GetDefaultConfig(*otfad_config_t* *config)
: OTFAD module initialization function.

Parameters

- config – OTFAD configuration.

void OTFAD_Init(OTFAD_Type *base, const *otfad_config_t* *config)
: OTFAD module initialization function.

Parameters

- base – OTFAD base address.
- config – OTFAD configuration.

void OTFAD_Deinit(OTFAD_Type *base)
: Deinitializes the OTFAD.

static inline uint32_t OTFAD_GetOperateMode(OTFAD_Type *base)
: OTFAD module get operate mode.

Parameters

- base – OTFAD base address.

static inline uint32_t OTFAD_GetStatus(OTFAD_Type *base)

OTFAD module get status.

Parameters

- base – OTFAD base address.

status_t OTFAD_SetEncryptionConfig(OTFAD_Type *base, const *otfad_encryption_config_t* *config)

OTFAD module set encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

Parameters

- base – OTFAD base address.
- config – encryption configuration.

status_t OTFAD_GetEncryptionConfig(OTFAD_Type *base, *otfad_encryption_config_t* *config)

OTFAD module get encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

Parameters

- base – OTFAD base address.
- config – encryption configuration.

status_t OTFAD_HitDetermination(OTFAD_Type *base, uint32_t address, uint8_t *contextIndex)

OTFAD module do hit determination.

Parameters

- base – OTFAD base address.
- address – the physical address space assigned to the QuadSPI(FlexSPI) module.
- contextIndex – hitted context region index.

Returns

status, such as kStatus_Success or kStatus_OTFAD_ResRegAccessMode.

FSL_OTFAD_DRIVER_VERSION

Driver version.

Status codes for the OTFAD driver.

Values:

enumerator kStatus_OTFAD_ResRegAccessMode

Restricted register mode

enumerator kStatus_OTFAD_AddressError

End address less than start address

enumerator kStatus_OTFAD_RegionOverlap

the OTFAD does not support any form of memory region overlap, for system accesses that hit in multiple contexts or no contexts, the fetched data is simply bypassed

enumerator kStatus_OTFAD_RegionMiss
For accesses that hit in a single context, but not the selected one

OTFAD context type.

Values:

enumerator kOTFAD_Context_0
context 0

enumerator kOTFAD_Context_1
context 1

enumerator kOTFAD_Context_2
context 2

enumerator kOTFAD_Context_3
context 3

OTFAD operate mode.

Values:

enumerator kOTFAD_NRM
Normal Mode

enumerator kOTFAD_SVM
Security Violation Mode

enumerator kOTFAD_LDM
Logically Disabled Mode

typedef struct *_otfad_encryption_config* otfad_encryption_config_t
OTFAD encryption configuration structure.

typedef struct *_otfad_config* otfad_config_t
OTFAD configuration structure.

struct _otfad_encryption_config
#include <fsl_otfad.h> OTFAD encryption configuration structure.

Public Members

bool valid
The context is valid or not

bool AESdecryption
AES decryption enable

uint8_t readOnly
read write attribute for the entire set of context registers

uint8_t contextIndex
OTFAD context index

uint32_t startAddr
Start address

uint32_t endAddr
End address

uint32_t key[4]
Encryption key

uint32_t counter[2]
Encryption counter

struct _otfad_config
#include <fsl_otfad.h> OTFAD configuration structure.

Public Members

bool forceSVM
Force entry into SVM after a write

bool forceLDM
Force entry into LDM after a write

bool restrictedRegAccess
Restricted register access enable

bool enableOTFAD
OTFAD has decryption enabled

2.100 PDM: Microphone Interface

2.101 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,
    .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

2.102 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

2.103 PWM: Pulse Width Modulator

status_t PWM_Init(PWM_Type *base, *pwm_submodule_t* subModule, const *pwm_config_t* *config)

Ungates the PWM submodule clock and configures the peripheral for basic operation.

This API should be called at the beginning of the application using the PWM driver. When user select PWMX, user must choose edge aligned output, becasue there are some limitation on center aligned PWMX output. When output PWMX in center aligned mode, VAL1 register controls both PWM period and PWMX duty cycle, PWMA and PWMB output will be corrupted. But edge aligned PWMX output do not have such limit. In master reload counter initialization mode, PWM period is depended by period of set LDOK in submodule 0 because this operation will reload register. Submodule 0 counter initialization cannot be master sync or master reload.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- config – Pointer to user's PWM config structure.

Returns
kStatus_Success means success; else failed.

void PWM_Deinit(PWM_Type *base, *pwm_submodule_t* subModule)

Gate the PWM submodule clock.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to deinitialize

void PWM_GetDefaultConfig(*pwm_config_t* *config)

Fill in the PWM config struct with the default settings.

The default values are:

```
config->enableDebugMode = false;
config->enableWait = false;
config->reloadSelect = kPWM_LocalReload;
config->clockSource = kPWM_BusClock;
config->prescale = kPWM_Prescale_Divide_1;
config->initializationControl = kPWM_Initialize_LocalSync;
config->forceTrigger = kPWM_Force_Local;
config->reloadFrequency = kPWM_LoadEveryOportunity;
config->reloadLogic = kPWM_ReloadImmediate;
config->pairOperation = kPWM_Independent;
```

Parameters

- config – Pointer to user's PWM config structure.

status_t PWM_SetupPwm(PWM_Type *base, *pwm_submodule_t* subModule, const *pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Sets up the PWM signals for a PWM submodule.

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. When user select PWMX, user must choose edge aligned output, becasue there are some limitation on center aligned PWMX output. Due to edge aligned PWMX is negative true signal, need to configure PWMX active low true level to get correct duty cycle. The half cycle point will not be exactly in the middle of the PWM cycle when PWMX enabled.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- chnlParams – Array of PWM channel parameters to configure the channel(s).
- numOfChnls – Number of channels to configure, this should be the size of the array passed in. Array size should not be more than 3 as each submodule has 3 pins to output PWM.
- mode – PWM operation mode, options available in enumeration pwm_mode_t
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – PWM source clock of correspond submodule in Hz. If source clock of submodule1,2,3 is from submodule0 AUX_CLK, its source clock is submodule0 source clock divided with submodule0 prescaler value instead of submodule0 source clock.

Returns

Returns kStatus_Fail if there was error setting up the signal; kStatus_Success otherwise

status_t PWM_SetupPwmPhaseShift(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint8_t shiftvalue, bool doSync)

Set PWM phase shift for PWM channel running on channel PWM_A, PWM_B which with 50% duty cycle.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – PWM main counter clock in Hz.
- shiftvalue – Phase shift value, range in 0 ~ 50
- doSync – true: Set LDOK bit for the submodule list; false: LDOK bit don't set, need to call PWM_SetPwmLdok to sync update.

Returns

Returns kStatus_Fail if there was error setting up the signal; kStatus_Success otherwise

void PWM_UpdatePwmDutycycle(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmSignal, *pwm_mode_t* currPwmMode, uint8_t dutyCyclePercent)

Updates the PWM signal's dutycycle.

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update
- currPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

void PWM_UpdatePwmDutycycleHighAccuracy(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmSignal, *pwm_mode_t* currPwmMode, uint16_t dutyCycle)

Updates the PWM signal's dutycycle with 16-bit accuracy.

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update
- currPwmMode – The current PWM mode set during PWM setup
- dutyCycle – New PWM pulse width, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle)

void PWM_UpdatePwmPeriodAndDutycycle(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmSignal, *pwm_mode_t* currPwmMode, uint16_t pulseCnt, uint16_t dutyCycle)

Update the PWM signal's period and dutycycle for a PWM submodule.

The function updates PWM signal period generated by a specific submodule according to the parameters passed in by the user. This function can also set dutycycle weather you want to keep original dutycycle or update new dutycycle. Call this function in local sync control mode because PWM period is depended by

INIT and VAL1 register of each submodule. In master sync initialization control mode, call this function to update INIT and VAL1 register of all submodule because PWM period is depended by INIT and VAL1 register in submodule0. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. PWM signal will not be generated if its period is less than dead time duration.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

- pwmSignal – Signal (PWM A or PWM B) to update
- currPwmMode – The current PWM mode set during PWM setup, options available in enumeration pwm_mode_t
- pulseCnt – New PWM period, value should be between 0 to 65535 0=minimum PWM period... 65535=maximum PWM period
- dutyCycle – New PWM pulse width of channel, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle) You can keep original duty cycle or update new duty cycle

static inline void PWM_EnableInterrupts(PWM_Type *base, *pwm_submodule_t* subModule, uint32_t mask)

Enables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The interrupts to enable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t

static inline void PWM_DisableInterrupts(PWM_Type *base, *pwm_submodule_t* subModule, uint32_t mask)

Disables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The interrupts to enable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t

static inline uint32_t PWM_GetEnabledInterrupts(PWM_Type *base, *pwm_submodule_t* subModule)

Gets the enabled PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

Returns

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

static inline void PWM_DMAFIFOWatermarkControl(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_watermark_control_t* pwm_watermark_control)

Capture DMA Enable Source Select.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwm_watermark_control – PWM FIFO watermark and control

static inline void PWM_DMACaptureSourceSelect(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_dma_source_select_t* pwm_dma_source_select)

Capture DMA Enable Source Select.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwm_dma_source_select – PWM capture DMA enable source select

static inline void PWM_EnableDMACapture(PWM_Type *base, *pwm_submodule_t* subModule, uint16_t mask, bool activate)

Enables or disables the selected PWM DMA Capture read request.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The DMA to enable or disable. This is a logical OR of members of the enumeration pwm_dma_enable_t
- activate – true: Enable DMA read request; false: Disable DMA read request

static inline void PWM_EnableDMAWrite(PWM_Type *base, *pwm_submodule_t* subModule, bool activate)

Enables or disables the PWM DMA write request.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- activate – true: Enable DMA write request; false: Disable DMA write request

static inline uint32_t PWM_GetStatusFlags(PWM_Type *base, *pwm_submodule_t* subModule)

Gets the PWM status flags.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

Returns

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

static inline void PWM_ClearStatusFlags(PWM_Type *base, *pwm_submodule_t* subModule, uint32_t mask)

Clears the PWM status flags.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The status flags to clear. This is a logical OR of members of the enumeration pwm_status_flags_t

static inline void PWM_StartTimer(PWM_Type *base, uint8_t subModulesToStart)

Starts the PWM counter for a single or multiple submodules.

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

Parameters

- base – PWM peripheral base address

- subModulesToStart – PWM submodules to start. This is a logical OR of members of the enumeration pwm_module_control_t

static inline void PWM_StopTimer(PWM_Type *base, uint8_t subModulesToStop)

Stops the PWM counter for a single or multiple submodules.

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.

Parameters

- base – PWM peripheral base address
- subModulesToStop – PWM submodules to stop. This is a logical OR of members of the enumeration pwm_module_control_t

FSL_PWM_DRIVER_VERSION

Version 2.9.1

enum _pwm_submodule

List of PWM submodules.

Values:

enumerator kPWM_Module_0

Submodule 0

enumerator kPWM_Module_1

Submodule 1

enumerator kPWM_Module_2

Submodule 2

enum _pwm_channels

List of PWM channels in each module.

Values:

enumerator kPWM_PwmB

enumerator kPWM_PwmA

enumerator kPWM_PwmX

enum _pwm_value_register

List of PWM value registers.

Values:

enumerator kPWM_ValueRegister_0

PWM Value0 register

enumerator kPWM_ValueRegister_1

PWM Value1 register

enumerator kPWM_ValueRegister_2

PWM Value2 register

enumerator kPWM_ValueRegister_3

PWM Value3 register

enumerator kPWM_ValueRegister_4

PWM Value4 register

enumerator kPWM_ValueRegister_5

PWM Value5 register

enum _pwm_value_register_mask

List of PWM value registers mask.

Values:

enumerator kPWM_ValueRegisterMask_0

PWM Value0 register mask

enumerator kPWM_ValueRegisterMask_1

PWM Value1 register mask

enumerator kPWM_ValueRegisterMask_2

PWM Value2 register mask

enumerator kPWM_ValueRegisterMask_3

PWM Value3 register mask

enumerator kPWM_ValueRegisterMask_4

PWM Value4 register mask

enumerator kPWM_ValueRegisterMask_5

PWM Value5 register mask

enum _pwm_clock_source

PWM clock source selection.

Values:

enumerator kPWM_BusClock

Device specific IPBus clock, refer reference manual for frequency

enumerator kPWM_ExternalClock

EXT_CLK is used as the clock

enumerator kPWM_Submodule0Clock

Clock of the submodule 0 (AUX_CLK) is used as the source clock

enum _pwm_clock_prescale

PWM prescaler factor selection for clock source.

Values:

enumerator kPWM_Prescale_Divide_1

PWM clock frequency = fclk/1

enumerator kPWM_Prescale_Divide_2

PWM clock frequency = fclk/2

enumerator kPWM_Prescale_Divide_4

PWM clock frequency = fclk/4

enumerator kPWM_Prescale_Divide_8

PWM clock frequency = fclk/8

enumerator kPWM_Prescale_Divide_16

PWM clock frequency = fclk/16

enumerator kPWM_Prescale_Divide_32

PWM clock frequency = fclk/32

enumerator kPWM_Prescale_Divide_64

PWM clock frequency = fclk/64

enumerator kPWM_Prescale_Divide_128
PWM clock frequency = fclk/128

enum _pwm_force_output_trigger
Options that can trigger a PWM FORCE_OUT.

Values:

enumerator kPWM_Force_Local
The local force signal, CTRL2[FORCE], from the submodule is used to force updates

enumerator kPWM_Force_Master
The master force signal from submodule 0 is used to force updates

enumerator kPWM_Force_LocalReload
The local reload signal from this submodule is used to force updates without regard to the state of LDOK

enumerator kPWM_Force_MasterReload
The master reload signal from submodule 0 is used to force updates if LDOK is set

enumerator kPWM_Force_LocalSync
The local sync signal from this submodule is used to force updates

enumerator kPWM_Force_MasterSync
The master sync signal from submodule0 is used to force updates

enumerator kPWM_Force_External
The external force signal, EXT_FORCE, from outside the PWM module causes updates

enumerator kPWM_Force_ExternalSync
The external sync signal, EXT_SYNC, from outside the PWM module causes updates

enum _pwm_output_state
PWM channel output status.

Values:

enumerator kPWM_HighState
The output state of PWM channel is high

enumerator kPWM_LowState
The output state of PWM channel is low

enumerator kPWM_NormalState
The output state of PWM channel is normal

enumerator kPWM_InvertState
The output state of PWM channel is invert

enumerator kPWM_MaskState
The output state of PWM channel is mask

enum _pwm_init_source
PWM counter initialization options.

Values:

enumerator kPWM_Initialize_LocalSync
Local sync causes initialization

enumerator kPWM_Initialize_MasterReload
Master reload from submodule 0 causes initialization

enumerator kPWM_Initialize_MasterSync
Master sync from submodule 0 causes initialization

enumerator kPWM_Initialize_ExtSync
EXT_SYNC causes initialization

enum __pwm_load_frequency
PWM load frequency selection.

Values:

enumerator kPWM_LoadEveryOportunity
Every PWM opportunity

enumerator kPWM_LoadEvery2Oportunity
Every 2 PWM opportunities

enumerator kPWM_LoadEvery3Oportunity
Every 3 PWM opportunities

enumerator kPWM_LoadEvery4Oportunity
Every 4 PWM opportunities

enumerator kPWM_LoadEvery5Oportunity
Every 5 PWM opportunities

enumerator kPWM_LoadEvery6Oportunity
Every 6 PWM opportunities

enumerator kPWM_LoadEvery7Oportunity
Every 7 PWM opportunities

enumerator kPWM_LoadEvery8Oportunity
Every 8 PWM opportunities

enumerator kPWM_LoadEvery9Oportunity
Every 9 PWM opportunities

enumerator kPWM_LoadEvery10Oportunity
Every 10 PWM opportunities

enumerator kPWM_LoadEvery11Oportunity
Every 11 PWM opportunities

enumerator kPWM_LoadEvery12Oportunity
Every 12 PWM opportunities

enumerator kPWM_LoadEvery13Oportunity
Every 13 PWM opportunities

enumerator kPWM_LoadEvery14Oportunity
Every 14 PWM opportunities

enumerator kPWM_LoadEvery15Oportunity
Every 15 PWM opportunities

enumerator kPWM_LoadEvery16Oportunity
Every 16 PWM opportunities

enum __pwm_fault_input
List of PWM fault selections.

Values:

enumerator kPWM_Fault_0
Fault 0 input pin

enumerator kPWM_Fault_1
Fault 1 input pin

enumerator kPWM_Fault_2
Fault 2 input pin

enumerator kPWM_Fault_3
Fault 3 input pin

enum _pwm_fault_disable
List of PWM fault disable mapping selections.

Values:

enumerator kPWM_FaultDisable_0
Fault 0 disable mapping

enumerator kPWM_FaultDisable_1
Fault 1 disable mapping

enumerator kPWM_FaultDisable_2
Fault 2 disable mapping

enumerator kPWM_FaultDisable_3
Fault 3 disable mapping

enum _pwm_fault_channels
List of PWM fault channels.

Values:

enumerator kPWM_faultchannel_0

enum _pwm_input_capture_edge
PWM capture edge select.

Values:

enumerator kPWM_Disable
Disabled

enumerator kPWM_FallingEdge
Capture on falling edge only

enumerator kPWM_RisingEdge
Capture on rising edge only

enumerator kPWM_RiseAndFallEdge
Capture on rising or falling edge

enum _pwm_force_signal
PWM output options when a FORCE_OUT signal is asserted.

Values:

enumerator kPWM_UsePwm
Generated PWM signal is used by the deadtime logic.

enumerator kPWM_InvertedPwm
Inverted PWM signal is used by the deadtime logic.

enumerator kPWM_SoftwareControl
Software controlled value is used by the deadtime logic.

enumerator kPWM_UseExternal
PWM_EXTA signal is used by the deadtime logic.

enum _pwm_chnl_pair_operation
Options available for the PWM A & B pair operation.

Values:

enumerator kPWM_Independent
PWM A & PWM B operate as 2 independent channels

enumerator kPWM_ComplementaryPwmA
PWM A & PWM B are complementary channels, PWM A generates the signal

enumerator kPWM_ComplementaryPwmB
PWM A & PWM B are complementary channels, PWM B generates the signal

enum _pwm_register_reload
Options available on how to load the buffered-registers with new values.

Values:

enumerator kPWM_ReloadImmediate
Buffered-registers get loaded with new values as soon as LDOK bit is set

enumerator kPWM_ReloadPwmHalfCycle
Registers loaded on a PWM half cycle

enumerator kPWM_ReloadPwmFullCycle
Registers loaded on a PWM full cycle

enumerator kPWM_ReloadPwmHalfAndFullCycle
Registers loaded on a PWM half & full cycle

enum _pwm_fault_recovery_mode
Options available on how to re-enable the PWM output when recovering from a fault.

Values:

enumerator kPWM_NoRecovery
PWM output will stay inactive

enumerator kPWM_RecoverHalfCycle
PWM output re-enabled at the first half cycle

enumerator kPWM_RecoverFullCycle
PWM output re-enabled at the first full cycle

enumerator kPWM_RecoverHalfAndFullCycle
PWM output re-enabled at the first half or full cycle

enum _pwm_interrupt_enable
List of PWM interrupt options.

Values:

enumerator kPWM_CompareVal0InterruptEnable
PWM VAL0 compare interrupt

enumerator kPWM_CompareVal1InterruptEnable
PWM VAL1 compare interrupt

enumerator kPWM_CompareVal2InterruptEnable
PWM VAL2 compare interrupt

enumerator kPWM_CompareVal3InterruptEnable
PWM VAL3 compare interrupt

enumerator kPWM_CompareVal4InterruptEnable
PWM VAL4 compare interrupt

enumerator kPWM_CompareVal5InterruptEnable
PWM VAL5 compare interrupt

enumerator kPWM_CaptureX0InterruptEnable
PWM capture X0 interrupt

enumerator kPWM_CaptureX1InterruptEnable
PWM capture X1 interrupt

enumerator kPWM_CaptureB0InterruptEnable
PWM capture B0 interrupt

enumerator kPWM_CaptureB1InterruptEnable
PWM capture B1 interrupt

enumerator kPWM_CaptureA0InterruptEnable
PWM capture A0 interrupt

enumerator kPWM_CaptureA1InterruptEnable
PWM capture A1 interrupt

enumerator kPWM_ReloadInterruptEnable
PWM reload interrupt

enumerator kPWM_ReloadErrorInterruptEnable
PWM reload error interrupt

enumerator kPWM_Fault0InterruptEnable
PWM fault 0 interrupt

enumerator kPWM_Fault1InterruptEnable
PWM fault 1 interrupt

enumerator kPWM_Fault2InterruptEnable
PWM fault 2 interrupt

enumerator kPWM_Fault3InterruptEnable
PWM fault 3 interrupt

enum __pwm_status_flags
List of PWM status flags.

Values:

enumerator kPWM_CompareVal0Flag
PWM VAL0 compare flag

enumerator kPWM_CompareVal1Flag
PWM VAL1 compare flag

enumerator kPWM_CompareVal2Flag
PWM VAL2 compare flag

enumerator kPWM_CompareVal3Flag
PWM VAL3 compare flag

enumerator kPWM_CompareVal4Flag
PWM VAL4 compare flag

enumerator kPWM_CompareVal5Flag
PWM VAL5 compare flag

enumerator kPWM_CaptureX0Flag
PWM capture X0 flag

enumerator kPWM_CaptureX1Flag
PWM capture X1 flag

enumerator kPWM_CaptureB0Flag
PWM capture B0 flag

enumerator kPWM_CaptureB1Flag
PWM capture B1 flag

enumerator kPWM_CaptureA0Flag
PWM capture A0 flag

enumerator kPWM_CaptureA1Flag
PWM capture A1 flag

enumerator kPWM_ReloadFlag
PWM reload flag

enumerator kPWM_ReloadErrorFlag
PWM reload error flag

enumerator kPWM_RegUpdatedFlag
PWM registers updated flag

enumerator kPWM_Fault0Flag
PWM fault 0 flag

enumerator kPWM_Fault1Flag
PWM fault 1 flag

enumerator kPWM_Fault2Flag
PWM fault 2 flag

enumerator kPWM_Fault3Flag
PWM fault 3 flag

enum __pwm_dma_enable
List of PWM DMA options.

Values:

enumerator kPWM_CaptureX0DMAEnable
PWM capture X0 DMA

enumerator kPWM_CaptureX1DMAEnable
PWM capture X1 DMA

enumerator kPWM_CaptureB0DMAEnable
PWM capture B0 DMA

enumerator kPWM_CaptureB1DMAEnable
PWM capture B1 DMA

enumerator kPWM_CaptureA0DMAEnable
PWM capture A0 DMA

enumerator kPWM_CaptureA1DMAEnable
PWM capture A1 DMA

enum __pwm_dma_source_select
List of PWM capture DMA enable source select.

Values:

enumerator kPWM_DMARequestDisable
Read DMA requests disabled

enumerator kPWM_DMAWatermarksEnable
Exceeding a FIFO watermark sets the DMA read request

enumerator kPWM_DMALocalSync
A local sync (VAL1 matches counter) sets the read DMA request

enumerator kPWM_DMALocalReload
A local reload (STS[RF] being set) sets the read DMA request

enum __pwm_watermark_control
PWM FIFO Watermark AND Control.

Values:

enumerator kPWM_FIFOWatermarksOR
Selected FIFO watermarks are OR'ed together

enumerator kPWM_FIFOWatermarksAND
Selected FIFO watermarks are AND'ed together

enum __pwm_mode
PWM operation mode.

Values:

enumerator kPWM_SignedCenterAligned
Signed center-aligned

enumerator kPWM_CenterAligned
Unsigned cente-aligned

enumerator kPWM_SignedEdgeAligned
Signed edge-aligned

enumerator kPWM_EdgeAligned
Unsigned edge-aligned

enum __pwm_level_select
PWM output pulse mode, high-true or low-true.

Values:

enumerator kPWM_HighTrue
High level represents "on" or "active" state

enumerator kPWM_LowTrue
Low level represents "on" or "active" state

enum _pwm_fault_state

PWM output fault status.

Values:

enumerator kPWM_PwmFaultState0

Output is forced to logic 0 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState1

Output is forced to logic 1 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState2

Output is tristated.

enumerator kPWM_PwmFaultState3

Output is tristated.

enum _pwm_reload_source_select

PWM reload source select.

Values:

enumerator kPWM_LocalReload

The local reload signal is used to reload registers

enumerator kPWM_MasterReload

The master reload signal (from submodule 0) is used to reload

enum _pwm_fault_clear

PWM fault clearing options.

Values:

enumerator kPWM_Automatic

Automatic fault clearing

enumerator kPWM_ManualNormal

Manual fault clearing with no fault safety mode

enumerator kPWM_ManualSafety

Manual fault clearing with fault safety mode

enum _pwm_module_control

Options for submodule master control operation.

Values:

enumerator kPWM_Control_Module_0

Control submodule 0's start/stop,buffer reload operation

enumerator kPWM_Control_Module_1

Control submodule 1's start/stop,buffer reload operation

enumerator kPWM_Control_Module_2

Control submodule 2's start/stop,buffer reload operation

enumerator kPWM_Control_Module_3

Control submodule 3's start/stop,buffer reload operation

typedef enum *_pwm_submodule* pwm_submodule_t

List of PWM submodules.

typedef enum *_pwm_channels* pwm_channels_t

List of PWM channels in each module.

typedef enum *_pwm_value_register* pwm_value_register_t
List of PWM value registers.

typedef enum *_pwm_clock_source* pwm_clock_source_t
PWM clock source selection.

typedef enum *_pwm_clock_prescale* pwm_clock_prescale_t
PWM prescaler factor selection for clock source.

typedef enum *_pwm_force_output_trigger* pwm_force_output_trigger_t
Options that can trigger a PWM FORCE_OUT.

typedef enum *_pwm_output_state* pwm_output_state_t
PWM channel output status.

typedef enum *_pwm_init_source* pwm_init_source_t
PWM counter initialization options.

typedef enum *_pwm_load_frequency* pwm_load_frequency_t
PWM load frequency selection.

typedef enum *_pwm_fault_input* pwm_fault_input_t
List of PWM fault selections.

typedef enum *_pwm_fault_disable* pwm_fault_disable_t
List of PWM fault disable mapping selections.

typedef enum *_pwm_fault_channels* pwm_fault_channels_t
List of PWM fault channels.

typedef enum *_pwm_input_capture_edge* pwm_input_capture_edge_t
PWM capture edge select.

typedef enum *_pwm_force_signal* pwm_force_signal_t
PWM output options when a FORCE_OUT signal is asserted.

typedef enum *_pwm_chnl_pair_operation* pwm_chnl_pair_operation_t
Options available for the PWM A & B pair operation.

typedef enum *_pwm_register_reload* pwm_register_reload_t
Options available on how to load the buffered-registers with new values.

typedef enum *_pwm_fault_recovery_mode* pwm_fault_recovery_mode_t
Options available on how to re-enable the PWM output when recovering from a fault.

typedef enum *_pwm_interrupt_enable* pwm_interrupt_enable_t
List of PWM interrupt options.

typedef enum *_pwm_status_flags* pwm_status_flags_t
List of PWM status flags.

typedef enum *_pwm_dma_enable* pwm_dma_enable_t
List of PWM DMA options.

typedef enum *_pwm_dma_source_select* pwm_dma_source_select_t
List of PWM capture DMA enable source select.

typedef enum *_pwm_watermark_control* pwm_watermark_control_t
PWM FIFO Watermark AND Control.

typedef enum *_pwm_mode* pwm_mode_t
PWM operation mode.

typedef enum *_pwm_level_select* pwm_level_select_t

PWM output pulse mode, high-true or low-true.

typedef enum *_pwm_fault_state* pwm_fault_state_t

PWM output fault status.

typedef enum *_pwm_reload_source_select* pwm_reload_source_select_t

PWM reload source select.

typedef enum *_pwm_fault_clear* pwm_fault_clear_t

PWM fault clearing options.

typedef enum *_pwm_module_control* pwm_module_control_t

Options for submodule master control operation.

typedef struct *_pwm_signal_param* pwm_signal_param_t

Structure for the user to define the PWM signal characteristics.

typedef struct *_pwm_config* pwm_config_t

PWM config structure.

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

The config struct can be made const so it resides in flash

typedef struct *_pwm_fault_input_filter_param* pwm_fault_input_filter_param_t

Structure for the user to configure the fault input filter.

typedef struct *_pwm_fault_param* pwm_fault_param_t

Structure is used to hold the parameters to configure a PWM fault.

typedef struct *_pwm_input_capture_param* pwm_input_capture_param_t

Structure is used to hold parameters to configure the capture capability of a signal pin.

void PWM_SetupInputCapture(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, const *pwm_input_capture_param_t* *inputCaptureParams)

Sets up the PWM input capture.

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel in the submodule to setup
- inputCaptureParams – Parameters passed in to set up the input pin

void PWM_SetupFaultInputFilter(PWM_Type *base, const *pwm_fault_input_filter_param_t* *faultInputFilterParams)

Sets up the PWM fault channel 0 input filter.

Parameters

- base – PWM peripheral base address
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

void PWM_SetupFaultInputFilterExt(PWM_Type *base, *pwm_fault_channels_t* faultChannel, const *pwm_fault_input_filter_param_t* *faultInputFilterParams)

Sets up the PWM fault input filter.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

void PWM_SetupFaults(PWM_Type *base, *pwm_fault_input_t* faultNum, const *pwm_fault_param_t* *faultParams)

Sets up the PWM fault channel 0 protection.

Parameters

- base – PWM peripheral base address
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

void PWM_SetupFaultsExt(PWM_Type *base, *pwm_fault_channels_t* faultChannel, *pwm_fault_input_t* faultNum, const *pwm_fault_param_t* *faultParams)

Sets up the PWM fault protection.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

void PWM_FaultDefaultConfig(*pwm_fault_param_t* *config)

Fill in the PWM fault config struct with the default settings.

The default values are:

```
config->faultClearingMode = kPWM_Automatic;
config->faultLevel = false;
config->enableCombinationalPath = true;
config->recoverMode = kPWM_NoRecovery;
```

Parameters

- config – Pointer to user's PWM fault config structure.

void PWM_SetupForceSignal(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_force_signal_t* mode)

Selects the signal to output on a PWM pin when a FORCE_OUT signal is asserted.

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure

- mode – Signal to output when a FORCE_OUT is triggered

static inline void PWM_SetVALxValue(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_value_register_t* valueRegister, uint16_t value)

Set the PWM VALx registers.

This function allows the user to write value into VAL registers directly. And it will destroying the PWM clock period set by the PWM_SetupPwm()/PWM_SetupPwmPhaseShift() functions. Due to VALx registers are bufferd, the new value will not active uless call PWM_SetPwmLdok() and the reload point is reached.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegister – VALx register that will be writen new value
- value – Value that will been write into VALx register

static inline uint16_t PWM_GetVALxValue(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_value_register_t* valueRegister)

Get the PWM VALx registers.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegister – VALx register that will be read value

Returns

The VALx register value

static inline void PWM_OutputTriggerEnable(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_value_register_t* valueRegister, bool activate)

Enables or disables the PWM output trigger.

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegister – Value register that will activate the trigger
- activate – true: Enable the trigger; false: Disable the trigger

static inline void PWM_ActivateOutputTrigger(PWM_Type *base, *pwm_submodule_t* subModule, uint16_t valueRegisterMask)

Enables the PWM output trigger.

This function allows the user to enable one or more (VAL0-5) PWM trigger.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegisterMask – Value register mask that will activate one or more (VAL0-5) trigger enumeration _pwm_value_register_mask

static inline void PWM_DeactivateOutputTrigger(PWM_Type *base, *pwm_submodule_t* subModule, uint16_t valueRegisterMask)

Disables the PWM output trigger.

This function allows the user to disables one or more (VAL0-5) PWM trigger.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegisterMask – Value register mask that will Deactivate one or more (VAL0-5) trigger enumeration _pwm_value_register_mask

static inline void PWM_SetupSwCtrlOut(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, bool value)

Sets the software control output for a pin to high or low.

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- value – true: Supply a logic 1, false: Supply a logic 0.

static inline void PWM_SetPwmLdok(PWM_Type *base, uint8_t subModulesToUpdate, bool value)

Sets or clears the PWM LDOK bit on a single or multiple submodules.

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if kPWM_ReloadImmediate option was choosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

Parameters

- base – PWM peripheral base address
- subModulesToUpdate – PWM submodules to update with buffered values. This is a logical OR of members of the enumeration pwm_module_control_t
- value – true: Set LDOK bit for the submodule list; false: Clear LDOK bit

static inline void PWM_SetPwmFaultState(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_fault_state_t* faultState)

Set PWM output fault status.

These bits determine the fault state for the PWM_A output in fault conditions and STOP mode. It may also define the output state in WAIT and DEBUG modes depending on the settings of CTRL2[WAITEN] and CTRL2[DBGEN]. This function can update PWM output fault status.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure

- faultState – PWM output fault status

static inline void PWM_SetupFaultDisableMap(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_fault_channels_t* pwm_fault_channels, uint16_t value)

Set PWM fault disable mapping.

Each of the four bits of this read/write field is one-to-one associated with the four FAULTx inputs of fault channel 0/1. The PWM output will be turned off if there is a logic 1 on an FAULTx input and a 1 in the corresponding bit of this field. A reset sets all bits in this field.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- pwm_fault_channels – PWM fault channel to configure
- value – Fault disable mapping mask value enumeration pwm_fault_disable_t

static inline void PWM_OutputEnable(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule)

Set PWM output enable.

This feature allows the user to enable the PWM Output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure

static inline void PWM_OutputDisable(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule)

Set PWM output disable.

This feature allows the user to disable the PWM output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure

uint8_t PWM_GetPwmChannelState(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel)

Get the dutycycle value.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure

Returns
Current channel dutycycle value.

status_t PWM_SetOutputToIdle(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, bool idleStatus)

Set PWM output in idle status (high or low).

Note: This API should call after PWM_SetupPwm() APIs, and PWMX submodule is not supported.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure
- idleStatus – True: PWM output is high in idle status; false: PWM output is low in idle status.

Returns
kStatus_Fail if there was error setting up the signal; kStatus_Success if set output idle success

void PWM_SetClockMode(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_clock_prescale_t* prescaler)

Set the pwm submodule prescaler.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- prescaler – Set prescaler value

void PWM_SetPwmForceOutputToZero(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, bool forcetozero)

This function enables-disables the forcing of the output of a given eFlexPwm channel to logic 0.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure
- forcetozero – True: Enable the pwm force output to zero; False: Disable the pwm output resumes normal function.

void PWM_SetChannelOutput(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_output_state_t* outputstate)

This function set the output state of the PWM pin as requested for the current cycle.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- outputstate – Set pwm output state, see pwm_output_state_t.

status_t PWM_SetPhaseDelay(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, uint16_t delayCycles)

This function set the phase delay from the master sync signal of submodule 0.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- delayCycles – Number of cycles delayed from submodule 0.

Returns

kStatus_Fail if the number of delay cycles is set larger than the period defined in submodule 0; kStatus_Success if set phase delay success

static inline void PWM_SetFilterSampleCount(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, uint8_t filterSampleCount)

This function set the number of consecutive samples that must agree prior to the input filter.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSampleCount – Number of consecutive samples.

static inline void PWM_SetFilterSamplePeriod(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, uint8_t filterSamplePeriod)

This function set the sampling period of the fault pin input filter.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSamplePeriod – Sampling period of input filter.

PWM_SUBMODULE_SWCONTROL_WIDTH

Number of bits per submodule for software output control

PWM_SUBMODULE_CHANNEL

Submodule channels include PWMA, PWMB, PWMX.

struct __pwm_signal_param

#include <fsl_pwm.h> Structure for the user to define the PWM signal characteristics.

Public Members

pwm_channels_t pwmChannel

PWM channel being configured; PWM A or PWM B

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

pwm_level_select_t level
PWM output active level select

uint16_t deadtimeValue
The deadtime value; only used if channel pair is operating in complementary mode

pwm_fault_state_t faultState
PWM output fault status

bool pwmchannelenable
Enable PWM output

struct __pwm__config
#include <fsl_pwm.h> PWM config structure.

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

The config struct can be made const so it resides in flash

Public Members

bool enableDebugMode
true: PWM continues to run in debug mode; false: PWM is paused in debug mode

pwm_init_source_t initializationControl
Option to initialize the counter

pwm_clock_source_t clockSource
Clock source for the counter

pwm_clock_prescale_t prescale
Pre-scaler to divide down the clock

pwm_chnl_pair_operation_t pairOperation
Channel pair in indepedent or complementary mode

pwm_register_reload_t reloadLogic
PWM Reload logic setup

pwm_reload_source_select_t reloadSelect
Reload source select

pwm_load_frequency_t reloadFrequency
Specifies when to reload, used when user's choice is not immediate reload

pwm_force_output_trigger_t forceTrigger
Specify which signal will trigger a FORCE_OUT

struct __pwm__fault__input__filter__param
#include <fsl_pwm.h> Structure for the user to configure the fault input filter.

Public Members

uint8_t faultFilterCount
Fault filter count

uint8_t faultFilterPeriod
Fault filter period;value of 0 will bypass the filter

bool faultGlitchStretch

Fault Glitch Stretch Enable: A logic 1 means that input fault signals will be stretched to at least 2 IPBus clock cycles

struct __pwm__fault__param

#include <fsl_pwm.h> Structure is used to hold the parameters to configure a PWM fault.

Public Members

pwm_fault_clear_t faultClearingMode

Fault clearing mode to use

bool faultLevel

true: Logic 1 indicates fault; false: Logic 0 indicates fault

bool enableCombinationalPath

true: Combinational Path from fault input is enabled; false: No combination path is available

pwm_fault_recovery_mode_t recoverMode

Specify when to re-enable the PWM output

struct __pwm__input__capture__param

#include <fsl_pwm.h> Structure is used to hold parameters to configure the capture capability of a signal pin.

Public Members

bool captureInputSel

true: Use the edge counter signal as source false: Use the raw input signal from the pin as source

uint8_t edgeCompareValue

Compare value, used only if edge counter is used as source

pwm_input_capture_edge_t edge0

Specify which edge causes a capture for input circuitry 0

pwm_input_capture_edge_t edge1

Specify which edge causes a capture for input circuitry 1

bool enableOneShotCapture

true: Use one-shot capture mode; false: Use free-running capture mode

uint8_t fifoWatermark

Watermark level for capture FIFO. The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

2.104 QTMR: Quad Timer Driver

void QTMR__Init(TMR_Type *base, *qtmr_channel_selection_t* channel, const *qtmr_config_t* *config)

Ungates the Quad Timer clock and configures the peripheral for basic operation.

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

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- config – Pointer to user's Quad Timer config structure

void QTMR_Deinit(TMR_Type *base, *qtmr_channel_selection_t* channel)

Stops the counter and gates the Quad Timer clock.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

void QTMR_GetDefaultConfig(*qtmr_config_t* *config)

Fill in the Quad Timer config struct with the default settings.

The default values are:

```
config->debugMode = kQTMR_RunNormalInDebug;
config->enableExternalForce = false;
config->enableMasterMode = false;
config->faultFilterCount = 0;
config->faultFilterPeriod = 0;
config->primarySource = kQTMR_ClockDivide_2;
config->secondarySource = kQTMR_Counter0InputPin;
```

Parameters

- config – Pointer to user's Quad Timer config structure.

void QTMR_EnableInterrupts(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Enables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration qtmr_interrupt_enable_t

void QTMR_DisableInterrupts(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Disables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base addres
- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration qtmr_interrupt_enable_t

uint32_t QTMR_GetEnabledInterrupts(TMR_Type *base, *qtmr_channel_selection_t* channel)

Gets the enabled Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns
The enabled interrupts. This is the logical OR of members of the enumeration qtmr_interrupt_enable_t

uint32_t QTMR_GetStatus(TMR_Type *base, *qtmr_channel_selection_t* channel)

Gets the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns
The status flags. This is the logical OR of members of the enumeration qtmr_status_flags_t

void QTMR_ClearStatusFlags(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Clears the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration qtmr_status_flags_t

void QTMR_SetTimerPeriod(TMR_Type *base, *qtmr_channel_selection_t* channel, uint16_t ticks)

Sets the timer period in ticks.

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note:

a. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction

b. User can call the utility macros provided in fsl_common.h to convert to ticks

c. This function supports cases, providing only primary source clock without secondary source clock.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- ticks – Timer period in units of ticks

void QTMR_SetCompareValue(TMR_Type *base, *qtmr_channel_selection_t* channel, uint16_t ticks)

Set compare value.

This function sets the value used for comparison with the counter value.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- ticks – Timer period in units of ticks.

static inline void QTMR_SetLoadValue(TMR_Type *base, *qtmr_channel_selection_t* channel, uint16_t value)

Set load value.

This function sets the value used to initialize the counter after a counter comparison.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- value – Load register initialization value.

static inline uint16_t QTMR_GetCurrentTimerCount(TMR_Type *base, *qtmr_channel_selection_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 – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current counter value in ticks

static inline void QTMR_StartTimer(TMR_Type *base, *qtmr_channel_selection_t* channel, *qtmr_counting_mode_t* clockSource)

Starts the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- clockSource – Quad Timer clock source

static inline void QTMR_StopTimer(TMR_Type *base, *qtmr_channel_selection_t* channel)

Stops the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

void QTMR_EnableDma(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Enable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration qtmr_dma_enable_t

void QTMR_DisableDma(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Disable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration qtmr_dma_enable_t

void QTMR_SetPwmOutputToIdle(TMR_Type *base, *qtmr_channel_selection_t* channel, bool idleStatus)

Set PWM output in idle status (high or low).

Note: When the PWM is set again, the counting needs to be restarted.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- idleStatus – True: PWM output is high in idle status; false: PWM output is low in idle status.

static inline *qtmr_pwm_out_state_t* QTMR_GetPwmOutputStatus(TMR_Type *base, *qtmr_channel_selection_t* channel)

Get the channel output status.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel output status.

uint8_t QTMR_GetPwmChannelStatus(TMR_Type *base, *qtmr_channel_selection_t* channel)

Get the PWM channel dutycycle value.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel dutycycle value.

void QTMR_SetPwmClockMode(TMR_Type *base, *qtmr_channel_selection_t* channel, *qtmr_primary_count_source_t* prescaler)

This function set the value of the prescaler on QTimer channels.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- prescaler – Set prescaler value

FSL_QTMR_DRIVER_VERSION

Version

enum _qtmr_primary_count_source

Quad Timer primary clock source selection.

Values:

enumerator kQTMR_ClockCounter0InputPin
Use counter 0 input pin

enumerator kQTMR_ClockCounter1InputPin
Use counter 1 input pin

enumerator kQTMR_ClockCounter2InputPin
Use counter 2 input pin

enumerator kQTMR_ClockCounter3InputPin
Use counter 3 input pin

enumerator kQTMR_ClockCounter0Output
Use counter 0 output

enumerator kQTMR_ClockCounter1Output
Use counter 1 output

enumerator kQTMR_ClockCounter2Output
Use counter 2 output

enumerator kQTMR_ClockCounter3Output
Use counter 3 output

enumerator kQTMR_ClockDivide_1
IP bus clock divide by 1 prescaler

enumerator kQTMR_ClockDivide_2
IP bus clock divide by 2 prescaler

enumerator kQTMR_ClockDivide_4
IP bus clock divide by 4 prescaler

enumerator kQTMR_ClockDivide_8
IP bus clock divide by 8 prescaler

enumerator kQTMR_ClockDivide_16
IP bus clock divide by 16 prescaler

enumerator kQTMR_ClockDivide_32
IP bus clock divide by 32 prescaler

enumerator kQTMR_ClockDivide_64
IP bus clock divide by 64 prescaler

enumerator kQTMR_ClockDivide_128
IP bus clock divide by 128 prescaler

enum _qtmr_input_source

Quad Timer input sources selection.

Values:

enumerator kQTMR_Counter0InputPin
Use counter 0 input pin

enumerator kQTMR_Counter1InputPin
Use counter 1 input pin

enumerator kQTMR_Counter2InputPin
Use counter 2 input pin

enumerator kQTMR_Counter3InputPin
Use counter 3 input pin

enum _qtmr_counting_mode
Quad Timer counting mode selection.

Values:

enumerator kQTMR_NoOperation
No operation

enumerator kQTMR_PriSrcRiseEdge
Count rising edges of primary source

enumerator kQTMR_PriSrcRiseAndFallEdge
Count rising and falling edges of primary source

enumerator kQTMR_PriSrcRiseEdgeSecInpHigh
Count rise edges of pri SRC while sec inp high active

enumerator kQTMR_QuadCountMode
Quadrature count mode, uses pri and sec sources

enumerator kQTMR_PriSrcRiseEdgeSecDir
Count rising edges of pri SRC; sec SRC specifies dir

enumerator kQTMR_SecSrcTrigPriCnt
Edge of sec SRC trigger primary count until compare

enumerator kQTMR_CascadeCount
Cascaded count mode (up/down)

enum _qtmr_pwm_out_state
Quad Timer PWM output state.

Values:

enumerator kQTMR_PwmLow
The output state of PWM channel is low

enumerator kQTMR_PwmHigh
The output state of PWM channel is low

enum _qtmr_output_mode
Quad Timer output mode selection.

Values:

enumerator kQTMR_AssertWhenCountActive
Assert OFLAG while counter is active

enumerator kQTMR_ClearOnCompare
Clear OFLAG on successful compare

enumerator kQTMR_SetOnCompare
Set OFLAG on successful compare

enumerator kQTMR_ToggleOnCompare
Toggle OFLAG on successful compare

enumerator kQTMR_ToggleOnAltCompareReg
Toggle OFLAG using alternating compare registers

enumerator kQTMR_SetOnCompareClearOnSecSrcInp
Set OFLAG on compare, clear on sec SRC input edge

enumerator kQTMR_SetOnCompareClearOnCountRoll
Set OFLAG on compare, clear on counter rollover

enumerator kQTMR_EnableGateClock
Enable gated clock output while count is active

enum _qtmr_input_capture_edge
Quad Timer input capture edge mode, rising edge, or falling edge.

Values:

enumerator kQTMR_NoCapture
Capture is disabled

enumerator kQTMR_RisingEdge
Capture on rising edge (IPS=0) or falling edge (IPS=1)

enumerator kQTMR_FallingEdge
Capture on falling edge (IPS=0) or rising edge (IPS=1)

enumerator kQTMR_RisingAndFallingEdge
Capture on both edges

enum _qtmr_preload_control
Quad Timer input capture edge mode, rising edge, or falling edge.

Values:

enumerator kQTMR_NoPreload
Never preload

enumerator kQTMR_LoadOnComp1
Load upon successful compare with value in COMP1

enumerator kQTMR_LoadOnComp2
Load upon successful compare with value in COMP2

enum _qtmr_debug_action
List of Quad Timer run options when in Debug mode.

Values:

enumerator kQTMR_RunNormalInDebug
Continue with normal operation

enumerator kQTMR_HaltCounter
Halt counter

enumerator kQTMR_ForceOutToZero
Force output to logic 0

enumerator kQTMR_HaltCountForceOutZero
Halt counter and force output to logic 0

enum _qtmr_interrupt_enable
List of Quad Timer interrupts.

Values:

enumerator kQTMR_CompareInterruptEnable
Compare interrupt.

enumerator kQTMR_Compare1InterruptEnable
Compare 1 interrupt.

enumerator kQTMR_Compare2InterruptEnable
Compare 2 interrupt.

enumerator kQTMR_OverflowInterruptEnable
Timer overflow interrupt.

enumerator kQTMR_EdgeInterruptEnable
Input edge interrupt.

enum _qtmr_status_flags
List of Quad Timer flags.

Values:

enumerator kQTMR_CompareFlag
Compare flag

enumerator kQTMR_Compare1Flag
Compare 1 flag

enumerator kQTMR_Compare2Flag
Compare 2 flag

enumerator kQTMR_OverflowFlag
Timer overflow flag

enumerator kQTMR_EdgeFlag
Input edge flag

enum _qtmr_channel_selection
List of channel selection.

Values:

enumerator kQTMR_Channel_0
TMR Channel 0

enumerator kQTMR_Channel_1
TMR Channel 1

enumerator kQTMR_Channel_2
TMR Channel 2

enumerator kQTMR_Channel_3
TMR Channel 3

enum _qtmr_dma_enable
List of Quad Timer DMA enable.

Values:

enumerator kQTMR_InputEdgeFlagDmaEnable
Input Edge Flag DMA Enable.

enumerator kQTMR_ComparatorPreload1DmaEnable
Comparator Preload Register 1 DMA Enable.

enumerator kQTMR_ComparatorPreload2DmaEnable
Comparator Preload Register 2 DMA Enable.

typedef uint16_t qtmrRegType

typedef enum *_qtmr_primary_count_source* qtmr_primary_count_source_t
Quad Timer primary clock source selection.

typedef enum *_qtmr_input_source* qtmr_input_source_t
Quad Timer input sources selection.

typedef enum *_qtmr_counting_mode* qtmr_counting_mode_t
Quad Timer counting mode selection.

typedef enum *_qtmr_pwm_out_state* qtmr_pwm_out_state_t
Quad Timer PWM output state.

typedef enum *_qtmr_output_mode* qtmr_output_mode_t
Quad Timer output mode selection.

typedef enum *_qtmr_input_capture_edge* qtmr_input_capture_edge_t
Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum *_qtmr_preload_control* qtmr_preload_control_t
Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum *_qtmr_debug_action* qtmr_debug_action_t
List of Quad Timer run options when in Debug mode.

typedef enum *_qtmr_interrupt_enable* qtmr_interrupt_enable_t
List of Quad Timer interrupts.

typedef enum *_qtmr_status_flags* qtmr_status_flags_t
List of Quad Timer flags.

typedef enum *_qtmr_channel_selection* qtmr_channel_selection_t
List of channel selection.

typedef enum *_qtmr_dma_enable* qtmr_dma_enable_t
List of Quad Timer DMA enable.

typedef struct *_qtmr_config* qtmr_config_t
Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the QTMR_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 QTMR_SetupPwm(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity, uint32_t srcClock_Hz)
Sets up Quad timer module for PWM signal output.

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

- pwmFreqHz – PWM signal frequency in Hz
- dutyCyclePercent – PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)
- outputPolarity – true: invert polarity of the output signal, false: no inversion
- srcClock_Hz – Main counter clock in Hz.

Returns

Returns an error if there was error setting up the signal.

void QTMR_SetupInputCapture(TMR_Type *base, *qtmr_channel_selection_t* channel, *qtmr_input_source_t* capturePin, bool inputPolarity, bool reloadOnCapture, *qtmr_input_capture_edge_t* captureMode)

Allows the user to count the source clock cycles until a capture event arrives.

The count is stored in the capture register.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- capturePin – Pin through which we receive the input signal to trigger the capture
- inputPolarity – true: invert polarity of the input signal, false: no inversion
- reloadOnCapture – true: reload the counter when an input capture occurs, false: no reload
- captureMode – Specifies which edge of the input signal triggers a capture

TMR_CSCTRL_OFLAG_MASK

TMR_CSCTRL_OFLAG_SHIFT

struct _qtmr_config

#include <fsl_qtmr.h> Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the QTMR_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

qtmr_primary_count_source_t primarySource

Specify the primary count source

qtmr_input_source_t secondarySource

Specify the secondary count source

bool enableMasterMode

true: Broadcast compare function output to other counters; false no broadcast

bool enableExternalForce

true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter

uint8_t faultFilterCount

Fault filter count

uint8_t faultFilterPeriod
Fault filter period;value of 0 will bypass the filter

qtmr_debug_action_t debugMode
Operation in Debug mode

2.105 RGPIO: Rapid General-Purpose Input/Output Driver

FSL_RGPIO_DRIVER_VERSION
RGPIO driver version 2.1.0.

enum _rgpio_pin_direction
RGPIO direction definition.

Values:

enumerator kRGPIO_DigitalInput
Set current pin as digital input

enumerator kRGPIO_DigitalOutput
Set current pin as digital output

enum _rgpio_checker_attribute
RGPIO checker attribute.

Values:

enumerator kRGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW
User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW
User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW
User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kRGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kRGPIO_IgnoreAttributeCheck
Ignores the attribute check

enum _rgpio_interrupt_sel
Configures the interrupt generation condition.

Values:

enumerator kRGPIO_InterruptOutput0
Interrupt/DMA request/trigger output 0.

enumerator kRGPIO_InterruptOutput1
Interrupt/DMA request/trigger output 1.

enumerator kRGPIO_InterruptOutput2
Interrupt/DMA request/trigger output 2.

enumerator kRGPIO_InterruptOutput3
Interrupt/DMA request/trigger output 3.

enum _rgpio_interrupt_config
Configures the interrupt generation condition.

Values:

enumerator kRGPIO_InterruptOrDMADisabled
Interrupt/DMA request is disabled.

enumerator kRGPIO_DMARisingEdge
DMA request on rising edge.

enumerator kRGPIO_DMAFallingEdge
DMA request on falling edge.

enumerator kRGPIO_DMAEitherEdge
DMA request on either edge.

enumerator kRGPIO_FlagRisingEdge
Flag sets on rising edge.

enumerator kRGPIO_FlagFallingEdge
Flag sets on falling edge.

enumerator kRGPIO_FlagEitherEdge
Flag sets on either edge.

enumerator kRGPIO_InterruptLogicZero
Interrupt when logic zero.

enumerator kRGPIO_InterruptRisingEdge
Interrupt on rising edge.

enumerator kRGPIO_InterruptFallingEdge
Interrupt on falling edge.

enumerator kRGPIO_InterruptEitherEdge
Interrupt on either edge.

enumerator kRGPIO_InterruptLogicOne
Interrupt when logic one.

enumerator kRGPIO_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kRGPIO_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

typedef enum *_rgpio_pin_direction* rgpio_pin_direction_t
RGPIO direction definition.

typedef enum *_rgpio_checker_attribute* rgpio_checker_attribute_t
RGPIO checker attribute.

typedef enum *_rgpio_interrupt_sel* rgpio_interrupt_sel_t

Configures the interrupt generation condition.

typedef enum *_rgpio_interrupt_config* rgpio_interrupt_config_t

Configures the interrupt generation condition.

typedef struct *_rgpio_pin_config* rgpio_pin_config_t

The RGPIO 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().

struct _rgpio_pin_config

#include <fsl_rgpio.h> The RGPIO 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

rgpio_pin_direction_t pinDirection

RGPIO direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

2.106 RGPIO Driver

void RGPIO_PinInit(RGPIO_Type *base, uint32_t pin, const *rgpio_pin_config_t* *config)

Initializes a RGPIO pin used by the board.

To initialize the RGPIO, define a pin configuration, as either input or output, in the user file. Then, call the RGPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
 Define a digital input pin configuration,
rgpio_pin_config_t config =
{
  kRGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
rgpio_pin_config_t config =
{
  kRGPIO_DigitalOutput,
  0,
}
```

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO port pin number
- config – RGPIO pin configuration pointer

uint32_t RGPIO_GetInstance(RGPIO_Type *base)

Gets the RGPIO instance according to the RGPIO base.

Parameters

- base – RGPIO peripheral base pointer(PTA, PTB, PTC, etc.)

Return values

RGPIO – instance

static inline void RGPIO_PinWrite(RGPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple RGPIO pins to the logic 1 or 0.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO pin number
- output – RGPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void RGPIO_WritePinOutput(RGPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple RGPIO pins to the logic 1 or 0.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PinWrite.

static inline void RGPIO_PortSet(RGPIO_Type *base, uint32_t mask)

Sets the output level of the multiple RGPIO pins to the logic 1.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

static inline void RGPIO_SetPinsOutput(RGPIO_Type *base, uint32_t mask)

Sets the output level of the multiple RGPIO pins to the logic 1.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PortSet.

static inline void RGPIO_PortClear(RGPIO_Type *base, uint32_t mask)

Sets the output level of the multiple RGPIO pins to the logic 0.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

static inline void RGPIO_ClearPinsOutput(RGPIO_Type *base, uint32_t mask)

Sets the output level of the multiple RGPIO pins to the logic 0.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PortClear.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

static inline void RGPIO_PortToggle(RGPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple RGPIO pins.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

static inline void RGPIO_TogglePinsOutput(RGPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple RGPIO pins.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PortToggle.

static inline uint32_t RGPIO_PinRead(RGPIO_Type *base, uint32_t pin)

Reads the current input value of the RGPIO port.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO pin number

Return values

RGPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32_t RGPIO_ReadPinInput(RGPIO_Type *base, uint32_t pin)

Reads the current input value of the RGPIO port.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PinRead.

static inline void RGPIO_EnablePortInput(RGPIO_Type *base, uint32_t mask, bool enable)

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number mask
- enable – RGPIO digital input enable/disable flag.

void RGPIO_CheckAttributeBytes(RGPIO_Type *base, *rgpio_checker_attribute_t* attribute)

The RGPIO module supports a device-specific number of data ports, organized as 32-bit words. Each 32-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the RGPIO programming model. The attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

static inline void RGPIO_SetPinInterruptConfig(RGPIO_Type *base, uint32_t pin, *rgpio_interrupt_sel_t* sel, *rgpio_interrupt_config_t* config)

Configures the gpio pin interrupt/DMA request.

Parameters

- base – RGPIO peripheral base pointer.
- pin – RGPIO pin number.
- sel – RGPIO pin interrupt selection(0-3).
- config – RGPIO pin interrupt configuration.

static inline void _SetMultipleInterruptPinsConfig(RGPIO_Type *base, uint32_t mask, *rgpio_interrupt_sel_t* sel, *rgpio_interrupt_config_t* config)

Sets the gpio interrupt configuration in ICR register for multiple pins.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro.
- sel – RGPIO pin interrupt selection(0-3).
- config – RGPIO pin interrupt configuration.

static inline uint32_t RGPIO_GetPinsInterruptFlags(RGPIO_Type *base, *rgpio_interrupt_sel_t* sel)

Reads the whole gpio status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

- base – RGPIO peripheral base pointer.
- sel – RGPIO pin interrupt selection(0-3).

Returns

Current gpio interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

static inline void RGPIO_ClearPinsInterruptFlags(RGPIO_Type *base, *rgpio_interrupt_sel_t* sel, uint32_t mask)

Clears the multiple pin interrupt status flag.

Parameters

- base – RGPIO peripheral base pointer.
- sel – RGPIO pin interrupt selection(0-3).
- mask – RGPIO pin number macro.

2.107 RTWDOG: 32-bit Watchdog Timer

void RTWDOG_GetDefaultConfig(*rtwdog_config_t* *config)

Initializes the RTWDOG configuration structure.

This function initializes the RTWDOG configuration structure to default values. The default values are:

```
rtwdogConfig->enableRtwdog = true;
rtwdogConfig->clockSource = kRTWDOG_ClockSource1;
rtwdogConfig->prescaler = kRTWDOG_ClockPrescalerDivide1;
rtwdogConfig->workMode.enableWait = true;
rtwdogConfig->workMode.enableStop = false;
rtwdogConfig->workMode.enableDebug = false;
rtwdogConfig->testMode = kRTWDOG_TestModeDisabled;
rtwdogConfig->enableUpdate = true;
rtwdogConfig->enableInterrupt = false;
rtwdogConfig->enableWindowMode = false;
rtwdogConfig->windowValue = 0U;
rtwdogConfig->timeoutValue = 0xFFFFU;
```

See also:

rtwdog_config_t

Parameters

- config – Pointer to the RTWDOG configuration structure.

void RTWDOG_Init(RTWDOG_Type *base, const *rtwdog_config_t* *config)

Initializes the RTWDOG module.

This function initializes the RTWDOG. To reconfigure the RTWDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
rtwdog_config_t config;
RTWDOG_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
RTWDOG_Init(wdog_base,&config);
```

Parameters

- base – RTWDOG peripheral base address.
- config – The configuration of the RTWDOG.

void RTWDOG_Deinit(RTWDOG_Type *base)

De-initializes the RTWDOG module.

This function shuts down the RTWDOG. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

- base – RTWDOG peripheral base address.

static inline void RTWDOG_Enable(RTWDOG_Type *base)

Enables the RTWDOG module.

This function writes a value into the WDOG_CS register to enable the RTWDOG. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.

static inline void RTWDOG_Disable(RTWDOG_Type *base)

Disables the RTWDOG module.

This function writes a value into the WDOG_CS register to disable the RTWDOG. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address

static inline void RTWDOG_EnableInterrupts(RTWDOG_Type *base, uint32_t mask)

Enables the RTWDOG interrupt.

This function writes a value into the WDOG_CS register to enable the RTWDOG interrupt. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.
- mask – The interrupts to enable. The parameter can be a combination of the following source if defined:
  - kRTWDOG_InterruptEnable

static inline void RTWDOG_DisableInterrupts(RTWDOG_Type *base, uint32_t mask)

Disables the RTWDOG interrupt.

This function writes a value into the WDOG_CS register to disable the RTWDOG interrupt. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.
- mask – The interrupts to disabled. The parameter can be a combination of the following source if defined:
  - kRTWDOG_InterruptEnable

static inline uint32_t RTWDOG_GetStatusFlags(RTWDOG_Type *base)

Gets the RTWDOG all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;
status = RTWDOG_GetStatusFlags(wdog_base) & kRTWDOG_RunningFlag;
```

See also:

_rtwdog_status_flags_t

- true: related status flag has been set.
- false: related status flag is not set.

Parameters

- base – RTWDOG peripheral base address

Returns
State of the status flag: asserted (true) or not-asserted (false).

static inline void RTWDOG_EnableWindowMode(RTWDOG_Type *base, bool enable)

Enables/disables the window mode.

Parameters

- base – RTWDOG peripheral base address.
- enable – Enables(true) or disables(false) the feature.

static inline uint32_t RTWDOG_CountToMesec(RTWDOG_Type *base, uint32_t count, uint32_t clockFreqInHz)

Converts raw count value to millisecond.

Note that if the clock frequency is too high the timeout period can be less than 1 ms. In this case this api will return 0 value.

Parameters

- base – RTWDOG peripheral base address.
- count – Raw count value.
- clockFreqInHz – The frequency of the clock source RTWDOG uses.

Returns
Return converted time. Will return 0 if result is larger than 0xFFFFFFFF.

void RTWDOG_ClearStatusFlags(RTWDOG_Type *base, uint32_t mask)

Clears the RTWDOG flag.

This function clears the RTWDOG status flag.

Example to clear an interrupt flag:

```
RTWDOG_ClearStatusFlags(wdog_base,kRTWDOG_InterruptFlag);
```

Parameters

- base – RTWDOG peripheral base address.
- mask – The status flags to clear. The parameter can be any combination of the following values:
  - kRTWDOG_InterruptFlag

static inline void RTWDOG_SetTimeoutValue(RTWDOG_Type *base, uint16_t timeoutCount)

Sets the RTWDOG timeout value.

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address
- timeoutCount – RTWDOG timeout value, count of RTWDOG clock ticks.

static inline void RTWDOG_SetWindowValue(RTWDOG_Type *base, uint16_t windowValue)

Sets the RTWDOG window value.

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.
- windowValue – RTWDOG window value.

___STATIC_FORCEINLINE void RTWDOG_Unlock (RTWDOG_Type *base)

Unlocks the RTWDOG register written.

This function unlocks the RTWDOG register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

- base – RTWDOG peripheral base address

static inline void RTWDOG_Refresh(RTWDOG_Type *base)

Refreshes the RTWDOG timer.

This function feeds the RTWDOG. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – RTWDOG peripheral base address

static inline uint32_t RTWDOG_GetCounterValue(RTWDOG_Type *base)

Gets the RTWDOG counter value.

This function gets the RTWDOG counter value.

Parameters

- base – RTWDOG peripheral base address.

Returns

Current RTWDOG counter value.

WDOG_FIRST_WORD_OF_UNLOCK

First word of unlock sequence

WDOG_SECOND_WORD_OF_UNLOCK

Second word of unlock sequence

WDOG_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WDOG_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

FSL_RTWDOG_DRIVER_VERSION

RTWDOG driver version.

enum _rtwdog_clock_source

Describes RTWDOG clock source.

Values:

enumerator kRTWDOG_ClockSource0

Clock source 0

enumerator kRTWDOG_ClockSource1

Clock source 1

enumerator kRTWDOG_ClockSource2
Clock source 2

enumerator kRTWDOG_ClockSource3
Clock source 3

enum _rtwdog_clock_prescaler
Describes the selection of the clock prescaler.

Values:

enumerator kRTWDOG_ClockPrescalerDivide1
Divided by 1

enumerator kRTWDOG_ClockPrescalerDivide256
Divided by 256

enum _rtwdog_test_mode
Describes RTWDOG test mode.

Values:

enumerator kRTWDOG_TestModeDisabled
Test Mode disabled

enumerator kRTWDOG_UserModeEnabled
User Mode enabled

enumerator kRTWDOG_LowByteTest
Test Mode enabled, only low byte is used

enumerator kRTWDOG_HighByteTest
Test Mode enabled, only high byte is used

enum _rtwdog_interrupt_enable_t
RTWDOG interrupt configuration structure.

This structure contains the settings for all of the RTWDOG interrupt configurations.

Values:

enumerator kRTWDOG_InterruptEnable
Interrupt is generated before forcing a reset

enum _rtwdog_status_flags_t
RTWDOG status flags.

This structure contains the RTWDOG status flags for use in the RTWDOG functions.

Values:

enumerator kRTWDOG_RunningFlag
Running flag, set when RTWDOG is enabled

enumerator kRTWDOG_InterruptFlag
Interrupt flag, set when interrupt occurs

typedef enum *_rtwdog_clock_source* rtwdog_clock_source_t
Describes RTWDOG clock source.

typedef enum *_rtwdog_clock_prescaler* rtwdog_clock_prescaler_t
Describes the selection of the clock prescaler.

typedef struct *_rtwdog_work_mode* rtwdog_work_mode_t
Defines RTWDOG work mode.

typedef enum *_rtwdog_test_mode* rtwdog_test_mode_t
Describes RTWDOG test mode.

typedef struct *_rtwdog_config* rtwdog_config_t
Describes RTWDOG configuration structure.

struct _rtwdog_work_mode
#include <fsl_rtwdog.h> Defines RTWDOG work mode.

Public Members

bool enableWait
Enables or disables RTWDOG in wait mode

bool enableStop
Enables or disables RTWDOG in stop mode

bool enableDebug
Enables or disables RTWDOG in debug mode

struct _rtwdog_config
#include <fsl_rtwdog.h> Describes RTWDOG configuration structure.

Public Members

bool enableRtwdog
Enables or disables RTWDOG

rtwdog_clock_source_t clockSource
Clock source select

rtwdog_clock_prescaler_t prescaler
Clock prescaler value

rtwdog_work_mode_t workMode
Configures RTWDOG work mode in debug stop and wait mode

rtwdog_test_mode_t testMode
Configures RTWDOG test mode

bool enableUpdate
Update write-once register enable

bool enableInterrupt
Enables or disables RTWDOG interrupt

bool enableWindowMode
Enables or disables RTWDOG window mode

uint16_t windowValue
Window value

uint16_t timeoutValue
Timeout value

2.108 S3MU: Sentinel

FSL_S3MU_DRIVER_VERSION

Defines S3MU driver version 2.0.2.

Change log:

- Version 2.0.2
  - Fix macro BIT redefined warning when compiling with Zephyr.
- Version 2.0.1
  - Update kStatusGroup_SNT to kStatusGroup_ELEMU
- Version 2.0.0
  - initial version

Values:

enumerator kStatus_S3MU_AgumentOutOfRange

S3MU status for out of range access.

enumerator kStatus_S3MU_InvalidArgument

S3MU status for invalid argument check.

enumerator kStatus_S3MU_RequestTimeout

S3MU status for timeout.

enumerator kStatus_S3MU_Busy

S3MU status for reservation by other core.

status_t S3MU_SendMessage(S3MU_Type *mu, void *buf, size_t wordCount)

Send message to MU.

This function writes messgae into MU regsters and send message to EdgeLock Enclave.

Parameters

- base – MU peripheral base address
- buf – buffer to store read data
- wordCount – size of data in words

Returns

Status kStatus_Success if success, kStatus_Fail if fail Possible errors: kStatus_S3MU_InvalidArgument, kStatus_S3MU_AgumentOutOfRange

status_t S3MU_GetResponse(S3MU_Type *mu, void *buf)

Get response from MU.

This function reads response data from EdgeLock Enclave if available.

Note: number of read response word is obtained by header, so user need to prepare buffer with enough space for expected response

Parameters

- base – MU peripheral base address
- buf – buffer to store read data

Returns
Status kStatus_Success if success, kStatus_Fail if fail Possible errors: kStatus_S3MU_InvalidArgument, kStatus_S3MU_AgumentOutOfRange

status_t S3MU_WaitForData(S3MU_Type *mu, uint32_t *buf, size_t wordCount, uint32_t wait)

Wait and Read data from MU.

This function waits limited time (ticks) and tests if data are ready to be read. When data are ready, reads them into buffer, timeout otherwise.

Parameters

- base – MU peripheral base address
- buf – buffer to store read data
- wordCount – size of data in words
- wait – number of iterations to wait

Returns
Status kStatus_Success if success, kStatus_S3MU_RequestTimeout if timeout Possible errors: kStatus_S3MU_InvalidArgument, kStatus_S3MU_AgumentOutOfRange

status_t S3MU_ReadMessage(S3MU_Type *mu, uint32_t *buf, size_t *size, uint8_t read_header)

Read message from MU.

This function reads message date from EdgeLock Enclave if available.

Parameters

- base – MU peripheral base address
- buf – buffer to store read data
- size – lenght of data in words. If read_header equal MU_READ_HEADER, size is read from response header.
- read_header – specifies if size is obtained by response header or provided in parameter.

Returns
Status kStatus_Success if success, kStatus_Fail if fail Possible errors: kStatus_S3MU_InvalidArgument, kStatus_S3MU_AgumentOutOfRange

void S3MU_Init(S3MU_Type *mu)

Init MU.

This function does nothing. MU is initialized after leaving ROM.

Parameters

- base – MU peripheral base address

uint32_t S3MU_ComputeMsgCrc(uint32_t *msg, uint32_t msg_len)

Computes CRC.

This function computes CRC of input message.

Parameters

- msg – pointer to message
- msg_len – size of message in words

Returns
CRC32 checksum value

MU_MSG_HEADER_SIZE

MESSAGING_TAG_COMMAND

MESSAGING_TAG_REPLY

struct mu_hdr_t

#include <fsl_s3mu.h>

union ___unnamed211___

Public Members

uint32_t value

struct *mu_hdr_t* hdr_byte

struct hdr_byte

2.109 SAI: Serial Audio Interface

2.110 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.

- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_SetMasterClockConfig(I2S_Type *base, *sai_master_clock_t* *config)

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

void SAI_TxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.

- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly    = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

void SAI_TxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But

software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable

- kSAI_FIFOWarningInterruptEnable
- kSAI_FIFORequestInterruptEnable
- kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

void SAI_TransferTxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SAI_TransferRxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

void SAI_TransferTxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – tranmitter configurations.

void SAI_TransferRxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer

- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.10

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy
SAI Tx is busy.

enumerator kStatus_SAI_RxBusy
SAI Rx is busy.

enumerator kStatus_SAI_TxError
SAI Tx FIFO error.

enumerator kStatus_SAI_RxError
SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull
SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle
SAI Tx is idle

enumerator kStatus_SAI_RxIdle
SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.

Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.

Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.

Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx
Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx
Synchronous with another SAI receiver

enum _sai_bclk_source
Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk
Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1
Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2
Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3
Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv
Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0
Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1
Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable
FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable
FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing
The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate

Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz

Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz

Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz

Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz

Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz

Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz

Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz

Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz

Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz

Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz

Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz

Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz

Sample rate 384000 Hz

enum _sai_word_width

Audio word width.

Values:

enumerator kSAI_WordWidth8bits

Audio data width 8 bits

enumerator kSAI_WordWidth16bits

Audio data width 16 bits

enumerator kSAI_WordWidth24bits

Audio data width 24 bits

enumerator kSAI_WordWidth32bits

Audio data width 32 bits

enum _sai_data_pin_state

sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine
sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled
sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead
sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite
sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite
sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead
sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite
sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type
sai transceiver type

Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol

Audio bus protocol in SAI

sai_sync_mode_t syncMode

SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource

Bit Clock source

sai_master_slave_t masterSlave

Master or slave

struct _sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz

Sample rate of audio data

uint32_t bitWidth

Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo

Mono or stereo

uint32_t masterClockHz
: Master clock frequency in Hz

uint8_t watermark
: Watermark value

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct __sai__master__clock
: *#include <fsl_sai.h>* master clock configurations

Public Members

bool mclkOutputEnable
: master clock output enable

uint32_t mclkHz
: target mclk frequency

uint32_t mclkSourceClkHz
: mclk source frequency

struct __sai__fifo
: *#include <fsl_sai.h>* sai fifo configurations

Public Members

bool fifoContinueOneError
: fifo continues when error occur

sai_fifo_combine_t fifoCombine
: fifo combine mode

sai_fifo_packing_t fifoPacking
: fifo packing mode

uint8_t fifoWatermark
: fifo watermark

struct __sai__bit__clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkSrcSwap
: bit clock source swap

bool bclkInputDelay
: bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
: bit clock polarity

sai_bclk_source_t bclkSource
: bit Clock source

struct __sai__frame__sync
: *#include <fsl_sai.h>* sai frame sync configurations

Public Members

uint8_t frameSyncWidth
: frame sync width in number of bit clocks

bool frameSyncEarly
: TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
: internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity
: frame sync polarity

struct __sai__serial__data
: *#include <fsl_sai.h>* sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
: sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
: configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
: configure the number of bits in the first word in each frame

uint8_t dataWordNLength
: configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
: used to record the data length for dma transfer

uint8_t dataFirstBitShifted
: Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
: configure the number of words in each frame

uint32_t dataMaskedWord
configure whether the transmit word is masked

struct _sai_transceiver
#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct _sai_transfer
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct _sai_handle
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
Transfer status

sai_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

uint8_t bitWidth
: Bit width for transfer, 8/16/24/32 bits

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

volatile uint8_t queueDriver
: Index for driver to get the transfer data and size

uint8_t watermark
: Watermark value

2.111 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

2.112 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.

2.113 SEMC: Smart External DRAM Controller Driver

void SEMC_GetDefaultConfig(*semc_config_t* *config)

Gets the SEMC default basic configuration structure.

The purpose of this API is to get the default SEMC configure structure for SEMC_Init(). User may use the initialized structure unchanged in SEMC_Init(), or modify some fields of the structure before calling SEMC_Init(). Example:

```
semc_config_t config;
SEMC_GetDefaultConfig(&config);
```

Parameters

- config – The SEMC configuration structure pointer.

void SEMC_Init(SEMC_Type *base, *semc_config_t* *configure)

Initializes SEMC. This function ungates the SEMC clock and initializes SEMC. This function must be called before calling any other SEMC driver functions.

Parameters

- base – SEMC peripheral base address.

- configure – The SEMC configuration structure pointer.

void SEMC_Deinit(SEMC_Type *base)

Deinitializes the SEMC module and gates the clock.

This function gates the SEMC clock. As a result, the SEMC module doesn't work after calling this function, for some IDE, calling this API may cause the next downloading operation failed. so, please call this API cautiously. Additional, users can using "#define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (1)" to disable the clock control operation in drivers.

Parameters

- base – SEMC peripheral base address.

status_t SEMC_ConfigureSDRAM(SEMC_Type *base, *semc_sdram_cs_t* cs, *semc_sdram_config_t* *config, uint32_t clkSrc_Hz)

Configures SDRAM controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- cs – The chip selection.
- config – The sdram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureNAND(SEMC_Type *base, *semc_nand_config_t* *config, uint32_t clkSrc_Hz)

Configures NAND controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The nand configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureNOR(SEMC_Type *base, *semc_nor_config_t* *config, uint32_t clkSrc_Hz)

Configures NOR controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The nor configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureSRAMWithChipSelection(SEMC_Type *base, *semc_sram_cs_t* cs, *semc_sram_config_t* *config, uint32_t clkSrc_Hz)

Configures SRAM controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- cs – The chip selection.
- config – The sram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureSRAM(SEMC_Type *base, *semc_sram_config_t* *config, uint32_t clkSrc_Hz)

Configures SRAM controller in SEMC.

Deprecated:

Do not use this function. It has been superceded by SEMC_ConfigureSRAMWithChipSelection.

Parameters

- base – SEMC peripheral base address.
- config – The sram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureDBI(SEMC_Type *base, *semc_dbi_config_t* *config, uint32_t clkSrc_Hz)

Configures DBI controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The dbi configuration.
- clkSrc_Hz – The SEMC clock frequency.

static inline void SEMC_EnableInterrupts(SEMC_Type *base, uint32_t mask)

Enables the SEMC interrupt.

This function enables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See semc_interrupt_enable_t. For example, to enable the IP command done and error interrupt, do the following.

```
SEMC_EnableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
```

Parameters

- base – SEMC peripheral base address.
- mask – SEMC interrupts to enable. This is a logical OR of the enumeration :: semc_interrupt_enable_t.

static inline void SEMC_DisableInterrupts(SEMC_Type *base, uint32_t mask)

Disables the SEMC interrupt.

This function disables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See semc_interrupt_enable_t. For example, to disable the IP command done and error interrupt, do the following.

```
SEMC_DisableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
```

Parameters

- base – SEMC peripheral base address.
- mask – SEMC interrupts to disable. This is a logical OR of the enumeration :: semc_interrupt_enable_t.

static inline bool SEMC_GetStatusFlag(SEMC_Type *base)

Gets the SEMC status.

This function gets the SEMC interrupts event status. User can use the a logical OR of enumeration member as a mask. See semc_interrupt_enable_t.

Parameters

- base – SEMC peripheral base address.

Returns

status flag, use status flag in semc_interrupt_enable_t to get the related status.

static inline void SEMC_ClearStatusFlags(SEMC_Type *base, uint32_t mask)

Clears the SEMC status flag state.

The following status register flags can be cleared SEMC interrupt status.

Parameters

- base – SEMC base pointer
- mask – The status flag mask, a logical OR of enumeration member semc_interrupt_enable_t.

static inline bool SEMC_IsInIdle(SEMC_Type *base)

Check if SEMC is in idle.

Parameters

- base – SEMC peripheral base address.

Returns

True SEMC is in idle, false is not in idle.

status_t SEMC_SendIPCommand(SEMC_Type *base, *semc_mem_type_t* memType, uint32_t address, uint32_t command, uint32_t write, uint32_t *read)

SEMC IP command access.

Parameters

- base – SEMC peripheral base address.
- memType – SEMC memory type. refer to “semc_mem_type_t”
- address – SEMC device address.
- command – SEMC IP command. For NAND device, we should use the SEMC_BuildNandIPCommand to get the right nand command. For NOR/DBI device, take refer to “semc_ipcmd_nor_dbi_t”. For SRAM device, take refer to “semc_ipcmd_sram_t”. For SDRAM device, take refer to “semc_ipcmd_sdram_t”.
- write – Data for write access.
- read – Data pointer for read data out.

static inline uint16_t SEMC_BuildNandIPCommand(uint8_t userCommand, *semc_ipcmd_nand_addrmode_t* addrMode, *semc_ipcmd_nand_cmdmode_t* cmdMode)

Build SEMC IP command for NAND.

This function build SEMC NAND IP command. The command is build of user command code, SEMC address mode and SEMC command mode.

Parameters

- userCommand – NAND device normal command.
- addrMode – NAND address mode. Refer to “semc_ipcmd_nand_addrmode_t”.
- cmdMode – NAND command mode. Refer to “semc_ipcmd_nand_cmdmode_t”.

static inline bool SEMC_IsNandReady(SEMC_Type *base)

Check if the NAND device is ready.

Parameters

- base – SEMC peripheral base address.

Returns

True NAND is ready, false NAND is not ready.

status_t SEMC_IPCommandNandWrite(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory write through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NAND device address.
- data – Data for write access.
- size_bytes – Data length.

status_t SEMC_IPCommandNandRead(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory read through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NAND device address.
- data – Data pointer for data read out.
- size_bytes – Data length.

status_t SEMC_IPCommandNorWrite(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory write through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NOR device address.
- data – Data for write access.
- size_bytes – Data length.

status_t SEMC_IPCommandNorRead(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory read through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NOR device address.
- data – Data pointer for data read out.
- size_bytes – Data length.

FSL_SEMC_DRIVER_VERSION

SEMC driver version.

SEMC status, _semc_status.

Values:

enumerator kStatus_SEMC_InvalidDeviceType
Invalid device type.

enumerator kStatus_SEMC_IpCommandExecutionError
IP command execution error.

enumerator kStatus_SEMC_AxiCommandExecutionError
AXI command execution error.

enumerator kStatus_SEMC_InvalidMemorySize
Invalid memory sie.

enumerator kStatus_SEMC_InvalidIpcmdDataSize
Invalid IP command data size.

enumerator kStatus_SEMC_InvalidAddressPortWidth
Invalid address port width.

enumerator kStatus_SEMC_InvalidDataPortWidth
Invalid data port width.

enumerator kStatus_SEMC_InvalidSwPinmuxSelection
Invalid SW pinmux selection.

enumerator kStatus_SEMC_InvalidBurstLength
Invalid burst length

enumerator kStatus_SEMC_InvalidColumnAddressBitWidth
Invalid column address bit width.

enumerator kStatus_SEMC_InvalidBaseAddress
Invalid base address.

enumerator kStatus_SEMC_InvalidTimerSetting
Invalid timer setting.

enum _semc_mem_type
SEMC memory device type.

Values:

enumerator kSEMC_MemType_SDRAM
SDRAM

enumerator kSEMC_MemType_SRAM
SRAM

enumerator kSEMC_MemType_NOR
NOR

enumerator kSEMC_MemType_NAND
NAND

enumerator kSEMC_MemType_8080

i.

enum _semc_waitready_polarity
SEMC WAIT/RDY polarity.

Values:

enumerator kSEMC_LowActive
Low active.

enumerator kSEMC_HighActive
High active.

enum _semc_sdram_cs
SEMC SDRAM Chip selection .

Values:

enumerator kSEMC_SDRAM_CS0
SEMC SDRAM CS0.

enumerator kSEMC_SDRAM_CS1
SEMC SDRAM CS1.

enumerator kSEMC_SDRAM_CS2
SEMC SDRAM CS2.

enumerator kSEMC_SDRAM_CS3
SEMC SDRAM CS3.

enum _semc_sram_cs
SEMC SRAM Chip selection .

Values:

enumerator kSEMC_SRAM_CS0
SEMC SRAM CS0.

enum _semc_nand_access_type
SEMC NAND device type.

Values:

enumerator kSEMC_NAND_ACCESS_BY_AXI
Access to NAND flash by AXI bus.

enumerator kSEMC_NAND_ACCESS_BY_IPCMD
Access to NAND flash by IP bus.

enum _semc_interrupt_enable
SEMC interrupts .

Values:

enumerator kSEMC_IPCmdDoneInterrupt
Ip command done interrupt.

enumerator kSEMC_IPCmdErrInterrupt
Ip command error interrupt.

enumerator kSEMC_AXICmdErrInterrupt
AXI command error interrupt.

enumerator kSEMC_AXIBusErrInterrupt
AXI bus error interrupt.

enum _semc_ipcmd_datasize

SEMC IP command data size in bytes.

Values:

enumerator kSEMC_IPcmdDataSize_1bytes

The IP command data size 1 byte.

enumerator kSEMC_IPcmdDataSize_2bytes

The IP command data size 2 byte.

enumerator kSEMC_IPcmdDataSize_3bytes

The IP command data size 3 byte.

enumerator kSEMC_IPcmdDataSize_4bytes

The IP command data size 4 byte.

enum _semc_refresh_time

SEMC auto-refresh timing.

Values:

enumerator kSEMC_RefreshThreeClocks

The refresh timing with three bus clocks.

enumerator kSEMC_RefreshSixClocks

The refresh timing with six bus clocks.

enumerator kSEMC_RefreshNineClocks

The refresh timing with nine bus clocks.

enum _semc_caslatency

CAS latency.

Values:

enumerator kSEMC_LatencyOne

Latency 1.

enumerator kSEMC_LatencyTwo

Latency 2.

enumerator kSEMC_LatencyThree

Latency 3.

enum _semc_sdram_column_bit_num

SEMC sdram column address bit number.

Values:

enumerator kSEMC_SdramColunm_12bit

12 bit.

enumerator kSEMC_SdramColunm_11bit

11 bit.

enumerator kSEMC_SdramColunm_10bit

10 bit.

enumerator kSEMC_SdramColunm_9bit

9 bit.

enumerator kSEMC_SdramColunm_8bit

8 bit.

enum _semc_sdram_burst_len

SEMC sdram burst length.

Values:

enumerator kSEMC_Sdram_BurstLen1

According to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1. Burst length 1

enum _semc_nand_column_bit_num

SEMC nand column address bit number.

Values:

enumerator kSEMC_NandColum_16bit

16 bit.

enumerator kSEMC_NandColum_15bit

15 bit.

enumerator kSEMC_NandColum_14bit

14 bit.

enumerator kSEMC_NandColum_13bit

13 bit.

enumerator kSEMC_NandColum_12bit

12 bit.

enumerator kSEMC_NandColum_11bit

11 bit.

enumerator kSEMC_NandColum_10bit

10 bit.

enumerator kSEMC_NandColum_9bit

9 bit.

enum _semc_nand_burst_len

SEMC nand burst length.

Values:

enumerator kSEMC_Nand_BurstLen1

Burst length 1

enumerator kSEMC_Nand_BurstLen2

Burst length 2

enumerator kSEMC_Nand_BurstLen4

Burst length 4

enumerator kSEMC_Nand_BurstLen8

Burst length 8

enumerator kSEMC_Nand_BurstLen16

Burst length 16

enumerator kSEMC_Nand_BurstLen32

Burst length 32

enumerator kSEMC_Nand_BurstLen64

Burst length 64

enum _semc_norsram_column_bit_num

SEMC nor/sram column address bit number.

Values:

enumerator kSEMC_NorColum_12bit

12 bit.

enumerator kSEMC_NorColum_11bit

11 bit.

enumerator kSEMC_NorColum_10bit

10 bit.

enumerator kSEMC_NorColum_9bit

9 bit.

enumerator kSEMC_NorColum_8bit

8 bit.

enumerator kSEMC_NorColum_7bit

7 bit.

enumerator kSEMC_NorColum_6bit

6 bit.

enumerator kSEMC_NorColum_5bit

5 bit.

enumerator kSEMC_NorColum_4bit

4 bit.

enumerator kSEMC_NorColum_3bit

3 bit.

enumerator kSEMC_NorColum_2bit

2 bit.

enum _semc_norsram_burst_len

SEMC nor/sram burst length.

Values:

enumerator kSEMC_Nor_BurstLen1

Burst length 1

enumerator kSEMC_Nor_BurstLen2

Burst length 2

enumerator kSEMC_Nor_BurstLen4

Burst length 4

enumerator kSEMC_Nor_BurstLen8

Burst length 8

enumerator kSEMC_Nor_BurstLen16

Burst length 16

enumerator kSEMC_Nor_BurstLen32

Burst length 32

enumerator kSEMC_Nor_BurstLen64

Burst length 64

enum _semc_dbi_column_bit_num

SEMC dbi column address bit number.

Values:

enumerator kSEMC_Dbi_Colum_12bit
12 bit.

enumerator kSEMC_Dbi_Colum_11bit
11 bit.

enumerator kSEMC_Dbi_Colum_10bit
10 bit.

enumerator kSEMC_Dbi_Colum_9bit
9 bit.

enumerator kSEMC_Dbi_Colum_8bit
8 bit.

enumerator kSEMC_Dbi_Colum_7bit
7 bit.

enumerator kSEMC_Dbi_Colum_6bit
6 bit.

enumerator kSEMC_Dbi_Colum_5bit
5 bit.

enumerator kSEMC_Dbi_Colum_4bit
4 bit.

enumerator kSEMC_Dbi_Colum_3bit
3 bit.

enumerator kSEMC_Dbi_Colum_2bit
2 bit.

enum _semc_dbi_burst_len

SEMC dbi burst length.

Values:

enumerator kSEMC_Dbi_BurstLen1
Burst length 1

enumerator kSEMC_Dbi_BurstLen2
Burst length 2

enumerator kSEMC_Dbi_Dbi_BurstLen4
Burst length 4

enumerator kSEMC_Dbi_BurstLen8
Burst length 8

enumerator kSEMC_Dbi_BurstLen16
Burst length 16

enumerator kSEMC_Dbi_BurstLen32
Burst length 32

enumerator kSEMC_Dbi_BurstLen64
Burst length 64

enum _semc_iomux_pin

SEMC IOMUXC.

Values:

enumerator kSEMC_MUXA8

MUX A8 pin.

enumerator kSEMC_MUXCSX0

MUX CSX0 pin

enumerator kSEMC_MUXCSX1

MUX CSX1 Pin.

enumerator kSEMC_MUXCSX2

MUX CSX2 Pin.

enumerator kSEMC_MUXCSX3

MUX CSX3 Pin.

enumerator kSEMC_MUXRDY

MUX RDY pin.

enum _semc_iomux_nora27_pin

SEMC NOR/PSRAM Address bit 27 A27.

Values:

enumerator kSEMC_MORA27_NONE

No NOR/SRAM A27 pin.

enumerator kSEMC_NORA27_MUXCSX3

MUX CSX3 Pin.

enumerator kSEMC_NORA27_MUXRDY

MUX RDY pin.

enum _semc_port_size

SEMC port size.

Values:

enumerator kSEMC_PortSize8Bit

8-Bit port size.

enumerator kSEMC_PortSize16Bit

16-Bit port size.

enum _semc_addr_mode

SEMC address mode.

Values:

enumerator kSEMC_AddrDataMux

SEMC address/data mux mode.

enumerator kSEMC_AdvAddrdataMux

Advanced address/data mux mode.

enumerator kSEMC_AddrDataNonMux

Address/data non-mux mode.

enum _semc_dqs_mode
SEMC DQS read strobe mode.

Values:

enumerator kSEMC_Loopbackinternal
Dummy read strobe loopbacked internally.

enumerator kSEMC_Loopbackdqspad
Dummy read strobe loopbacked from DQS pad.

enum _semc_adv_polarity
SEMC ADV signal active polarity.

Values:

enumerator kSEMC_AdvActiveLow
Adv active low.

enumerator kSEMC_AdvActiveHigh
Adv active high.

enum _semc_sync_mode
SEMC sync mode.

Values:

enumerator kSEMC_AsyncMode
Async mode.

enumerator kSEMC_SyncMode
Sync mode.

enum _semc_adv_level_control
SEMC ADV signal level control.

Values:

enumerator kSEMC_AdvHigh
Adv is high during address hold state.

enumerator kSEMC_AdvLow
Adv is low during address hold state.

enum _semc_rdy_polarity
SEMC RDY signal active polarity.

Values:

enumerator kSEMC_RdyActiveLow
Adv active low.

enumerator kSEMC_RdyActivehigh
Adv active low.

enum _semc_ipcmd_nand_addrmode
SEMC IP command for NAND: address mode.

Values:

enumerator kSEMC_NANDAM_ColumnRow
Address mode: column and row address(5Byte-CA0/CA1/RA0/RA1/RA2).

enumerator kSEMC_NANDAM_ColumnCA0
Address mode: column address only(1 Byte-CA0).

enumerator kSEMC_NANDAM_ColumnCA0CA1
Address mode: column address only(2 Byte-CA0/CA1).

enumerator kSEMC_NANDAM_RawRA0
Address mode: row address only(1 Byte-RA0).

enumerator kSEMC_NANDAM_RawRA0RA1
Address mode: row address only(2 Byte-RA0/RA1).

enumerator kSEMC_NANDAM_RawRA0RA1RA2
Address mode: row address only(3 Byte-RA0).

enum _semc_ipcmd_nand_cmdmode
SEMC IP command for NAND command mode.

Values:

enumerator kSEMC_NANDCM_Command
command.

enumerator kSEMC_NANDCM_CommandHold
Command hold.

enumerator kSEMC_NANDCM_CommandAddress
Command address.

enumerator kSEMC_NANDCM_CommandAddressHold
Command address hold.

enumerator kSEMC_NANDCM_CommandAddressRead
Command address read.

enumerator kSEMC_NANDCM_CommandAddressWrite
Command address write.

enumerator kSEMC_NANDCM_CommandRead
Command read.

enumerator kSEMC_NANDCM_CommandWrite
Command write.

enumerator kSEMC_NANDCM_Read
Read.

enumerator kSEMC_NANDCM_Write
Write.

enum _semc_nand_address_option
SEMC NAND address option.

Values:

enumerator kSEMC_NandAddrOption_5byte_CA2RA3
CA0+CA1+RA0+RA1+RA2

enumerator kSEMC_NandAddrOption_4byte_CA2RA2
CA0+CA1+RA0+RA1

enumerator kSEMC_NandAddrOption_3byte_CA2RA1
CA0+CA1+RA0

enumerator kSEMC_NandAddrOption_4byte_CA1RA3
CA0+RA0+RA1+RA2

enumerator kSEMC_NandAddrOption_3byte_CA1RA2
CA0+RA0+RA1

enumerator kSEMC_NandAddrOption_2byte_CA1RA1
CA0+RA0

enum _semc_ipcmd_nor_dbi
SEMC IP command for NOR.

Values:

enumerator kSEMC_NORDBICM_Read
NOR read.

enumerator kSEMC_NORDBICM_Write
NOR write.

enum _semc_ipcmd_sram
SEMC IP command for SRAM.

Values:

enumerator kSEMC_SRAMCM_ArrayRead
SRAM memory array read.

enumerator kSEMC_SRAMCM_ArrayWrite
SRAM memory array write.

enumerator kSEMC_SRAMCM_RegRead
SRAM memory register read.

enumerator kSEMC_SRAMCM_RegWrite
SRAM memory register write.

enum _semc_ipcmd_sdram
SEMC IP command for SDARM.

Values:

enumerator kSEMC_SDRAMCM_Read
SDRAM memory read.

enumerator kSEMC_SDRAMCM_Write
SDRAM memory write.

enumerator kSEMC_SDRAMCM_Modeset
SDRAM MODE SET.

enumerator kSEMC_SDRAMCM_Active
SDRAM active.

enumerator kSEMC_SDRAMCM_AutoRefresh
SDRAM auto-refresh.

enumerator kSEMC_SDRAMCM_SelfRefresh
SDRAM self-refresh.

enumerator kSEMC_SDRAMCM_Precharge
SDRAM precharge.

enumerator kSEMC_SDRAMCM_Prechargeall
SDRAM precharge all.

typedef enum *_semc_mem_type* semc_mem_type_t
SEMC memory device type.

typedef enum *_semc_waitready_polarity* semc_waitready_polarity_t
SEMC WAIT/RDY polarity.

typedef enum *_semc_sdram_cs* semc_sdram_cs_t
SEMC SDRAM Chip selection .

typedef enum *_semc_sram_cs* semc_sram_cs_t
SEMC SRAM Chip selection .

typedef enum *_semc_nand_access_type* semc_nand_access_type_t
SEMC NAND device type.

typedef enum *_semc_interrupt_enable* semc_interrupt_enable_t
SEMC interrupts .

typedef enum *_semc_ipcmd_datasize* semc_ipcmd_datasize_t
SEMC IP command data size in bytes.

typedef enum *_semc_refresh_time* semc_refresh_time_t
SEMC auto-refresh timing.

typedef enum *_semc_caslatency* semc_caslatency_t
CAS latency.

typedef enum *_semc_sdram_column_bit_num* semc_sdram_column_bit_num_t
SEMC sdram column address bit number.

typedef enum *_semc_sdram_burst_len* sem_sdram_burst_len_t
SEMC sdram burst length.

typedef enum *_semc_nand_column_bit_num* semc_nand_column_bit_num_t
SEMC nand column address bit number.

typedef enum *_semc_nand_burst_len* sem_nand_burst_len_t
SEMC nand burst length.

typedef enum *_semc_norsram_column_bit_num* semc_norsram_column_bit_num_t
SEMC nor/sram column address bit number.

typedef enum *_semc_norsram_burst_len* sem_norsram_burst_len_t
SEMC nor/sram burst length.

typedef enum *_semc_dbi_column_bit_num* semc_dbi_column_bit_num_t
SEMC dbi column address bit number.

typedef enum *_semc_dbi_burst_len* sem_dbi_burst_len_t
SEMC dbi burst length.

typedef enum *_semc_iomux_pin* semc_iomux_pin
SEMC IOMUXC.

typedef enum *_semc_iomux_nora27_pin* semc_iomux_nora27_pin
SEMC NOR/PSRAM Address bit 27 A27.

typedef enum *_semc_port_size* smec_port_size_t
SEMC port size.

typedef enum *_semc_addr_mode* semc_addr_mode_t
SEMC address mode.

typedef enum *_semc_dqs_mode* semc_dqs_mode_t
SEMC DQS read strobe mode.

typedef enum *_semc_adv_polarity* semc_adv_polarity_t
SEMC ADV signal active polarity.

typedef enum *_semc_sync_mode* semc_sync_mode_t
SEMC sync mode.

typedef enum *_semc_adv_level_control* semc_adv_level_control_t
SEMC ADV signal level control.

typedef enum *_semc_rdy_polarity* semc_rdy_polarity_t
SEMC RDY signal active polarity.

typedef enum *_semc_ipcmd_nand_addrmode* semc_ipcmd_nand_addrmode_t
SEMC IP command for NAND: address mode.

typedef enum *_semc_ipcmd_nand_cmdmode* semc_ipcmd_nand_cmdmode_t
SEMC IP command for NAND command mode.

typedef enum *_semc_nand_address_option* semc_nand_address_option_t
SEMC NAND address option.

typedef enum *_semc_ipcmd_nor_dbi* semc_ipcmd_nor_dbi_t
SEMC IP command for NOR.

typedef enum *_semc_ipcmd_sram* semc_ipcmd_sram_t
SEMC IP command for SRAM.

typedef enum *_semc_ipcmd_sdram* semc_ipcmd_sdram_t
SEMC IP command for SDARM.

typedef struct *_semc_sdram_config* semc_sdram_config_t
SEMC SDRAM configuration structure.

a. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2, 2^3, 2^4 .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.

b. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer_n16cycle = 0, it means the prescaler timer period is 256 * 16 clock cycles. For precalerIf precalerTimer_n16cycle not equal to 0, The prescaler timer period is prescalePeriod_N16Cycle * 16 clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

typedef struct *_semc_nand_timing_config* semc_nand_timing_config_t
SEMC NAND device timing configuration structure.

typedef struct *_semc_nand_config* semc_nand_config_t
SEMC NAND configuration structure.

typedef struct *_semc_nor_config* semc_nor_config_t
SEMC NOR configuration structure.

typedef struct *_semc_sram_config* semc_sram_config_t
SEMC SRAM configuration structure.

typedef struct *_semc_dbi_config* semc_dbi_config_t
SEMC DBI configuration structure.

typedef struct *_semc_queuea_weight_struct* semc_queuea_weight_struct_t

SEMC AXI queue a weight setting structure.

typedef union *_semc_queuea_weight* semc_queuea_weight_t

SEMC AXI queue a weight setting union.

typedef struct *_semc_queueb_weight_struct* semc_queueb_weight_struct_t

SEMC AXI queue b weight setting structure.

typedef union *_semc_queueb_weight* semc_queueb_weight_t

SEMC AXI queue b weight setting union.

typedef struct *_semc_axi_queueweight* semc_axi_queueweight_t

SEMC AXI queue weight setting.

typedef struct *_semc_config_t* semc_config_t

SEMC configuration structure.

busTimeoutCycles: when busTimeoutCycles is zero, the bus timeout cycle is 255*1024. otherwise the bus timeout cycles is busTimeoutCycles*1024. cmdTimeoutCycles: is used for command execution timeout cycles. it's similar to the busTimeoutCycles.

struct _semc_sdram_config

#include <fsl_semc.h> SEMC SDRAM configuration structure.

a. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2, 2^3, 2^4 .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.

b. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer_n16cycle = 0, it means the prescaler timer period is 256 * 16 clock cycles. For precalerIf precalerTimer_n16cycle not equal to 0, The prescaler timer period is prescalePeriod_N16Cycle * 16 clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

Public Members

semc_iomux_pin csxPinMux

CS pin mux. The kSEMC_MUXA8 is not valid in sdram pin mux setting.

uint32_t address

The base address.

uint32_t memsize_kbytes

The memory size in unit of kbytes.

smec_port_size_t portSize

Port size.

sem_sdram_burst_len_t burstLen

Burst length.

semc_sdram_column_bit_num_t columnAddrBitNum

Column address bit number.

semc_caslatency_t casLatency

CAS latency.

uint8_t tPrecharge2Act_Ns
: Precharge to active wait time in unit of nanosecond.

uint8_t tAct2ReadWrite_Ns
: Act to read/write wait time in unit of nanosecond.

uint8_t tRefreshRecovery_Ns
: Refresh recovery time in unit of nanosecond.

uint8_t tWriteRecovery_Ns
: write recovery time in unit of nanosecond.

uint8_t tCkeOff_Ns
: CKE off minimum time in unit of nanosecond.

uint8_t tAct2Prechage_Ns
: Active to precharge in unit of nanosecond.

uint8_t tSelfRefRecovery_Ns
: Self refresh recovery time in unit of nanosecond.

uint8_t tRefresh2Refresh_Ns
: Refresh to refresh wait time in unit of nanosecond.

uint8_t tAct2Act_Ns
: Active to active wait time in unit of nanosecond.

uint32_t tPrescalePeriod_Ns
: Prescaler timer period should not be larger than 256 * 16 * clock cycle.

uint32_t tIdleTimeout_Ns
: Idle timeout in unit of prescale time period.

uint32_t refreshPeriod_nsPerRow
: Refresh timer period like 64ms * 1000000/8192 .

uint32_t refreshUrgThreshold
: Refresh urgent threshold.

uint8_t refreshBurstLen
: Refresh burst length.

uint8_t delayChain
: Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

uint8_t autofreshTimes
: Auto Refresh cycles times.

struct _semc_nand_timing_config
: *#include <fsl_semc.h>* SEMC NAND device timing configuration structure.

Public Members

uint8_t tCeSetup_Ns
: CE setup time: tCS.

uint8_t tCeHold_Ns
: CE hold time: tCH.

uint8_t tCeInterval_Ns
: CE interval time:tCEITV.

uint8_t tWeLow_Ns
: WE low time: tWP.

uint8_t tWeHigh_Ns
: WE high time: tWH.

uint8_t tReLow_Ns
: RE low time: tRP.

uint8_t tReHigh_Ns
: RE high time: tREH.

uint8_t tTurnAround_Ns
: Turnaround time for async mode: tTA.

uint8_t tWehigh2Relow_Ns
: WE# high to RE# wait time: tWHR.

uint8_t tRehigh2Welow_Ns
: RE# high to WE# low wait time: tRHW.

uint8_t tAle2WriteStart_Ns
: ALE to write start wait time: tADL.

uint8_t tReady2Relow_Ns
: Ready to RE# low min wait time: tRR.

uint8_t tWehigh2Busy_Ns
: WE# high to busy wait time: tWB.

struct _semc_nand_config
: *#include <fsl_semc.h>* SEMC NAND configuration structure.

Public Members

semc_iomux_pin cePinMux
: The CE pin mux setting. The kSEMC_MUXRDY is not valid for CE pin setting.

uint32_t axiAddress
: The base address for AXI nand.

uint32_t axiMemsize_kbytes
: The memory size in unit of kbytes for AXI nand.

uint32_t ipgAddress
: The base address for IPG nand .

uint32_t ipgMemsize_kbytes
: The memory size in unit of kbytes for IPG nand.

semc_rdy_polarity_t rdyactivePolarity
: Wait ready polarity.

bool edoModeEnabled
: EDO mode enabled.

semc_nand_column_bit_num_t columnAddrBitNum
: Column address bit number.

semc_nand_address_option_t arrayAddrOption
: Address option.

sem_nand_burst_len_t burstLen
Burst length.

smec_port_size_t portSize
Port size.

semc_nand_timing_config_t *timingConfig
SEMC nand timing configuration.

struct _semc_nor_config
#include <fsl_semc.h> SEMC NOR configuration structure.

Public Members

semc_iomux_pin cePinMux
The CE# pin mux setting.

semc_iomux_nora27_pin addr27
The Addr bit 27 pin mux setting.

uint32_t address
The base address.

uint32_t memsize_kbytes
The memory size in unit of kbytes.

uint8_t addrPortWidth
The address port width.

semc_rdy_polarity_t rdyactivePolarity
Wait ready polarity.

semc_adv_polarity_t advActivePolarity
ADV# polarity.

semc_norsram_column_bit_num_t columnAddrBitNum
Column address bit number.

semc_addr_mode_t addrMode
Address mode.

sem_norsram_burst_len_t burstLen
Burst length.

smec_port_size_t portSize
Port size.

uint8_t tCeSetup_Ns
The CE setup time.

uint8_t tCeHold_Ns
The CE hold time.

uint8_t tCeInterval_Ns
CE interval minimum time.

uint8_t tAddrSetup_Ns
The address setup time.

uint8_t tAddrHold_Ns
The address hold time.

uint8_t tWeLow_Ns
: WE low time for async mode.

uint8_t tWeHigh_Ns
: WE high time for async mode.

uint8_t tReLow_Ns
: RE low time for async mode.

uint8_t tReHigh_Ns
: RE high time for async mode.

uint8_t tTurnAround_Ns
: Turnaround time for async mode.

uint8_t tAddr2WriteHold_Ns
: Address to write data hold time for async mode.

uint8_t tWriteSetup_Ns
: Write data setup time for sync mode.

uint8_t tWriteHold_Ns
: Write hold time for sync mode.

uint8_t latencyCount
: Latency count for sync mode.

uint8_t readCycle
: Read cycle time for sync mode.

uint8_t delayChain
: Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

struct _semc_sram_config
: *#include <fsl_semc.h>* SEMC SRAM configuration structure.

Public Members

semc_iomux_pin cePinMux
: The CE# pin mux setting.

semc_iomux_nora27_pin addr27
: The Addr bit 27 pin mux setting.

uint32_t address
: The base address.

uint32_t memsize_kbytes
: The memory size in unit of kbytes.

uint8_t addrPortWidth
: The address port width.

semc_adv_polarity_t advActivePolarity
: ADV# polarity 1: active high, 0: active low.

semc_addr_mode_t addrMode
: Address mode.

sem_norsram_burst_len_t burstLen
Burst length.

smec_port_size_t portSize
Port size.

semc_sync_mode_t syncMode
Sync mode.

bool waitEnable
Wait enable.

uint8_t waitSample
Wait sample.

semc_adv_level_control_t advLevelCtrl
ADV# level control during address hold state, 1: low, 0: high.

uint8_t tCeSetup_Ns
The CE setup time.

uint8_t tCeHold_Ns
The CE hold time.

uint8_t tCeInterval_Ns
CE interval minimum time.

uint8_t readHoldTime_Ns
read hold time.

uint8_t tAddrSetup_Ns
The address setup time.

uint8_t tAddrHold_Ns
The address hold time.

uint8_t tWeLow_Ns
WE low time for async mode.

uint8_t tWeHigh_Ns
WE high time for async mode.

uint8_t tReLow_Ns
RE low time for async mode.

uint8_t tReHigh_Ns
RE high time for async mode.

uint8_t tTurnAround_Ns
Turnaround time for async mode.

uint8_t tAddr2WriteHold_Ns
Address to write data hold time for async mode.

uint8_t tWriteSetup_Ns
Write data setup time for sync mode.

uint8_t tWriteHold_Ns
Write hold time for sync mode.

uint8_t latencyCount
Latency count for sync mode.

uint8_t readCycle
: Read cycle time for sync mode.

uint8_t delayChain
: Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

struct _semc_dbi_config
#include <fsl_semc.h> SEMC DBI configuration structure.

Public Members

semc_iomux_pin csxPinMux
: The CE# pin mux.

uint32_t address
: The base address.

uint32_t memsize_kbytes
: The memory size in unit of 4kbytes.

semc_dbi_column_bit_num_t columnAddrBitNum
: Column address bit number.

sem_dbi_burst_len_t burstLen
: Burst length.

smec_port_size_t portSize
: Port size.

uint8_t tCsxSetup_Ns
: The CSX setup time.

uint8_t tCsxHold_Ns
: The CSX hold time.

uint8_t tWexLow_Ns
: WEX low time.

uint8_t tWexHigh_Ns
: WEX high time.

uint8_t tRdxLow_Ns
: RDX low time.

uint8_t tRdxHigh_Ns
: RDX high time.

uint8_t tCsxInterval_Ns
: Write data setup time.

struct _semc_queuea_weight_struct
#include <fsl_semc.h> SEMC AXI queue a weight setting structure.

Public Members

uint32_t qos
: weight of qos for queue 0 .

uint32_t aging
: weight of aging for queue 0.

uint32_t slaveHitNoswitch
: weight of read/write no switch for queue 0 .

uint32_t slaveHitSwitch
: weight of read/write switch for queue 0.

union __semc__queuea__weight
#include <fsl_semc.h> SEMC AXI queue a weight setting union.

Public Members

semc_queuea_weight_struct_t queueaConfig
: Structure configuration for queueA.

uint32_t queueaValue
: Configuration value for queueA which could directly write to the reg.

struct __semc__queueb__weight__struct
#include <fsl_semc.h> SEMC AXI queue b weight setting structure.

Public Members

uint32_t qos
: weight of qos for queue 1.

uint32_t aging
: weight of aging for queue 1.

uint32_t weightPagehit
: weight of page hit for queue 1 only .

uint32_t slaveHitNoswitch
: weight of read/write no switch for queue 1.

uint32_t bankRotation
: weight of bank rotation for queue 1 only .

union __semc__queueb__weight
#include <fsl_semc.h> SEMC AXI queue b weight setting union.

Public Members

semc_queueb_weight_struct_t queuebConfig
: Structure configuration for queueB.

uint32_t queuebValue
: Configuration value for queueB which could directly write to the reg.

struct __semc__axi__queueweight
#include <fsl_semc.h> SEMC AXI queue weight setting.

Public Members

bool queueaEnable
: Enable queue a.

semc_queuea_weight_t queueaWeight
: Weight settings for queue a.

bool queuebEnable
: Enable queue b.

semc_queueb_weight_t queuebWeight
: Weight settings for queue b.

struct _semc_config_t
: *#include <fsl_semc.h>* SEMC configuration structure.

busTimeoutCycles: when busTimeoutCycles is zero, the bus timeout cycle is 255*1024. otherwise the bus timeout cycles is busTimeoutCycles*1024. cmdTimeoutCycles: is used for command execution timeout cycles. it's similar to the busTimeoutCycles.

Public Members

semc_dqs_mode_t dqsMode
: Dummy read strobe mode: use enum in "semc_dqs_mode_t".

uint8_t cmdTimeoutCycles
: Command execution timeout cycles.

uint8_t busTimeoutCycles
: Bus timeout cycles.

semc_axi_queueweight_t queueWeight
: AXI queue weight.

2.114 SINC: SINC Filter

void SINC_Init(SINC_Type *base, const *sinc_config_t* *config)
: Initialize selected SINC instance, including clock options and channel options.

Parameters

- base – SINC peripheral base address.
- config – The pointer to sinc_config_t structure.

void SINC_Deinit(SINC_Type *base)
: De-initialize selected SINC instance.

Parameters

- base – SINC peripheral base address.

void SINC_GetDefaultConfig(*sinc_config_t* *config)
: Get default configuration.

```
config->clockPreDivider = kSINC_ClkPrescale1;
config->modClkDivider   = 2UL;
config->disableModClk0Output = false;
config->disableModClk1Output = false;
```

(continues on next page)

(continued from previous page)

```
config->disableModClk2Output = false;

config->channelsConfigArray[4] = {NULL, NULL, NULL, NULL};

config->disableDozeMode      = false;
config->enableMaster         = false;
```

Parameters

- config – The pointer to sinc_config_t structure, must not be NULL.

static inline void SINC_EnableMaster(SINC_Type *base, bool enable)

Enable/disable all function blocks enabled in their respective registers.

Parameters

- base – SINC peripheral base address.
- enable – Used to enable/disable all function blocks:
  - **true** Enable all function blocks, please note that clock must be configured previously;
  - **false** Disable all function blocks.

static inline void SINC_DoSoftwareReset(SINC_Type *base)

Reset all function blocks(except for the clock blocks), interrupt statuses.

Parameters

- base – SINC peripheral base address.

static inline void SINC_DisableDozeMode(SINC_Type *base, bool disable)

Disable/enable SINC module when the chip enters Doze or Stop mode.

Parameters

- base – SINC peripheral base address.
- disable – Used to control if module functional when the chip enters Doze and Stop mode:
  - **true** Disable SINC when the chip enters Doze or Stop mode;
  - **false** Enable SINC when the chip enters Doze or stop mode.

static inline bool SINC_CheckModulatorClockReady(SINC_Type *base, uint32_t modClkMasks)

Check whether selected modulator clocks are ready.

Note: The result of this APIs means all selected modulator clocks are (not) ready.

Parameters

- base – SINC peripheral base address.
- modClkMasks – The mask of modulator clocks, please refer to _sinc_modulator_clock.

Return values

- true – The input mask of modulator clocks are ready.
- false – The input mask of modulator clocks are not ready.

static inline void SINC_DisableModulatorClockOutput(SINC_Type *base, uint32_t modClkMasks, bool disable)

Disable/enable modulator clocks' output.

Note: By default, modulator clock's output is enabled.

Parameters

- base – SINC peripheral base address.
- modClkMasks – The mask of modulator clocks, please refer to _sinc_modulator_clock.
- disable – Used to enable/disable clock output:
  - **true** Disable modulator clocks' output;
  - **false** Enable modulator clocks' output.

static inline void SINC_SetClkPrescale(SINC_Type *base, *sinc_clock_prescale_t* clkPrescale)

Set the clock divider ratio for the modulator clock.

Parameters

- base – SINC peripheral base address.
- clkPrescale – Clock prescale value, please refer to sinc_clock_prescale_t.

static inline void SINC_SetModulatorClockDivider(SINC_Type *base, uint32_t modClkDivider)

Set modulator clock divider value.

Note: IMCLK0 = PRE_CLK / modClkDivider, the minimum clock divider ration is 2.

Parameters

- base – SINC peripheral base address.
- modClkDivider – Range from 2 to 256, 0 and 1 are prohibited, to obtain a 50% duty cycle in the MCLK output, write an even value to **modClkDivider** .

void SINC_SetChannelConfig(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_channel_config_t* *chConfig)

Set channel configurations, including input options, conversion options and protection options.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- chConfig – Pointer to sinc_channel_config_t structure, must not be NULL.

void SINC_SetChannelInputOption(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_channel_input_option_t* *chInputOption)

Set channel input options, including input bit format, input bit source, input bit delay, input clock source, input clock edge.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

- chInputOption – Pointer to sinc_channel_input_option_t structure, must not be NULL.

void SINC_SetChannelConversionOption(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_channel_conv_option_t* *chConvOption)

Set channel conversion options, including conversion mode, trigger source, and primary filter settings.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- chConvOption – Pointer to sinc_channel_conv_option_t structure, must not be NULL.

void SINC_SetChannelProtectionOption(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_channel_protection_option_t* *chProtection)

Set channel protection options, including limit check, short-circuit detector, clock-absence detector, and zero-crossing detector.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- chProtection – Pointer to sinc_channel_protection_option_t, must not be NULL.

static inline uint32_t SINC_ReadChannelResultData(SINC_Type *base, *sinc_channel_id_t* chId)

Read selected channel's result data.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

Returns

Result data of the selected channel, 24 bits width.

static inline void SINC_EnableChannelFIFO(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable FIFO transfers for the primary filter.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disable channl FIFO:
  - **true** Enable channel FIFO.
  - **false** Disable channle FIFO.

static inline void SINC_SetChannelFifoWatermark(SINC_Type *base, *sinc_channel_id_t* chId, uint8_t fifoWaterMark)

Set the FIFO watermark.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- fifoWaterMark – Specify the fifo watermark, range from 0 to 15.

static inline void SINC_EnableChannel(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable selected channel.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disable selected channel:
  - **true** Enable selected channel;
  - **false** Disable selected channel.

static inline void SINC_EnableChannelPrimaryDma(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable selected channel's primary DMA transfers when the channel's FIFO exceeds its watermark.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disable primary DMA :
  - **true** Enable primary DMA;
  - **false** Disable primary DMA.

static inline void SINC_SetChannelAltDmaSource(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_alternate_dma_source_t* altDmaSource)

Set selected channel's alternate DMA source selection.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- altDmaSource – Specify the trigger source for alternate DMA, please refer to sinc_alternate_dma_source_t.

static inline void SINC_SetChannelResultDataFormat(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_result_data_format_t* dataFormat)

Set selected channel's result data format.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- dataFormat – Specify the result data format, please refer to sinc_result_data_format_t.

static inline uint8_t SINC_GetChannelFifoCount(SINC_Type *base, *sinc_channel_id_t* chId)

Get the number of remaining data entries in the FIFO.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

Returns

The number of remaining data entries in the FIFO.

static inline bool SINC_CheckChannelResultDataReady(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether the data in selected channel's result data register is stable when FIFO is disabled.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, refer to sinc_channel_id_t for details.

Return values

- true – Data in selected channel's result data regiter is stable.
- false – Data in selected channel's result data register is not stable.

static inline bool SINC_CheckChannelFifoEmpty(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether selected channel's FIFO is empty.

Parameters

- base – SINC peripheral base address.
- chId – The id of sinc channel to check.

Return values

- true – Selected channel's FIFO is empty.
- false – Selected channel's FIFO is not empty.

static inline void SINC_AffirmChannelSoftwareTrigger(SINC_Type *base, uint32_t chMask)

Trigger selected channel's conversion.

Parameters

- base – SINC peripheral base address.
- chMask – The mask of channels to trigger.

static inline void SINC_NegateChannelSoftwareTrigger(SINC_Type *base, uint32_t chMask)

Negate the trigger of selected channel.

Parameters

- base – SINC peripheral base address.
- chMask – The mask of channels.

static inline void SINC_SetChannelConversionMode(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_conv_mode_t* mode)

Set selected channel's conversion mode.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- mode – The conversion mode to set, please refer to sinc_conv_mode_t.

static inline void SINC_SetChannelTriggerSource(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_conv_trigger_source_t* triggerSource)

Set selected channel's trigger source.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

- triggerSource – Trigger source to set, please refer to sinc_conv_trigger_source_t.

static inline void SINC_SetChannelMultipurposeData(SINC_Type *base, *sinc_channel_id_t* chId, uint32_t data)

Set multipurpose data to selected channel.

Note: If input bit format is set as ManchesterCode, multipurpose data indicates the Manchester decoder threshold value and is 11 bits width; if input bit format is set as parallel, multipurpose data indicates the parallel 16-bit data and is 16 bits width; if input bit format is set as serial, multipurpose data indicates the serial data and is 32 bits width.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- data – Multipurpose data to set.

static inline void SINC_SetChannelPfOrder(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_primary_filter_order_t* pfOrder)

Set selected channel's PF order.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- pfOrder – Primary filter order to set, please refer to sinc_primary_filter_order_t

static inline void SINC_SetChannelPfOsr(SINC_Type *base, *sinc_channel_id_t* chId, uint16_t pfOsr)

Set selected channel's PF over sample rate.

Note: If PF order is third order and data format is signed, the maximum OSR value is 1289, if PF order is third order and data format is unsigned, the maximum OSR value is 1624, otherwise the maximum OSR value is 2047.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- pfOsr – Control the channel's PF OSR, the minimum permissible value is 3, low value produce unpredictable result, the maximum permissible value depend on PF order and the desired data format.

static inline void SINC_SetChannelPfHpfAlphaCoeff(SINC_Type *base, *sinc_channel_id_t* chId, *sin_primary_filter_hpf_alpha_coeff_t* pfHpfAlphaCoeff)

set selected channel's HPF DC remover Alpha coefficient.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

- pfHpfAlphaCoeff – Specify the HPF alpha coefficient or disable HPF as described in sin_primary_filter_hpf_alpha_coeff_t.

static inline void SINC_SetChannelPfShiftConfig(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_primary_filter_shift_direction_t* pfShiftDirection, uint8_t pfShiftBitsNum)

Set the value that shifts the PF data for the correct 24-bit precision.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- pfShiftDirection – Specify the PF shift direction, including right and left.
- pfShiftBitsNum – Specify the PF shift value, range from 0 to 15.

static inline void SINC_SetChannelPfBiasConfig(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_primary_filter_bias_sign_t* pfBiasSign, uint32_t pfBiasValue)

Set the bias offset for the selected channel's PF.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- pfBiasSign – Specify the bias sign, please refer to sinc_primary_filter_bias_sign_t for details.
- pfBiasValue – The bias value to subtracted from the output of PF shift block, range from 0 to 0x7FFFFFUL.

static inline void SINC_EnableChannelPrimaryFilter(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable selected channel's primary filter.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable primary filter:
  - **true** Enable channel's PF;
  - **false** Disable channel's PF.

static inline bool SINC_CheckChannelParallelSerialDataReady(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether selected channel's multipurpose data is ready to write parallel or serial data.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, refer to sinc_channel_id_t for details.

Return values

- true – Selected channel's multipurpose data is ready to write parallel or serial data.
- false – Selected channel's multipurpose data is not ready to write parallel or serial data.

static inline bool SINC_CheckChannelPrimaryCICSaturation(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether primary CIC filter saturation occurred.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, refer to sinc_channel_id_t for details.

Return values

- true – Selected channel's primary CIC filter saturation occurred.
- false – Selected channel's primary CIC filter saturation did not occurred.

static inline bool SINC_CheckChannelHPFSaturation(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether HPF saturation occurred.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, refer to sinc_channel_id_t for details.

Return values

- true – Selected channel's HPF saturation occurred.
- false – Selected channel's HPF saturation did not occurred.

static inline bool SINC_CheckChannelShiftSaturation(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether Shift saturation occurred.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, refer to sinc_channel_id_t for details.

Return values

- true – Selected channel's shift saturation occurred.
- false – Selected channel's shift saturation did not occurred.

static inline bool SINC_CheckChannelBiasSaturation(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether bias saturation occurred.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, refer to sinc_channel_id_t for details.

Return values

- true – Selected channel's bias saturation occurred.
- false – Selected channel's bias saturation did not occurred.

uint8_t SINC_GetChannelConversionCount(SINC_Type *base, *sinc_channel_id_t* chId)

Get selected channel's number of conversions.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, refer to sinc_channel_id_t for details.

Returns

uint8_t Selected channel's number of conversions.

static inline bool SINC_CheckChannelConvProgress(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether the selected channel is in conversion.

Parameters

- base – SINC peripheral base address.
- chId – The id of sinc channel to check.

Return values

- false – Selected channel conversion not in progress.
- true – Selected channel conversion in progress.

static inline bool SINC_CheckChannelReadyForConv(SINC_Type *base, *sinc_channel_id_t* chId)

Check whether the selected channel is ready for conversion.

Parameters

- base – SINC peripheral base address.
- chId – The id of sinc channel to check.

Return values

- true – Selected channel is ready for conversion.
- false – Selected channel is not ready for conversion.

static inline void SINC_SetChannelLowLimitThreshold(SINC_Type *base, *sinc_channel_id_t* chId, uint32_t lowLimitThreshold)

Set selected channel's low-limit threshold value.

Note: When the data exceeds the low-limit threshold value, a low-limit event occurs, and the limit threshold format is determines by channel's result data format sinc_result_data_format_t.

Note: Low limit value must lower than high limit value, otherwise the low-limit threshold does not work.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- lowLimitThreshold – Specify the low-limit threshold value, range from 0 to 0xFFFFFFUL.

static inline void SINC_SetChannelHighLimitThreshold(SINC_Type *base, *sinc_channel_id_t* chId, uint32_t highLimitThreshold)

Set selected channel's high-limit threshold value.

Note: When the data exceeds the high-limit threshold value, a high-limit event occurs, and the limit threshold format is determines by channel's result data format sinc_result_data_format_t.

Parameters

- base – SINC peripheral base address.

- chId – Selected channel id, please refer to sinc_channel_id_t.
- highLimitThreshold – Specify the high-limit threshold value, range from 0 to 0xFFFFFFUL.

static inline void SINC_SetChannelLimitDetectorMode(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_limit_detector_mode_t* mode)

Set selected channel's limit detector mode.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- mode – Specify the mode of limit detector, please refer to sinc_limit_detector_mode_t.

static inline void SINC_EnableChannelHighLimitBreakSignal(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable selected channel's high limit break signal.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disable high limit break signal:
  - **true** Enable the automatic assertion of the BREAK_HIGH signal when SINC detects a high-limit event on the selected channel.
  - **false** Disable high limit break signal.

static inline void SINC_EnableChannelWindowLimitBreakSignal(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable selected channel's window limit break signal.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disable window limit break signal:
  - **true** Enable the automatic assertion of the BREAK_WIN signal when SINC detects a window-limit event on the selected channel.
  - **false** Disable window limit break signal.

static inline void SINC_EnableChannelLowLimitBreakSignal(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable selected channel's low limit break signal.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disable low limit break signal:
  - **true** Enable the automatic assertion of the BREAK_LOW signal when SINC detects a low-limit event on the selected channel.
  - **false** Disable low limit break signal.

static inline void SINC_SetChannelScdOperateMode(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_scd_operate_mode_t* opMode)

Set selected channel's short-circuit detector operate mode.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- opMode – Specify the operate mode to set, please refer to sinc_scd_operate_mode_t.

static inline void SINC_SetChannelScdLimitThreshold(SINC_Type *base, *sinc_channel_id_t* chId, uint8_t u8ScdLimitThreshold)

Set selected channel's Scd limit threshold.

Note:

The SCD counter tracks the number of received bits with the same repeating value(always 0 or always 1, set by

SINC_SetChannelScdOption()), if that number exceeds the scdLimitThreshold, an SCD event occurs on the associated channel.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- u8ScdLimitThreshold – Specify the threshold value for the SCD counter, range from 2 to 255.

static inline void SINC_SetChannelScdOption(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_scd_option_t* option)

Set selected channel's SDC option.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- option – Specify which repeating bit value increments the SCD counter.

static inline void SINC_EnableChannelScdBreakSignal(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable the automatic assertion of the BREAK_SCD signal when SINC detects an SCD event on the selected channel.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disable SCD break signal:
  - **true** Enable SCD break signal.
  - **false** Disable SCD break signal.

static inline void SINC_SetChannelCadLimitThreshold(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_cad_threshold_t* cadLimitThreshold)

Set the threshold value for the CAD counter.

Note: The CAD counter tracks the number of clock cycles during which SINC does not detect a clock, if that number exceeds the threshold value, a CAD event occurs on the selected channel.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- cadLimitThreshold – Specify the threshold value for the CAD counter, please refer to sinc_cad_threshold_t.

static inline void SINC_EnableChannelCadBreakSignal(SINC_Type *base, *sinc_channel_id_t* chId, bool enable)

Enable/disable the automatic assertion of the BREAK_CAD signal when SINC detects a CAD event on the assoicated channel.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- enable – Used to enable/disble CAD break signal:
  - **true** Enable selected channel's CAD break signal;
  - **false** Disable selected channel's CAD break signal.

static inline void SINC_SetChannelZcdOperateMode(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_zero_cross_operate_mode_t* opMode)

Set selected channel's zero-crossing detector operate mode.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- opMode – Specify the operate mode, please refer to sinc_zero_cross_operate_mode_t.

static inline void SINC_SetChannelPulseTriggerMux(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_pulse_trigger_mux_t* pulseTrigMux)

Set selected channel's pulse trigger mux.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.
- pulseTrigMux – Used to selected the signal for pulse-trigger output.

static inline void SINC_SetChannelDebugOutput(SINC_Type *base, *sinc_channel_id_t* chId, *sinc_debug_output_t* debugOutput)

Set selected channel's debug output.

Parameters

- base – SINC peripheral base address.

- chId – Selected channel id, please refer to sinc_channel_id_t.
- debugOutput – Used to select debug output, please refer to sinc_debug_output_t.

static inline void SINC_LatchChannelDebugProceduce(SINC_Type *base, *sinc_channel_id_t* chId)

Start selected channel's debug data latch proceduce.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

static inline bool SINC_CheckChannelDebugDataValid(SINC_Type *base, *sinc_channel_id_t* chId)

Check if the selected channel's debug data is valid.

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

Return values

- true – Data is valid.
- false – Data is invalid.

static inline uint32_t SINC_GetChannelDebugData(SINC_Type *base, *sinc_channel_id_t* chId)

Return selected channel's the debug data that requested by SINC_SetChannelDebugOutput().

Parameters

- base – SINC peripheral base address.
- chId – Selected channel id, please refer to sinc_channel_id_t.

Returns

Selected channel's debug data.

static inline void SINC_EnableInterrupts(SINC_Type *base, uint64_t interruptMasks)

Enable the mask of interrupts, such as channel data ready interrupt, channel limit detect interrupt and so on.

Parameters

- base – SINC peripheral base address.
- interruptMasks – Mask of interrupts to enable, should be the OR'ed value of sinc_interrupt_enable_t.

static inline void SINC_DisableInterrupts(SINC_Type *base, uint64_t interruptMasks)

Enable the mask of interrupts, such as channel data ready interrupt, channel limit detect interrupt and so on.

Parameters

- base – SINC peripheral base address.
- interruptMasks – Mask of interrupts to disable, should be the OR'ed value of sinc_interrupt_enable_t.

static inline uint64_t SINC_GetInterruptStatus(SINC_Type *base)

Get interrupt status flags.

Parameters

- base – SINC peripheral base address.

Returns

SINC module's interrupt status flags, the OR'ed value of sinc_interrupt_status_t.

static inline void SINC_ClearInterruptStatus(SINC_Type *base, uint64_t statusMasks)

Clear selected mask of interrupt status flags.

Parameters

- base – SINC peripheral base address.
- statusMasks – The mask of interrupt status flags to clear, should be the OR'ed value of sinc_interrupt_status_t.

FSL_SINC_DRIVER_VERSION

lower_component_name driver version 2.1.5.

enum _sinc_interrupt_enable

The enumeration of SINC module's interrupts. .

Values:

enumerator kSINC_CH0ConvCompleteIntEnable

enumerator kSINC_CH1ConvCompleteIntEnable

enumerator kSINC_CH2ConvCompleteIntEnable

enumerator kSINC_CH3ConvCompleteIntEnable

enumerator kSINC_CH0DataReadyIntEnable

enumerator kSINC_CH1DataReadyIntEnable

enumerator kSINC_CH2DataReadyIntEnable

enumerator kSINC_CH3DataReadyIntEnable

enumerator kSINC_CH0ZeroCrossDetectedIntEnable

enumerator kSINC_CH1ZeroCrossDetectedIntEnable

enumerator kSINC_CH2ZeroCrossDetectedIntEnable

enumerator kSINC_CH3ZeroCrossDetectedIntEnable

enumerator kSINC_CH0SCDIntEnable

enumerator kSINC_CH1SCDIntEnable

enumerator kSINC_CH2SCDIntEnable

enumerator kSINC_CH3SCDIntEnable

enumerator kSINC_CH0WindowLimitIntEnable

enumerator kSINC_CH1WindowLimitIntEnable

enumerator kSINC_CH2WindowLimitIntEnable

enumerator kSINC_CH3WindowLimitIntEnable

enumerator kSINC_CH0LowLimitIntEnable

enumerator kSINC_CH1LowLimitIntEnable

enumerator kSINC_CH2LowLimitIntEnable

enumerator kSINC_CH3LowLimitIntEnable

enumerator kSINC_CH0HighLimitIntEnable

enumerator kSINC_CH1HighLimitIntEnable

enumerator kSINC_CH2HighLimitIntEnable

enumerator kSINC_CH3HighLimitIntEnable

enumerator kSINC_CH0FifoUnderflowIntEnable

enumerator kSINC_CH1FifoUnderflowIntEnable

enumerator kSINC_CH2FifoUnderflowIntEnable

enumerator kSINC_CH3FifoUnderflowIntEnable

enumerator kSINC_CH0FifoOverflowIntEnable

enumerator kSINC_CH1FifoOverflowIntEnable

enumerator kSINC_CH2FifoOverflowIntEnable

enumerator kSINC_CH3FifoOverflowIntEnable

enumerator kSINC_CH0CADIntEnable

enumerator kSINC_CH1CADIntEnable

enumerator kSINC_CH2CADIntEnable

enumerator kSINC_CH3CADIntEnable

enumerator kSINC_CH0SaturationIntEnable

enumerator kSINC_CH1SaturationIntEnable

enumerator kSINC_CH2SaturationIntEnable

enumerator kSINC_CH3SaturationIntEnable

enum _sinc_interrupt_status

The enumeration of SINC interrupt status flags. .

Values:

enumerator kSINC_CH0ConvCompleteIntStatus

enumerator kSINC_CH1ConvCompleteIntStatus

enumerator kSINC_CH2ConvCompleteIntStatus

enumerator kSINC_CH3ConvCompleteIntStatus

enumerator kSINC_CH0DataReadyIntStatus

enumerator kSINC_CH1DataReadyIntStatus

enumerator kSINC_CH2DataReadyIntStatus

enumerator kSINC_CH3DataReadyIntStatus

enumerator kSINC_CH0ZeroCrossDetectedIntStatus

enumerator kSINC_CH1ZeroCrossDetectedIntStatus

enumerator kSINC_CH2ZeroCrossDetectedIntStatus

enumerator kSINC_CH3ZeroCrossDetectedIntStatus

enumerator kSINC_CH0SCDIntStatus

enumerator kSINC_CH1SCDIntStatus

enumerator kSINC_CH2SCDIntStatus

enumerator kSINC_CH3SCDIntStatus

enumerator kSINC_CH0WindowLimitIntStatus

enumerator kSINC_CH1WindowLimitIntStatus

enumerator kSINC_CH2WindowLimitIntStatus

enumerator kSINC_CH3WindowLimitIntStatus

enumerator kSINC_CH0LowLimitIntStatus

enumerator kSINC_CH1LowLimitIntStatus

enumerator kSINC_CH2LowLimitIntStatus

enumerator kSINC_CH3LowLimitIntStatus

enumerator kSINC_CH0HighLimitIntStatus

enumerator kSINC_CH1HighLimitIntStatus

enumerator kSINC_CH2HighLimitIntStatus

enumerator kSINC_CH3HighLimitIntStatus

enumerator kSINC_CH0FifoUnderflowIntStatus

enumerator kSINC_CH1FifoUnderflowIntStatus

enumerator kSINC_CH2FifoUnderflowIntStatus

enumerator kSINC_CH3FifoUnderflowIntStatus

enumerator kSINC_CH0FifoOverflowIntStatus

enumerator kSINC_CH1FifoOverflowIntStatus

enumerator kSINC_CH2FifoOverflowIntStatus

enumerator kSINC_CH3FifoOverflowIntStatus

enumerator kSINC_CH0CADIntStatus

enumerator kSINC_CH1CADIntStatus

enumerator kSINC_CH2CADIntStatus

enumerator kSINC_CH3CADIntStatus

enumerator kSINC_CH0SaturationIntStatus

enumerator kSINC_CH1SaturationIntStatus

enumerator kSINC_CH2SaturationIntStatus

enumerator kSINC_CH3SaturationIntStatus

enum _sinc_channel_id

The enumeration of channel id, the sinc module contains 4 channels.

Values:

enumerator kSINC_Channel0

Channel 0.

enumerator kSINC_Channel1

Channel 1.

enumerator kSINC_Channel2

Channel 2.

enumerator kSINC_Channel3

Channel 3.

enum _sinc_modulator_clock

The enumeration of modulator clock name. .

Values:

enumerator kSINC_ModClk0

Modulator Clock 0 output.

enumerator kSINC_ModClk1

Modulator Clock 1 output.

enumerator kSINC_ModClk2

Modulator Clock 2 output.

enum _sinc_inputClk_source

The enumeration of input clock.

Values:

enumerator kSINC_InputClk_SourceMclkOut0

MCLK_OUT0 with internal routeback.

enumerator kSINC_InputClk_SourceMclkOut1

MCLK_OUT1 with internal routeback.

enumerator kSINC_InputClk_SourceMclkOut2

MCLK_OUT2 with internal routeback.

enumerator kSINC_InputClk_SourceExternalModulatorClk

External modulator clock dedicated to the selected channel.

enumerator kSINC_InputClk_SourceAdjacentChannelClk

Grouped clock shared with an adjacent channel.

enum _sinc_inputClk_edge

The enumeration of clock edge.

Values:

enumerator kSINC_InputClk_EdgePositive

Positive edge.

enumerator kSINC_InputClk_EdgeNegative

Negative edge.

enumerator kSINC_InputClk_EdgeBoth
Both edges.

enumerator kSINC_InputClk_EdgeOddPositive
Every other odd positive edge.

enumerator kSINC_InputClk_EdgeEvenPositive
Every other even positive edge.

enumerator kSINC_InputClk_EdgeOddNegative
Every other odd negative edge.

enumerator kSINC_InputClk_EdgeEvenNegative
Every other even negative edge.

enum _sinc_inputBit_format
The enumeration of input bit format.

Values:

enumerator kSINC_InputBit_FormatExternalBitstream
External bitstream from the MBIT[n] signal.

enumerator kSINC_InputBit_FormatExternalManchesterCode
External Manchester code.

enumerator kSINC_InputBit_FormatInternal16bitParallelData
Internal 16-bit parallel data from MPDATA register.

enumerator kSINC_InputBit_FormatInternal32bitSerialData
Internal 32-bit serial data from MPDATA.

enum _sinc_inputBit_source
The enumeration of input bit source.

Values:

enumerator kSINC_InputBit_SourceExternalBitstream
External bitstream from the MBIT[n] signal.

enumerator kSINC_InputBit_SourceInternalBitstream
Alternate internal bitstream from the INP[n] signal.

enumerator kSINC_InputBit_SourceAdjacentChannel
Grouped bitstream shared with an adjacent chanel.

enum _sinc_conv_trigger_source
The enumeration of trigger source.

Values:

enumerator kSINC_ConvTrig_SoftPosEdge
Positive edge software trigger.

enumerator kSINC_ConvTrig_SoftHighLevel
High level software trigger.

enumerator kSINC_ConvTrig_HardPosEdge
Positive edge hardware trigger.

enumerator kSINC_ConvTrig_HardHighLevel
High level hardware trigger.

enumerator kSINC_ConvTrig_AdjacentChannel
Grouped hardware trigger shared with an adjacent channel.

enum _sinc_conv_mode
The enumeration of conversion mode.

Values:

enumerator kSINC_ConvMode_Single
One conversion that follows an edge or level trigger event.

enumerator kSINC_ConvMode_Continuous
Multiple conversions that follow a triggering event, a new triggering event cancels and restarts conversion.

enumerator kSINC_ConvMode_Always
Multiple conversions that follow the first triggering event, SINC ignores the next triggering event.

enumerator kSINC_ConvMode_FixedNumber
Fixed number conversions that follow the first triggering event, a new triggering event cancels and restarts conversion.

enum _sinc_pulse_trigger_mux
The enumeration of pulse trigger mux.

Values:

enumerator kSINC_PulseTrigger_Disabled
Disable pulse trigger output.

enumerator kSINC_PulseTrigger_MuxHighLimitLevelSignal
Select high limit level signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxLowLimitLevelSignal
Select low limit level signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxHighLowLimitLevelSignal
Select low or high limit level signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxWindowLimitLevelSignal
Select window limit level signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxZeroCrossRisingLevelSignal
Select zero cross rising level signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxZeroCrossFallingLevelSignal
Select zero cross falling level signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxRsLimHighLevelSignal
Select level signal that indicates a high level from an RS flip-flop or a schmitt trigger for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxRsLimLowLevelSignal
Select level signal that indicates a low level from an RS flip-flop or a schmitt trigger for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxChannelRawInputModBitStream
Select channel raw input modulator bitstream for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxChannelRawInputModClock
Select channel raw input modulator clock for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxChannelRecoveredModBitStream
Select channel output recovered modulator bitstream for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxChannelRecoveredModClock
Select channel output recovered modulator clock for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxHighLimitPulseSignal
Select high limit pulse signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxLowLimitPulseSignal
Select low limit pulse signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxLimitPulseSignal
Select the pulse signal that indicates a high/low/window limit for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxWindowLimitPulseSignal
Select window limit pulse signal for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxHighLowLimitPulseSignal
Select the pulse signal that indicates a high or low limit for pulse trigger output.

enumerator kSINC_PulseTrigger_MuxZeroCrossRisePulseSignal
Select zero cross rise pulse signal for trigger output.

enumerator kSINC_PulseTrigger_MuxZeroCrossFallPulseSignal
Select zero cross fall pulse signal for trigger output.

enumerator kSINC_PulseTrigger_MuxZeroCrossRiseFallPulseSignal
Select zero cross rise/fall pulse signal for trigger output.

enumerator kSINC_PulseTrigger_MuxFifoWatermarkOkPulseSignal
Select FIFO watermark OK pulse signal for trigger output.

enumerator kSINC_PulseTrigger_MuxFifoOverflowPulseSignal
Select FIFO overflow pulse signal.

enumerator kSINC_PulseTrigger_MuxFifoUnderflowPulseSignal
Select FIFO underflow pulse signal.

enumerator kSINC_PulseTrigger_MuxFifoEmptyPulseSignal
Select FIFO empty pulse signal.

enumerator kSINC_PulseTrigger_MuxClockMonitorAssertPulseSignal
Select clock monitor assert pulse signal.

enumerator kSINC_PulseTrigger_MuxShortCircuitAssertPulseSignal
Select short circuit assert pulse signal.

enumerator kSINC_PulseTrigger_MuxSaturationPulseSignal
Select saturation pulse signal.

enumerator kSINC_PulseTrigger_MuxConversionCompletePulseSignal
Select conversion complete pulse signal.

enum _sinc_zero_cross_operate_mode
The enumeration of zero cross detector operate mode.

Values:

enumerator kSINC_ZCD_BothRiseAndFall
Zero cross detector operate on both rise and fall.

enumerator kSINC_ZCD_OnlyFall
Zero cross detector operate on fall edge.

enumerator kSINC_ZCD_OnlyRise
Zero cross detector operate on rise edge.

enumerator kSINC_ZCD_Disabled
Zero cross detector disabled.

enum _sinc_primary_filter_order
The enumeration of primary filter order.

Values:

enumerator kSINC_PF_FastSinc
Fast sinc filter, ORD is 4.

enumerator kSINC_PF_FirstOrder
First order filter, ORD is 1.

enumerator kSINC_PF_SecondOrder
Second order filter, ORD is 2.

enumerator kSINC_PF_ThirdOrder
Third order filter, ORD is 3.

enum _sinc_clock_prescale
The enumeration of clock prescale that specify the clock divider ratio for the modulator clock.

Values:

enumerator kSINC_ClkPrescale1
No prescale.

enumerator kSINC_ClkPrescale2
Modulator clock divider ratio is 2.

enumerator kSINC_ClkPrescale4
Modulator clock divider ratio is 4.

enumerator kSINC_ClkPrescale8
Modulator clock divider ratio is 8.

enum _sinc_primary_filter_shift_direction
The enumeration of primary filter shift direction.

Values:

enumerator kSINC_PF_ShiftRight
Right shift the raw data.

enumerator kSINC_PF_ShiftLeft
Left shift the raw data.

enum _sinc_primary_filter_bias_sign
The enumeration of primary filer bias sign.

Values:

enumerator kSINC_PF_BiasPositive
The bias sign is positive.

enumerator kSINC_PF_BiasNegative
The bias sign is negative.

enum _sinc_primary_filter_hpf_alpha_coeff
The enumeration of HPF DC remover Alpha coefficient.

Values:

enumerator kSINC_PF_HPFAlphaCoeff0
Disabled HPF.

enumerator kSINC_PF_HPFAlphaCoeff1
Alpha coefficient = 1 - (2^-5)

enumerator kSINC_PF_HPFAlphaCoeff2
Alpha coefficient = 1 - (2^-6)

enumerator kSINC_PF_HPFAlphaCoeff3
Alpha coefficient = 1 - (2^-7)

enumerator kSINC_PF_HPFAlphaCoeff4
Alpha coefficient = 1 - (2^-8)

enumerator kSINC_PF_HPFAlphaCoeff5
Alpha coefficient = 1 - (2^-9)

enumerator kSINC_PF_HPFAlphaCoeff6
Alpha coefficient = 1 - (2^-10)

enumerator kSINC_PF_HPFAlphaCoeff7
Alpha coefficient = 1 - (2^-11)

enumerator kSINC_PF_HPFAlphaCoeff8
Alpha coefficient = 1 - (2^-12)

enumerator kSINC_PF_HPFAlphaCoeff9
Alpha coefficient = 1 - (2^-13)

enumerator kSINC_PF_HPFAlphaCoeff10
Alpha coefficient = 1 - (2^-14)

enumerator kSINC_PF_HPFAlphaCoeff11
Alpha coefficient = 1 - (2^-15)

enumerator kSINC_PF_HPFAlphaCoeff12
Alpha coefficient = 1 - (2^-16)

enumerator kSINC_PF_HPFAlphaCoeff13
Alpha coefficient = 1 - (2^-17)

enumerator kSINC_PF_HPFAlphaCoeff14
Alpha coefficient = 1 - (2^-18)

enumerator kSINC_PF_HPFAlphaCoeff15
Alpha coefficient = 1 - (2^-19)

enum _sinc_inputBit_delay
The enumeration of input bit delay.

Values:

enumerator kSINC_InputBit_DelayDisabled
Input modulator bitstream delay disabled.

enumerator kSINC_InputBit_Delay1ClkCycle
Input modulator bitstream delay 1 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay2ClkCycle
Input modulator bitstream delay 2 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay3ClkCycle
Input modulator bitstream delay 3 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay4ClkCycle
Input modulator bitstream delay 4 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay5ClkCycle
Input modulator bitstream delay 5 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay6ClkCycle
Input modulator bitstream delay 6 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay7ClkCycle
Input modulator bitstream delay 7 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay8ClkCycle
Input modulator bitstream delay 8 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay9ClkCycle
Input modulator bitstream delay 9 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay10ClkCycle
Input modulator bitstream delay 10 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay11ClkCycle
Input modulator bitstream delay 11 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay12ClkCycle
Input modulator bitstream delay 12 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay13ClkCycle
Input modulator bitstream delay 13 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay14ClkCycle
Input modulator bitstream delay 14 PRE_CLK cycle.

enumerator kSINC_InputBit_Delay15ClkCycle
Input modulator bitstream delay 15 PRE_CLK cycle.

enum _sinc_scd_operate_mode
The enumeration of short-circuit detector operate mode.

Values:

enumerator kSINC_Scd_OperateAtChannelEnabled
SCD operates when the channel is enabled.

enumerator kSINC_Scd_OperateAtConversion
SCD operates when the PF is performing a conversion.

enumerator kSINC_Scd_OperateDisabled
Short circuit detect is disabled.

enum _sinc_scd_option
The enumeration of short-circuit detector option.

Values:

enumerator kSINC_Scd_DetectRepeating0And1

Both repeating 0 and 1 increment the SCD counter.

enumerator kSINC_Scd_DetectRepeatingOnly1

Only repeating 1 increment the SCD counter.

enumerator kSINC_Scd_DetectRepeatingOnly0

Only repeating 1 increment the SCD counter.

enum _sinc_limit_detector_mode

The mode of limit detector.

Note: The value of each limit detector contains lot of information: bit[1:0]: limit detection options, bit[2]: low limit break signal. bit[3]: window limit break signal. bit[4]: High limit break signal. bit[7]: Enable/disable limit detector.

Values:

enumerator kSINC_Lmt_BothHighAndLowLimit

Limit detector is enabled, and compare the filter sample value to high and low limit, if the value larger than high limit will trigger interrupt or break, and if the value lower than low limit will trigger interrupt or break.

enumerator kSINC_Lmt_OnlyHighLimit

Limit detector is enabled, and compare the filter sample value to high limit, if the value larger than high limit will trigger interrupt or break.

enumerator kSINC_Lmt_OnlyLowLimit

Limit detector is enabled, and compare the filter sample value to low limit, if the value lower than low limit will trigger interrupt or break.

enumerator kSINC_Lmt_WindowedValue

Limit detector is enabled, and compare the filter sample value to high and low limit, if the value higher than low limit and lower than high limit will trigger interrupt or break.

enumerator kSINC_Lmt_Disabled

Limit detector is disabled.

enum _sinc_cad_threshold

The enumeration of clock-absence threshold.

Values:

enumerator kSINC_Cad_Disabled

Clock absence detector is disabled.

enumerator kSINC_Cad_Count1ClkCycle

Clock absence detector threshold is 1 clock cycle.

enumerator kSINC_Cad_Count2ClkCycle

Clock absence detector threshold is 2 clock cycle.

enumerator kSINC_Cad_Count3ClkCycle

Clock absence detector threshold is 3 clock cycle.

enumerator kSINC_Cad_Count4ClkCycle

Clock absence detector threshold is 4 clock cycle.

enumerator kSINC_Cad_Count5ClkCycle

Clock absence detector threshold is 5 clock cycle.

enumerator kSINC_Cad_Count6ClkCycle
Clock absence detector threshold is 6 clock cycle.

enumerator kSINC_Cad_Count7ClkCycle
Clock absence detector threshold is 7 clock cycle.

enumerator kSINC_Cad_Count8ClkCycle
Clock absence detector threshold is 8 clock cycle.

enumerator kSINC_Cad_Count9ClkCycle
Clock absence detector threshold is 9 clock cycle.

enumerator kSINC_Cad_Count10ClkCycle
Clock absence detector threshold is 10 clock cycle.

enumerator kSINC_Cad_Count11ClkCycle
Clock absence detector threshold is 11 clock cycle.

enumerator kSINC_Cad_Count12ClkCycle
Clock absence detector threshold is 12 clock cycle.

enumerator kSINC_Cad_Count13ClkCycle
Clock absence detector threshold is 13 clock cycle.

enumerator kSINC_Cad_Count14ClkCycle
Clock absence detector threshold is 14 clock cycle.

enumerator kSINC_Cad_Count15ClkCycle
Clock absence detector threshold is 15 clock cycle.

enum _sinc_alternate_dma_source
The enumeration of alternate DMA source.

Values:

enumerator kSINC_AltDma_Disabled
Alternate DMA disabled.

enumerator kSINC_AltDma_PfConvComplete
Select PF conversion complete for alternate DMA.

enumerator kSINC_AltDma_PfDataOutputReady
Select PF data output ready for alternate DMA.

enumerator kSINC_AltDma_ZeroCrossDetected
Select zero crossing detected for alternate DMA.

enumerator kSINC_AltDma_ShortCircuitDetected
Select short circuit detected for alternate DMA.

enumerator kSINC_AltDma_WindowLimitDetected
Select window limit detected for alternate DMA.

enumerator kSINC_AltDma_LowLimitDetected
Select low limit detected for alternate DMA.

enumerator kSINC_AltDma_HighLimitDetected
Select high limit detected for alternate DMA.

enumerator kSINC_AltDma_FifoUnderflow
Select FIFO underflow detected for alternate DMA.

enumerator kSINC_AltDma_FifoOverflow
Select FIFO overflow detected for alternate DMA.

enumerator kSINC_AltDma_ClockAbsence
Select clock absence detected for alternate DMA.

enumerator kSINC_AltDma_Saturation
Select channel saturation for alternate DMA.

enum _sinc_result_data_format
The enumeration of result data format.

Values:

enumerator kSINC_LeftJustifiedSigned
Left justified, signed.

enumerator kSINC_LeftJustifiedUnsigned
Left justified, unsigned.

enum _sinc_debug_output
The enumeration of debug output.

Values:

enumerator kSINC_Debug_PfFinalData
Final data from PF(24 bits).

enumerator kSINC_Debug_OffsetData
Offset data(24 bits).

enumerator kSINC_Debug_PfShiftedData
Shifted data from the PF(24 bits).

enumerator kSINC_Debug_HpfData
DC remover(HPF) data(32 bits).

enumerator kSINC_Debug_CicRawData
Raw data from the PF's CIC filter.

enumerator kSINC_Debug_ScdHistoricalData
Historical data from SCD.

enumerator kSINC_Debug_ManchesterDecoderData
Data from the Manchester decoder.

enumerator kSINC_Debug_CadData
Data from CAD.

enumerator kSINC_Debug_FifoEntriesNum
Number of available entries in the FIFO.

enumerator kSINC_Debug_ParallelSerialConverterStatus
Status of the parallel or serial data converter.

typedef enum *_sinc_channel_id* sinc_channel_id_t
The enumeration of channel id, the sinc module contains 4 channels.

typedef enum *_sinc_inputClk_source* sinc_inputClk_source_t
The enumeration of input clock.

typedef enum *_sinc_inputClk_edge* sinc_inputClk_edge_t
The enumeration of clock edge.

typedef enum *_sinc_inputBit_format* sinc_inputBit_format_t
The enumeration of input bit format.

typedef enum *_sinc_inputBit_source* sinc_inputBit_source_t
The enumeration of input bit source.

typedef enum *_sinc_conv_trigger_source* sinc_conv_trigger_source_t
The enumeration of trigger source.

typedef enum *_sinc_conv_mode* sinc_conv_mode_t
The enumeration of conversion mode.

typedef enum *_sinc_pulse_trigger_mux* sinc_pulse_trigger_mux_t
The enumeration of pulse trigger mux.

typedef enum *_sinc_zero_cross_operate_mode* sinc_zero_cross_operate_mode_t
The enumeration of zero cross detector operate mode.

typedef enum *_sinc_primary_filter_order* sinc_primary_filter_order_t
The enumeration of primary filter order.

typedef enum *_sinc_clock_prescale* sinc_clock_prescale_t
The enumeration of clock prescale that specify the clock divider ratio for the modulator clock.

typedef enum *_sinc_primary_filter_shift_direction* sinc_primary_filter_shift_direction_t
The enumeration of primary filter shift direction.

typedef enum *_sinc_primary_filter_bias_sign* sinc_primary_filter_bias_sign_t
The enumeration of primary filer bias sign.

typedef enum *_sinc_primary_filter_hpf_alpha_coeff* sin_primary_filter_hpf_alpha_coeff_t
The enumeration of HPF DC remover Alpha coefficient.

typedef enum *_sinc_inputBit_delay* sinc_inputBit_delay_t
The enumeration of input bit delay.

typedef enum *_sinc_scd_operate_mode* sinc_scd_operate_mode_t
The enumeration of short-circuit detector operate mode.

typedef enum *_sinc_scd_option* sinc_scd_option_t
The enumeration of short-circuit detector option.

typedef enum *_sinc_limit_detector_mode* sinc_limit_detector_mode_t
The mode of limit detector.

Note: The value of each limit detector contains lot of information: bit[1:0]: limit detection options, bit[2]: low limit break signal. bit[3]: window limit break signal. bit[4]: High limit break signal. bit[7]: Enable/disable limit detector.

---

typedef enum *_sinc_cad_threshold* sinc_cad_threshold_t
The enumeration of clock-absence threshold.

typedef enum *_sinc_alternate_dma_source* sinc_alternate_dma_source_t
The enumeration of alternate DMA source.

typedef enum *_sinc_result_data_format* sinc_result_data_format_t
The enumeration of result data format.

typedef enum *_sinc_debug_output* sinc_debug_output_t
The enumeration of debug output.

typedef struct *_sinc_channel_input_option* sinc_channel_input_option_t

The structure of channel input options, including input bit settings and input clock settings.

typedef struct *_sinc_channel_conv_option* sinc_channel_conv_option_t

The structure of channel conversion options, including CIC filter settings, HPF settings, shift settings, bias settings and so on.

typedef struct *_sinc_channel_protection_option* sinc_channel_protection_option_t

The structure of channel protection options, including limit check settings, short-circuit settings, clock-absence settings, and zero-crossing settings.

typedef struct *_sinc_channel_config* sinc_channel_config_t

The structure of channel configurations, including channel input option, channel conversion options, channel protection options, and so on.

typedef struct *_sinc_config* sinc_config_t

The structure of sinc configurations, including clock settings anc channels' settings.

SINC_CHANNEL_COUNT

SINC_NORMAL_INT_REG_ID

SINC_NORMAL_INT_NAME_COCIE

SINC_NORMAL_INT_NAME_CHFIE

SINC_NORMAL_INT_NAME_ZCDIE

SINC_ERROR_INT_REG_ID

SINC_ERROR_INT_NAME_SCDIE

SINC_ERROR_INT_NAME_WLMTIE

SINC_ERROR_INT_NAME_LLMTIE

SINC_ERROR_INT_NAME_HLMTIE

SINC_FIFO_CAD_INT_REG_ID

SINC_FIFO_CAD_INT_FUNFIE

SINC_FIFO_CAD_INT_FOVFIE

SINC_FIFO_CAD_INT_CADIE

SINC_FIFO_CAD_INT_SATIE

SINC_ENCODE_INTERRUPT(regId, name, channelId)

SINC_DECODE_INTERRUPT(interruptMask)

SINC_FIND_INT_FIELD_VALUE(mask, name)

SINC_FIND_STATUS_FIELD_VALUE(statusValue, name)

struct _sinc_channel_input_option

#include <fsl_sinc.h> The structure of channel input options, including input bit settings and input clock settings.

Public Members

sinc_inputBit_format_t inputBitFormat

Specify input bit format, please refer to sinc_inputBit_format_t.

sinc_inputBit_source_t inputBitSource

Specify input bit source, please refer to sinc_inputBit_source_t.

sinc_inputBit_delay_t inputBitDelay

Specify input bit delay, please refer to sinc_inputBit_delay_t.

sinc_inputClk_source_t inputClkSource

Specify input clock source, please refer to sinc_inputClk_source_t.

sinc_inputClk_edge_t inputClkEdge

Specify input clock edge, please refer to sinc_inputClk_edge_t.

struct _sinc_channel_conv_option

#include <fsl_sinc.h> The structure of channel conversion options, including CIC filter settings, HPF settings, shift settings, bias settings and so on.

Public Members

sinc_conv_mode_t convMode

Specify conversion mode, please refer to sinc_conv_mode_t.

sinc_conv_trigger_source_t convTriggerSource

Specify conversion trigger source, please refer to sinc_conv_trigger_source_t.

bool enableChPrimaryFilter

Enable/disable channel's primary filter.

sinc_primary_filter_order_t pfOrder

Specify the order of primary filter, please refer to sinc_primary_filter_order_t.

uint16_t u16pfOverSampleRatio

Control primary filter's OSR, the minimum permissible value is 3, low value produce unpredictable result, the maximum permissible value depend on PF order and the desired data format, if PF order is third order and data format is signed, the maximum OSR value is 1289, if PF order is third order and data format is unsigned, the maximum OSR value is 1624, otherwise the maximum OSR value is 2047. Please note that the OSR for equation is u16pfOverSampleRatio + 1

sin_primary_filter_hpf_alpha_coeff_t pfHpfAlphaCoeff

Specify HPF's alpha coeff, please refer to sin_primary_filter_hpf_alpha_coeff_t.

sinc_primary_filter_shift_direction_t pfShiftDirection

Select shift direction, right or left.

uint8_t u8pfShiftBitsNum

Specify the number of bits to shift the data, ranges from 0 to 15.

sinc_primary_filter_bias_sign_t pfBiasSign

Select bias sign, please refer to sinc_primary_filter_bias_sign_t.

uint32_t u32pfBiasValue

Range from 0 to 0x7FFFFFUL.

struct _sinc_channel_protection_option

#include <fsl_sinc.h> The structure of channel protection options, including limit check settings, short-circuit settings, clock-absence settings, and zero-crossing settings.

Public Members

sinc_limit_detector_mode_t limitDetectorMode
: Specify limit detector mode, please refer to sinc_limit_detector_mode_t.

bool bEnableLmtBreakSignal
: Enable/disable limit break signal, the details of break signal is depended on detector mode.

uint32_t u32LowLimitThreshold
: Specify the low-limit threshold value, range from 0 to 0xFFFFFFUL.

uint32_t u32HighLimitThreshold
: Specify the high-limit threshold value, range from 0 to 0xFFFFFFUL.

sinc_scd_operate_mode_t scdOperateMode
: Enable/disable scd, and set SCD operate timming.

uint8_t u8ScdLimitThreshold
: Range from 2 to 255, 0 and 1 are prohibited.

sinc_scd_option_t scdOption
: Specify SCD options, please refer to sinc_scd_option_t.

bool bEnableScdBreakSignal
: Enable/disable SCD break signal.

sinc_cad_threshold_t cadLimitThreshold
: Specify the threshold value for the CAD counter.

bool bEnableCadBreakSignal
: Enable/disable CAD break signal.

sinc_zero_cross_operate_mode_t zcdOperateMode
: Specify zero cross detector operate mode.

struct _sinc_channel_config
: *#include <fsl_sinc.h>* The structure of channel configurations, including channel input option, channel conversion options, channel protection options, and so on.

Public Members

bool bEnableChannel
: Enable/disable channel.

bool bEnableFifo
: Enable/disable channel's FIFO.

uint8_t u8FifoWaterMark
: Specify the fifo watermark, range from 0 to 15.

bool bEnablePrimaryDma
: Used to enable/disable primary DMA.

sinc_alternate_dma_source_t altDmaSource
: Set channel's alternate DMA source, please refer to sinc_alternate_dma_source_t.

sinc_result_data_format_t dataFormat
: Set channel's result data format, please refer to sinc_result_data_format_t.

sinc_channel_input_option_t *chInputOption
The pointer to sinc_channel_input_option_t that contains channel input options.

sinc_channel_conv_option_t *chConvOption
The pointer to sinc_channel_conv_option_t that contains channel conversion options.

sinc_channel_protection_option_t *chProtectionOption
The pointer to sinc_channel_protection_option_t that contains channel protection options.

struct _sinc_config
#include <fsl_sinc.h> The structure of sinc configurations, including clock settings anc channels' settings.

Public Members

sinc_clock_prescale_t clockPreDivider
Specify modulator clock pre divider, please refer to sinc_clock_prescale_t.

uint32_t modClkDivider
Range from 2 to 256, 0 and 1 are prohibited, to obtain a 50% duty cycle in the MCLK output, write an even value to

bool disableModClk0Output
Disable/enable modulator clock0 output.

bool disableModClk1Output
Disable/enable modulator clock1 output.

bool disableModClk2Output
Disable/enable modulator clock2 output.

sinc_channel_config_t *channelsConfigArray[(1)]
The array that contains 4 elements, and the type of each element is **sinc_channel_config_t** *, channelsConfigArray[0] corresponding to channel0, channelsConfigArray[1] corresponding to channel1, channelsConfigArray[2] corresponding to channel2, channelsConfigArray[3] corresponding to channel3, if some channels are not used, the corresponding elements should be set as NULL.

bool disableDozeMode
Disable/enable SINC module when the chip enters Doze or stop mode.

bool enableMaster
Enable/disable all function blocks of SINC module.

2.115 SPDIF: Sony/Philips Digital Interface

void SPDIF_Init(SPDIF_Type *base, const *spdif_config_t* *config)
Initializes the SPDIF peripheral.

Ungates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by SPDIF_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

- base – SPDIF base pointer
- config – SPDIF configuration structure.

void SPDIF_GetDefaultConfig(*spdif_config_t* *config)

Sets the SPDIF configuration structure to default values.

This API initializes the configuration structure for use in SPDIF_Init. The initialized structure can remain unchanged in SPDIF_Init, or it can be modified before calling SPDIF_Init. This is an example.

```
spdif_config_t config;
SPDIF_GetDefaultConfig(&config);
```

Parameters

- config – pointer to master configuration structure

void SPDIF_Deinit(SPDIF_Type *base)

De-initializes the SPDIF peripheral.

This API gates the SPDIF clock. The SPDIF module can't operate unless SPDIF_Init is called to enable the clock.

Parameters

- base – SPDIF base pointer

uint32_t SPDIF_GetInstance(SPDIF_Type *base)

Get the instance number for SPDIF.

Parameters

- base – SPDIF base pointer.

static inline void SPDIF_TxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Tx.

This function makes Tx FIFO in reset mode.

Parameters

- base – SPDIF base pointer

static inline void SPDIF_RxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Rx.

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

- base – SPDIF base pointer

void SPDIF_TxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Tx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Tx, false means disable.

static inline void SPDIF_RxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Rx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Rx, false means disable.

static inline uint32_t SPDIF_GetStatusFlag(SPDIF_Type *base)

Gets the SPDIF status flag state.

Parameters

- base – SPDIF base pointer

Returns

SPDIF status flag value. Use the _spdif_interrupt_enable_t to get the status value needed.

static inline void SPDIF_ClearStatusFlags(SPDIF_Type *base, uint32_t mask)

Clears the SPDIF status flag state.

Parameters

- base – SPDIF base pointer
- mask – State mask. It can be a combination of the _spdif_interrupt_enable_t member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits:
  - kSPDIF_UChannelReceiveRegisterFull
  - kSPDIF_QChannelReceiveRegisterFull
  - kSPDIF_TxFIFOEmpty
  - kSPDIF_RxFIFOFull

static inline void SPDIF_EnableInterrupts(SPDIF_Type *base, uint32_t mask)

Enables the SPDIF Tx interrupt requests.

Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF_WordStartInterruptEnable
  - kSPDIF_SyncErrorInterruptEnable
  - kSPDIF_FIFOWarningInterruptEnable
  - kSPDIF_FIFORequestInterruptEnable
  - kSPDIF_FIFOErrorInterruptEnable

static inline void SPDIF_DisableInterrupts(SPDIF_Type *base, uint32_t mask)

Disables the SPDIF Tx interrupt requests.

Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF_WordStartInterruptEnable
  - kSPDIF_SyncErrorInterruptEnable
  - kSPDIF_FIFOWarningInterruptEnable
  - kSPDIF_FIFORequestInterruptEnable
  - kSPDIF_FIFOErrorInterruptEnable

static inline void SPDIF_EnableDMA(SPDIF_Type *base, uint32_t mask, bool enable)

Enables/disables the SPDIF DMA requests.

Parameters

- base – SPDIF base pointer
- mask – SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined
  - kSPDIF_RxDMAEnable
  - kSPDIF_TxDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t SPDIF_TxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

static inline uint32_t SPDIF_TxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

static inline uint32_t SPDIF_RxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

static inline uint32_t SPDIF_RxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

void SPDIF_TxSetSampleRate(SPDIF_Type *base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz)

Configures the SPDIF Tx sample rate.

The audio format can be changed at run-time. This function configures the sample rate.

Parameters

- base – SPDIF base pointer.
- sampleRate_Hz – SPDIF sample rate frequency in Hz.
- sourceClockFreq_Hz – SPDIF tx clock source frequency in Hz.

uint32_t SPDIF_GetRxSampleRate(SPDIF_Type *base, uint32_t clockSourceFreq_Hz)

Configures the SPDIF Rx audio format.

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – SPDIF base pointer.
- clockSourceFreq_Hz – SPDIF system clock frequency in hz.

void SPDIF_WriteBlocking(SPDIF_Type *base, 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 – SPDIF base pointer.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SPDIF_WriteLeftData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteRightData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusHigh(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusLow(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

void SPDIF_ReadBlocking(SPDIF_Type *base, 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 – SPDIF base pointer.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SPDIF_ReadLeftData(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadRightData(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadChannelStatusHigh(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadChannelStatusLow(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadQChannel(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadUChannel(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns
Data in SPDIF FIFO.

void SPDIF_TransferTxCreateHandle(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_callback_t* callback, void *userData)

Initializes the SPDIF Tx handle.

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SPDIF base pointer
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SPDIF_TransferRxCreateHandle(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_callback_t* callback, void *userData)

Initializes the SPDIF Rx handle.

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

status_t SPDIF_TransferSendNonBlocking(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- xfer – Pointer to the spdif_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SPDIF_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SPDIF_TransferReceiveNonBlocking(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- xfer – Pointer to the spdif_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SPDIF_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SPDIF_TransferGetSendCount(SPDIF_Type *base, *spdif_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_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 SPDIF_TransferGetReceiveCount(SPDIF_Type *base, *spdif_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_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 SPDIF_TransferAbortSend(SPDIF_Type *base, *spdif_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 – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.

void SPDIF_TransferAbortReceive(SPDIF_Type *base, *spdif_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 – SPDIF base pointer
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.

void SPDIF_TransferTxHandleIRQ(SPDIF_Type *base, *spdif_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure.

void SPDIF_TransferRxHandleIRQ(SPDIF_Type *base, *spdif_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure.

FSL_SPDIF_DRIVER_VERSION

Version 2.0.7

SPDIF return status.

Values:

enumerator kStatus_SPDIF_RxDPLLLocked

SPDIF Rx PLL locked.

enumerator kStatus_SPDIF_TxFIFOError

SPDIF Tx FIFO error.

enumerator kStatus_SPDIF_TxFIFOResync

SPDIF Tx left and right FIFO resync.

enumerator kStatus_SPDIF_RxCnew
SPDIF Rx status channel value updated.

enumerator kStatus_SPDIF_ValidatyNoGood
SPDIF validaty flag not good.

enumerator kStatus_SPDIF_RxIllegalSymbol
SPDIF Rx receive illegal symbol.

enumerator kStatus_SPDIF_RxParityBitError
SPDIF Rx parity bit error.

enumerator kStatus_SPDIF_UChannelOverrun
SPDIF receive U channel overrun.

enumerator kStatus_SPDIF_QChannelOverrun
SPDIF receive Q channel overrun.

enumerator kStatus_SPDIF_UQChannelSync
SPDIF U/Q channel sync found.

enumerator kStatus_SPDIF_UQChannelFrameError
SPDIF U/Q channel frame error.

enumerator kStatus_SPDIF_RxFIFOError
SPDIF Rx FIFO error.

enumerator kStatus_SPDIF_RxFIFOResync
SPDIF Rx left and right FIFO resync.

enumerator kStatus_SPDIF_LockLoss
SPDIF Rx PLL clock lock loss.

enumerator kStatus_SPDIF_TxIdle
SPDIF Tx is idle

enumerator kStatus_SPDIF_RxIdle
SPDIF Rx is idle

enumerator kStatus_SPDIF_QueueFull
SPDIF queue full

enum _spdif_rxfull_select
SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

Values:

enumerator kSPDIF_RxFull1Sample
Rx full at least 1 sample in left and right FIFO

enumerator kSPDIF_RxFull4Samples
Rx full at least 4 sample in left and right FIFO

enumerator kSPDIF_RxFull8Samples
Rx full at least 8 sample in left and right FIFO

enumerator kSPDIF_RxFull16Samples
Rx full at least 16 sample in left and right FIFO

enum _spdif_txempty_select
SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

Values:

enumerator kSPDIF_TxEmpty0Sample
Tx empty at most 0 sample in left and right FIFO

enumerator kSPDIF_TxEmpty4Samples
Tx empty at most 4 sample in left and right FIFO

enumerator kSPDIF_TxEmpty8Samples
Tx empty at most 8 sample in left and right FIFO

enumerator kSPDIF_TxEmpty12Samples
Tx empty at most 12 sample in left and right FIFO

enum _spdif_uchannel_source
SPDIF U channel source.

Values:

enumerator kSPDIF_NoUChannel
No embedded U channel

enumerator kSPDIF_UChannelFromRx
U channel from receiver, it is CD mode

enumerator kSPDIF_UChannelFromTx
U channel from on chip tx

enum _spdif_gain_select
SPDIF clock gain.

Values:

enumerator kSPDIF_GAIN_24
Gain select is 24

enumerator kSPDIF_GAIN_16
Gain select is 16

enumerator kSPDIF_GAIN_12
Gain select is 12

enumerator kSPDIF_GAIN_8
Gain select is 8

enumerator kSPDIF_GAIN_6
Gain select is 6

enumerator kSPDIF_GAIN_4
Gain select is 4

enumerator kSPDIF_GAIN_3
Gain select is 3

enum _spdif_tx_source
SPDIF tx data source.

Values:

enumerator kSPDIF_txFromReceiver
Tx data directly through SPDIF receiver

enumerator kSPDIF_txNormal
Normal operation, data from processor

enum _spdif_validity_config

SPDIF tx data source.

Values:

enumerator kSPDIF_validityFlagAlwaysSet
Outgoing validity flags always set

enumerator kSPDIF_validityFlagAlwaysClear
Outgoing validity flags always clear

The SPDIF interrupt enable flag.

Values:

enumerator kSPDIF_RxDPLLLocked
SPDIF DPLL locked

enumerator kSPDIF_TxFIFOError
Tx FIFO underrun or overrun

enumerator kSPDIF_TxFIFOResync
Tx FIFO left and right channel resync

enumerator kSPDIF_RxControlChannelChange
SPDIF Rx control channel value changed

enumerator kSPDIF_ValidityFlagNoGood
SPDIF validity flag no good

enumerator kSPDIF_RxIllegalSymbol
SPDIF receiver found illegal symbol

enumerator kSPDIF_RxParityBitError
SPDIF receiver found parity bit error

enumerator kSPDIF_UChannelReceiveRegisterFull
SPDIF U channel revceive register full

enumerator kSPDIF_UChannelReceiveRegisterOverrun
SPDIF U channel receive register overrun

enumerator kSPDIF_QChannelReceiveRegisterFull
SPDIF Q channel receive reigster full

enumerator kSPDIF_QChannelReceiveRegisterOverrun
SPDIF Q channel receive register overrun

enumerator kSPDIF_UQChannelSync
SPDIF U/Q channel sync found

enumerator kSPDIF_UQChannelFrameError
SPDIF U/Q channel frame error

enumerator kSPDIF_RxFIFOError
SPDIF Rx FIFO underrun/overrun

enumerator kSPDIF_RxFIFOResync
SPDIF Rx left and right FIFO resync

enumerator kSPDIF_LockLoss
SPDIF receiver loss of lock

enumerator kSPDIF_TxFIFOEmpty
SPDIF Tx FIFO empty

enumerator kSPDIF_RxFIFOFull
SPDIF Rx FIFO full

enumerator kSPDIF_AllInterrupt
all interrupt

The DMA request sources.

Values:

enumerator kSPDIF_RxDMAEnable
Rx FIFO full

enumerator kSPDIF_TxDMAEnable
Tx FIFO empty

typedef enum *_spdif_rxfull_select* spdif_rxfull_select_t
SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

typedef enum *_spdif_txempty_select* spdif_txempty_select_t
SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

typedef enum *_spdif_uchannel_source* spdif_uchannel_source_t
SPDIF U channel source.

typedef enum *_spdif_gain_select* spdif_gain_select_t
SPDIF clock gain.

typedef enum *_spdif_tx_source* spdif_tx_source_t
SPDIF tx data source.

typedef enum *_spdif_validity_config* spdif_validity_config_t
SPDIF tx data source.

typedef struct *_spdif_config* spdif_config_t
SPDIF user configuration structure.

typedef struct *_spdif_transfer* spdif_transfer_t
SPDIF transfer structure.

typedef struct *_spdif_handle* spdif_handle_t

typedef void (*spdif_transfer_callback_t)(SPDIF_Type *base, *spdif_handle_t* *handle, *status_t* status, void *userData)
SPDIF transfer callback prototype.

SPDIF_XFER_QUEUE_SIZE
SPDIF transfer queue size, user can refine it according to use case.

struct _spdif_config
#include <fsl_spdif.h> SPDIF user configuration structure.

Public Members

bool isTxAutoSync
If auto sync mechanism open

bool isRxAutoSync
: If auto sync mechanism open

uint8_t DPLLClkSource
: SPDIF DPLL clock source, range from 0~15, meaning is chip-specific

uint8_t txClkSource
: SPDIF tx clock source, range from 0~7, meaning is chip-specific

spdif_rxfull_select_t rxFullSelect
: SPDIF rx buffer full select

spdif_txempty_select_t txFullSelect
: SPDIF tx buffer empty select

spdif_uchannel_source_t uChannelSrc
: U channel source

spdif_tx_source_t txSource
: SPDIF tx data source

spdif_validity_config_t validityConfig
: Validity flag config

spdif_gain_select_t gain
: Rx receive clock measure gain parameter.

struct __spdif__transfer
: *#include <fsl_spdif.h>* SPDIF transfer structure.

Public Members

uint8_t *data
: Data start address to transfer.

uint8_t *qdata
: Data buffer for Q channel

uint8_t *udata
: Data buffer for C channel

size_t dataSize
: Transfer size.

struct __spdif__handle
: *#include <fsl_spdif.h>* SPDIF handle structure.

Public Members

uint32_t state
: Transfer status

spdif_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

spdif_transfer_t spdifQueue[(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

2.116 SPDIF eDMA Driver

void SPDIF_TransferTxCreateHandleEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaLeftHandle, *edma_handle_t* *dmaRightHandle)

Initializes the SPDIF eDMA handle.

This function initializes the SPDIF master DMA handle, which can be used for other SPDIF master transactional APIs. Usually, for a specified SPDIF instance, call this API once to get the initialized handle.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaLeftHandle – eDMA handle pointer for left channel, this handle shall be static allocated by users.
- dmaRightHandle – eDMA handle pointer for right channel, this handle shall be static allocated by users.

void SPDIF_TransferRxCreateHandleEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaLeftHandle, *edma_handle_t* *dmaRightHandle)

Initializes the SPDIF Rx eDMA handle.

This function initializes the SPDIF slave DMA handle, which can be used for other SPDIF master transactional APIs. Usually, for a specified SPDIF instance, call this API once to get the initialized handle.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaLeftHandle – eDMA handle pointer for left channel, this handle shall be static allocated by users.
- dmaRightHandle – eDMA handle pointer for right channel, this handle shall be static allocated by users.

status_t SPDIF_TransferSendEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_transfer_t* *xfer)

Performs a non-blocking SPDIF transfer using DMA.

Note: This interface returns immediately after the transfer initiates. Call SPDIF_GetTransferStatus to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SPDIF eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SPDIF is busy sending data.

status_t SPDIF_TransferReceiveEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_transfer_t* *xfer)

Performs a non-blocking SPDIF receive using eDMA.

Note: This interface returns immediately after the transfer initiates. Call the SPDIF_GetReceiveRemainingBytes to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

- base – SPDIF base pointer
- handle – SPDIF eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SPDIF eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SPDIF is busy receiving data.

void SPDIF_TransferAbortSendEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle)

Aborts a SPDIF transfer using eDMA.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.

void SPDIF_TransferAbortReceiveEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle)

Aborts a SPDIF receive using eDMA.

Parameters

- base – SPDIF base pointer
- handle – SPDIF eDMA handle pointer.

status_t SPDIF_TransferGetSendCountEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, size_t *count)

Gets byte count sent by SPDIF.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- count – Bytes count sent by SPDIF.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

status_t SPDIF_TransferGetReceiveCountEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, size_t *count)

Gets byte count received by SPDIF.

Parameters

- base – SPDIF base pointer
- handle – SPDIF eDMA handle pointer.
- count – Bytes count received by SPDIF.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

FSL_SPDIF_EDMA_DRIVER_VERSION

Version 2.0.8

typedef struct *_spdif_edma_handle* spdif_edma_handle_t

typedef void (*spdif_edma_callback_t)(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *status_t* status, void *userData)

SPDIF eDMA transfer callback function for finish and error.

typedef struct *_spdif_edma_transfer* spdif_edma_transfer_t

SPDIF transfer structure.

struct _spdif_edma_transfer

#include <fsl_spdif_edma.h> SPDIF transfer structure.

Public Members

uint8_t *leftData

Data start address to transfer.

uint8_t *rightData

Data start address to transfer.

size_t dataSize

Transfer size.

struct _spdif_edma_handle

#include <fsl_spdif_edma.h> SPDIF DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaLeftHandle
DMA handler for SPDIF left channel

edma_handle_t *dmaRightHandle
DMA handler for SPDIF right channel

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint8_t count
The transfer data count in a DMA request

uint32_t state
Internal state for SPDIF eDMA transfer

spdif_edma_callback_t callback
Callback for users while transfer finish or error occurs

void *userData
User callback parameter

edma_tcd_t leftTcd[(4U) + 1U]
TCD pool for eDMA transfer.

edma_tcd_t rightTcd[(4U) + 1U]
TCD pool for eDMA transfer.

spdif_edma_transfer_t spdifQueue[(4U)]
Transfer queue storing queued transfer.

size_t transferSize[(4U)]
Data bytes need to transfer, left and right are the same, so use one

volatile uint8_t queueUser
Index for user to queue transfer.

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

2.117 SRAMC: SRAMC Driver

void SRAMC_GetDefaultConfig(*sramc_config_t* *config)
Gets the default configuration for the SRAMC.

This function initializes the sramc_config_t structure with default values. The default value are:

```
config->turnaroundTime = 1U;
config->addressHoldTime = 1U;
config->addressSetupTime = 4U;
config->ceHoldTime = 1U;
config->ceSetupTime = 1U;
config->advPolarity = kSRAMC_AdvActiveLow;
config->addressMode = kSRAMC_AddressDataNonMux;
config->portSize = kSRAMC_PortSize16Bit;
config->busTimeoutEnable = false;
config->busTimeoutCounter = 0U;
config->prescaler = kSRAMC_Prescaler_4;
```

(continues on next page)

(continued from previous page)

```
config->readEnableHighTime = 9U;
config->readEnableLowTime = 0X10U;
config->writeEnableHighTime = 0xAU;
config->writeEnableLowTime = 0x1FU;
```

Parameters

- config – Pointer to the sramc_config_t structure to be initialized.

void SRAMC_Init(BLK_CTRL_WAKEUPMIX_Type *base, const *sramc_config_t* *config)

Initializes the SRAMC peripheral.

This function configures the SRAMC controller based on the provided configuration.

Parameters

- base – Pointer to the WAKEUP Domain Block Control module instance structure.
- config – Pointer to the SRAMC configuration structure (sramc_config_t).

void SRAMC_Deinit(BLK_CTRL_WAKEUPMIX_Type *base)

De-initializes the SRAMC peripheral.

This function resets the SRAMC configuration registers to their reset values.

Parameters

- base – Pointer to the WAKEUP Domain Block Control module instance structure.

status_t SRAMC_CalculateTimingValue(uint32_t hclkFreqHz, uint32_t time_ns, uint32_t prescalerValue, bool addOneCycle, uint8_t fieldWidth, uint32_t *result)

Calculates the SRAMC register timing field value from nanoseconds.

This helper function converts a time duration specified in nanoseconds (time_ns) into the corresponding register field value based on the SRAMC functional clock frequency (hclkFreqHz) and the selected prescaler (prescalerValue, corresponding to SRAMCR1[PRE] setting 0-3). It accounts for the '+1' logic required by some timing fields (e.g., REL, REH, WEL, WEH, AH, TA) as specified in the reference manual. It also checks if the calculated value fits within the specified bit-width of the target register field.

Parameters

- hclkFreqHz – Frequency of the hclk clock source for SRAMC in Hz.
- time_ns – Desired time duration in nanoseconds.
- prescalerValue – The value written to the SRAMCR1[PRE] field (0=1 cycle, 1=2 cycles, 2=3 cycles, 3=4 cycles granularity).
- addOneCycle – Set to true if the register field represents (N+1) * granularity (e.g., for REL, REH, etc.), false otherwise (e.g., for CES, CEH).
- fieldWidth – The number of bits available for the target register field (e.g., 6 for WEL, 4 for CEH).
- result – Pointer to store the calculated register field value.

Returns

- kStatus_Success: Calculation successful.
- kStatus_InvalidArgument: If hclkFreqHz is 0, or if the calculated value exceeds the capacity of the fieldWidth.

FSL_SRAMC_DRIVER_VERSION
Defines SRAMC driver version.

enum _sramc_address_mode
SRAMC address mode.

Values:

enumerator kSRAMC_AddressDataMux
Address/Data MUX mode (ADMUX)

enumerator kSRAMC_AddressDataNonMux
Address/Data non-MUX mode (Non-ADMUX)

enum _sramc_port_size
SRAMC port size.

Values:

enumerator kSRAMC_PortSize8Bit
8-bit data port size

enumerator kSRAMC_PortSize16Bit
16-bit data port size

enum _sramc_adv_polarity
SRAMC ADV# polarity.

Values:

enumerator kSRAMC_AdvActiveLow
ADV# signal is active low

enumerator kSRAMC_AdvActiveHigh
ADV# signal is active high

enum _sramc_prescaler
SRAMC prescaler.

Values:

enumerator kSRAMC_Prescaler_1
Time granularity is 1 hclk cycle

enumerator kSRAMC_Prescaler_2
Time granularity is 2 hclk cycles

enumerator kSRAMC_Prescaler_3
Time granularity is 3 hclk cycles

enumerator kSRAMC_Prescaler_4
Time granularity is 4 hclk cycles

typedef enum *_sramc_address_mode* sramc_address_mode_t
SRAMC address mode.

typedef enum *_sramc_port_size* sramc_port_size_t
SRAMC port size.

typedef enum *_sramc_adv_polarity* sramc_adv_polarity_t
SRAMC ADV# polarity.

typedef enum *_sramc_prescaler* sramc_prescaler_t
SRAMC prescaler.

typedef struct *_sramc_config* sramc_config_t
SRAMC configuration structure.

struct _sramc_config
#include <fsl_sramc.h> SRAMC configuration structure.

Public Members

uint8_t turnaroundTime
Turnaround time (TA). Actual time is (TA+1) * granularity. Range: 0-15.

uint8_t addressHoldTime
Address hold time (AH). Actual time is (AH+1) * granularity. Range: 0-15.

uint8_t addressSetupTime
Address setup time (AS). Actual time is AS * granularity (min 1 cycle in ADMUX). Range: 0-15.

uint8_t ceHoldTime
Chip Enable hold time (CEH). Actual time is CEH * granularity. Range: 0-15.

uint8_t ceSetupTime
Chip Enable setup time (CES). Actual time is CES * granularity. Range: 0-15.

sramc_adv_polarity_t advPolarity
ADV# polarity (ADVP).

sramc_address_mode_t addressMode
Address mode (AM). ADMUX or Non-ADMUX.

sramc_port_size_t portSize
Port Size (PS). 8-bit or 16-bit.

bool busTimeoutEnable
AHB Bus Timeout Enable (BTOEN).

uint8_t busTimeoutCounter
AHB Bus Timeout Wait Cycle (BTO). Timeout = BTO * 1024 hclk cycles. Range: 0-255.

sramc_prescaler_t prescaler
Prescaler timer (PRE). Sets time granularity for timing parameters.

uint8_t readEnableHighTime
Read Enable high time (REH). Actual time is (REH+1) * granularity. Range: 0-15.

uint8_t readEnableLowTime
Read Enable low time (REL). Actual time is (REL+1) * granularity. Range: 0-63.

uint8_t writeEnableHighTime
Write Enable high time (WEH). Actual time is (WEH+1) * granularity. Range: 0-15.

uint8_t writeEnableLowTime
Write Enable low time (WEL). Actual time is (WEL+1) * granularity. Range: 0-63.

2.118 Sysctr

enum _sysctr_clock_source

System Counter clock source selection.

Values:

enumerator kSYSCTR_BaseFrequency

24MHz base clock from crystal oscillator is used as the clock.

enumerator kSYSCTR_AlternateFrequency

32kHz alternate slow clock from crystal oscillator is used as the clock.

enum _sysctr_compare_frame

System Counter compare frame selection.

Values:

enumerator kSYSCTR_CompareFrame_0

Select compare frame 0.

enumerator kSYSCTR_CompareFrame_1

Select compare frame 1.

enum _sysctr_frequency_modes_table

System Counter frequency modes table selection.

Values:

enumerator kSYSCTR_FrequencyModesTable_0

Select frequency modes table 0.

enumerator kSYSCTR_FrequencyModesTable_1

Select frequency modes table 1.

enumerator kSYSCTR_FrequencyModesTable_2

Select frequency modes table 2.

enum _sysctr_interrupt_enable

List of System Counter interrupt options.

Values:

enumerator kSYSCTR_Compare0InterruptEnable

Compare frame 0 interrupt.

enumerator kSYSCTR_Compare1InterruptEnable

Compare frame 1 interrupt.

enum _sysctr_status_flags

List of System Counter status flags.

Values:

enumerator kSYSCTR_Compare0Flag

Compare frame 0 compare flag.

enumerator kSYSCTR_Compare1Flag

Compare frame 1 compare flag.

enum _sysctr_flags

System Counter status flags.

This provides constants for the System Counter status flags for use in the System Counter functions. The system counter status flags provide information concerning the clock frequency and debug state.

Values:

enumerator kSYSCTR_DebugHalt

enumerator kSYSCTR_FrequencyChangeAck_0

enumerator kSYSCTR_FrequencyChangeAck_1

typedef enum *_sysctr_clock_source* sysctr_clock_source_t

System Counter clock source selection.

typedef enum *_sysctr_compare_frame* sysctr_compare_frame_t

System Counter compare frame selection.

typedef enum *_sysctr_frequency_modes_table* sysctr_frequency_modes_table_t

System Counter frequency modes table selection.

typedef enum *_sysctr_interrupt_enable* sysctr_interrupt_enable_t

List of System Counter interrupt options.

typedef enum *_sysctr_status_flags* sysctr_status_flags_t

List of System Counter status flags.

typedef struct *_sysctr_config* sysctr_config_t

System Counter config structure.

This structure holds the configuration settings for the System Counter peripheral. To initialize this structure to reasonable defaults, call the PWM_GetDefaultConfig() function and pass a pointer to your config structure instance.

FSL_SYSCTR_DRIVER_VERSION

System Counter driver version.

void SYSCTR_Init(SYS_CTR_CONTROL_Type *ctrlBase, SYS_CTR_COMPARE_Type *cmpBase, const *sysctr_config_t* *pConfig)

Initializes a System Counter instance.

This function initializes the System Counter module with user-defined settings. This example shows how to set up the sysctr_config_t parameters and how to call the SYSCTR_Init function by passing in these parameters. Default clock source is 24MHz base clock from crystal oscillator.

```
sysctr_config_t sysctrConfig;
sysctrConfig.enableDebugHalt              = false;
sysctrConfig.enableHardwareFrequencyChange = false;
SYSCTR_Init(SYS_CTR_CONTROL, SYS_CTR_COMPARE, &sysctr_config_t);
```

Parameters

- ctrlBase – System Counter peripheral control model base address.
- cmpBase – System Counter peripheral compare model base address.
- pConfig – Configuration pointer to user's System Counter config structure.

void SYSCTR_Deinit(SYS_CTR_CONTROL_Type *ctrlBase, SYS_CTR_COMPARE_Type *cmpBase)

De-initializes a System Counter instance.

This function disables the System Counter module clock and sets all register values to the reset value.

Parameters

- ctrlBase – System Counter peripheral control model base address.
- cmpBase – System Counter peripheral compare model base address.

void SYSCTR_GetDefaultConfig(*sysctr_config_t* *pConfig)

Fill in the System Counter config struct with the default settings.

The default values are:

```
config->enableDebugHalt = false;
config->enableHardwareFrequencyChange = false;
```

Parameters

- pConfig – Configuration pointer to user's System Counter config structure.

static inline void SYSCTR_EnableInterrupts(SYS_CTR_COMPARE_Type *base, uint32_t mask)

Enables the selected System Counter interrupts.

Parameters

- base – System Counter peripheral compare model base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration sysctr_interrupt_enable_t.

static inline void SYSCTR_DisableInterrupts(SYS_CTR_COMPARE_Type *base, uint32_t mask)

Disable the selected System Counter interrupts.

Parameters

- base – System Counter peripheral compare model base address.
- mask – The interrupts to disable. This is a logical OR of members of the enumeration sysctr_interrupt_enable_t.

static inline uint32_t SYSCTR_GetStatusFlags(SYS_CTR_COMPARE_Type *base)

Get System Counter status flags.

Parameters

- base – System Counter peripheral compare model base address.

Returns

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

static inline void SYSCTR_StartCounter(SYS_CTR_CONTROL_Type *base)

Enables the System Counter counting.

Parameters

- base – System Counter peripheral control model base address.

static inline void SYSCTR_StopCounter(SYS_CTR_CONTROL_Type *base)

Disables the System Counter counting.

Parameters

- base – System Counter peripheral control model base address.

bool enableDebugHalt

true: Debug input causes the System Counter to halt; false: Debug input is ignored

bool enableHardwareFrequencyChange

true: Hardware changes frequency automatically when entering or exiting low power mode; false: Frequency changes are fully under software control

static inline void SYSCTR_SetCounterlValue(SYS_CTR_CONTROL_Type *base, uint64_t value)

Set System Counter count value.

User can set System Counter initial count value before starting counter. Be cautious that user must set count value while operating on the base frequency only. Set count value while running on the alternate frequency may have unpredictable results.

Parameters

- base – System Counter peripheral control model base address.
- value – System Counter count value.

static inline uint64_t SYSCTR_GetCounterlValue(SYS_CTR_READ_Type *base)

Get System Counter current count value.

Parameters

- base – System Counter peripheral read model base address.

Returns

System Counter current count value.

void SYSCTR_SetCounterClockSource(SYS_CTR_CONTROL_Type *base, *sysctr_clock_source_t* clockSource)

Set System Counter clock source.

Parameters

- base – System Counter peripheral control model base address.
- sysctr_clock_source_t – System Counter clock source.

void SYSCTR_EnableCompare(SYS_CTR_COMPARE_Type *base, *sysctr_compare_frame_t* cmpFrame, bool enable)

Enable or disable compare function of specific compare frame.

Parameters

- base – System Counter peripheral control model base address.
- cmpFrame – System Counter compare frame selection.
- enable – true: Enable compare function; false: Disable compare function

void SYSCTR_SetCompareValue(SYS_CTR_CONTROL_Type *ctrlBase, SYS_CTR_COMPARE_Type *cmpBase, *sysctr_compare_frame_t* cmpFrame, uint64_t value)

Set System Counter compare value of specific compare frame.

Be cautious that user must set compare value while operating on the base frequency only. Set compare value while running on the alternate frequency may have unpredictable results.

Parameters

- ctrlBase – System Counter peripheral control model base address.
- cmpBase – System Counter peripheral compare model base address.
- cmpFrame – System Counter compare frame selection.
- value – System Counter compare value.

uint32_t SYSCTR_GetFrequencyModesTableValue(SYS_CTR_CONTROL_Type *base, *sysctr_frequency_modes_table_t* index)

Get specific Frequency Modes Table value.

Parameters

- base – System Counter peripheral control model base address.

- index – Frequency Modes Table index.

Returns
Frequency modes table value.

struct _sysctr_config

#include <fsl_sysctr.h> System Counter config structure.

This structure holds the configuration settings for the System Counter peripheral. To initialize this structure to reasonable defaults, call the PWM_GetDefaultConfig() function and pass a pointer to your config structure instance.

2.119 SYSCTR: System Counter Driver

2.120 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

2.121 TEMPSENSOR: Temperature Sensor Module

FSL_TMPSNS_DRIVER_VERSION

TMPSNS interrupt status enable type, tmpsns_interrupt_status_enable_t.

Values:

enumerator kTEMPSENSOR_HighTempInterruptStatusEnable
High temperature interrupt status enable.

enumerator kTEMPSENSOR_LowTempInterruptStatusEnable
Low temperature interrupt status enable.

enumerator kTEMPSENSOR_PanicTempInterruptStatusEnable
Panic temperature interrupt status enable.

enumerator kTEMPSENSOR_FinishInterruptStatusEnable
Finish interrupt enable.

TMPSNS interrupt status type, tmpsns_interrupt_status_t.

Values:

enumerator kTEMPSENSOR_HighTempInterruptStatus
High temperature interrupt status.

enumerator kTEMPSENSOR_LowTempInterruptStatus
Low temperature interrupt status.

enumerator kTEMPSENSOR_PanicTempInterruptStatus
Panic temperature interrupt status.

enum tmpsns_measure_mode_t
TMPSNS measure mode, tempsensor_measure_mode.

Values:

enumerator kTEMPSENSOR_SingleMode
Single measurement mode.

enumerator kTEMPSENSOR_ContinuousMode
Continuous measurement mode.

enum _tmpsns_alarm_mode
TMPSNS alarm mode.

Values:

enumerator kTEMPMON_HighAlarmMode
The high alarm temperature interrupt mode.

enumerator kTEMPMON_PanicAlarmMode
The panic alarm temperature interrupt mode.

enumerator kTEMPMON_LowAlarmMode
The low alarm temperature interrupt mode.

typedef struct *_tmpsns_config* tmpsns_config_t
TMPSNS temperature structure.

typedef enum *_tmpsns_alarm_mode* tmpsns_alarm_mode_t
TMPSNS alarm mode.

void TMPSNS_Init(TMPSNS_Type *base, const *tmpsns_config_t* *config)
Initializes the TMPSNS module.

Parameters

- base – TMPSNS base pointer
- config – Pointer to configuration structure.

void TMPSNS_Deinit(TMPSNS_Type *base)
Deinitializes the TMPSNS module.

Parameters

- base – TMPSNS base pointer

void TMPSNS_GetDefaultConfig(*tmpsns_config_t* *config)
Gets the default configuration structure.

This function initializes the TMPSNS configuration structure to a default value. The default values are: tempmonConfig->frequency = 0x02U; tempmonConfig->highAlarmTemp = 44U; tempmonConfig->panicAlarmTemp = 90U; tempmonConfig->lowAlarmTemp = 39U;

Parameters

- config – Pointer to a configuration structure.

void TMPSNS_StartMeasure(TMPSNS_Type *base)
start the temperature measurement process.

Parameters

- base – TMPSNS base pointer.

void TMPSNS_StopMeasure(TMPSNS_Type *base)

stop the measurement process.

Parameters

- base – TMPSNS base pointer

float TMPSNS_GetCurrentTemperature(TMPSNS_Type *base)

Get current temperature with the fused temperature calibration data.

Parameters

- base – TMPSNS base pointer

Returns

current temperature with degrees Celsius.

void TMPSNS_SetTempAlarm(TMPSNS_Type *base, int32_t tempVal, *tmpsns_alarm_mode_t* alarmMode)

Set the temperature count (raw sensor output) that will generate an alarm interrupt.

Parameters

- base – TMPSNS base pointer
- tempVal – The alarm temperature with degrees Celsius
- alarmMode – The alarm mode.

void TMPSNS_EnableInterrupt(TMPSNS_Type *base, uint32_t mask)

Enable interrupt status.

Parameters

- base – TMPSNS base pointer
- mask – The interrupts to enable from tmpsns_interrupt_status_enable_t.

void TMPSNS_DisableInterrupt(TMPSNS_Type *base, uint32_t mask)

Disable interrupt status.

Parameters

- base – TMPSNS base pointer
- mask – The interrupts to disable from tmpsns_interrupt_status_enable_t.

static inline uint32_t TMPSNS_GetInterruptFlags(TMPSNS_Type *base)

Get interrupt status flag.

Parameters

- base – TMPSNS base pointer

static inline void TMPSNS_ClearInterruptFlags(TMPSNS_Type *base, uint32_t mask)

Clear interrupt status flag.

Parameters

- base – TMPSNS base pointer
- mask – The interrupts to disable from tmpsns_interrupt_status_t.

struct __tmpsns_config

#include <fsl_tempsensor.h> TMPSNS temperature structure.

Public Members

tmpsns_measure_mode_t measureMode
The temperature measure mode.

uint16_t frequency
The temperature measure frequency.

int32_t highAlarmTemp
The high alarm temperature.

int32_t panicAlarmTemp
The panic alarm temperature.

int32_t lowAlarmTemp
The low alarm temperature.

2.122 TPM: Timer PWM Module

uint32_t TPM_GetInstance(TPM_Type *base)
Gets the instance from the base address.

Parameters

- base – TPM peripheral base address

Returns
The TPM instance

void TPM_Init(TPM_Type *base, const *tpm_config_t* *config)
Ungates the TPM clock and configures the peripheral for basic operation.

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

Parameters

- base – TPM peripheral base address
- config – Pointer to user's TPM config structure.

void TPM_Deinit(TPM_Type *base)
Stops the counter and gates the TPM clock.

Parameters

- base – TPM peripheral base address

void TPM_GetDefaultConfig(*tpm_config_t* *config)
Fill in the TPM config struct with the default settings.

The default values are:

```
config->prescale = kTPM_Prescale_Divide_1;
config->useGlobalTimeBase = false;
config->syncGlobalTimeBase = false;
config->dozeEnable = false;
config->dbgMode = false;
config->enableReloadOnTrigger = false;
config->enableStopOnOverflow = false;
config->enableStartOnTrigger = false;
```

(continues on next page)

(continued from previous page)

```
#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
    config->enablePauseOnTrigger = false;
#endif
    config->triggerSelect = kTPM_Trigger_Select_0;
#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
    config->triggerSource = kTPM_TriggerSource_External;
    config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
#endif
#if defined(FSL_FEATURE_TPM_HAS_POL) && FSL_FEATURE_TPM_HAS_POL
    config->chnlPolarity = 0U;
#endif
```

Parameters

- config – Pointer to user's TPM config structure.

tpm_clock_prescale_t TPM_CalculateCounterClkDiv(TPM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

Calculates the counter clock prescaler.

This function calculates the values for SC[PS].

return Calculated clock prescaler value.

Parameters

- base – TPM peripheral base address
- counterPeriod_Hz – The desired frequency in Hz which corresponding to the time when the counter reaches the mod value
- srcClock_Hz – TPM counter clock in Hz

status_t TPM_SetupPwm(TPM_Type *base, const *tpm_chnl_pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *tpm_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

- base – TPM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)
- numOfChnls – Number of channels to configure, this should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration tpm_pwm_mode_t
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – TPM counter clock in Hz

Returns

kStatus_Success PWM setup successful kStatus_Error PWM setup failed kStatus_Timeout PWM setup timeout when write register CnV or MOD

status_t TPM_UpdatePwmDutycycle(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_pwm_mode_t* currentPwmMode, uint8_t dutyCyclePercent)

Update the duty cycle of an active PWM signal.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number. In combined mode, this represents the channel pair number
- currentPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns

kStatus_Success if the PWM setup was successful, kStatus_Error on failure

void TPM_UpdateChnlEdgeLevelSelect(TPM_Type *base, *tpm_chnl_t* chnlNumber, uint8_t level)

Update the edge level selection for a channel.

Note: When the TPM has PWM pause level select feature (FSL_FEATURE_TPM_HAS_PAUSE_LEVEL_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use TPM_DisableChannel API to close the PWM output.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- level – The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

static inline uint8_t TPM_GetChannelContorlBits(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fileds).

This function disable the channel by clear all mode and level control bits.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns

The contorl bits value. This is the logical OR of members of the enumeration tpm_chnl_control_bit_mask_t.

static inline *status_t* TPM_DisableChannel(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Dsiable the channel.

This function disable the channel by clear all mode and level control bits.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns

kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

static inline *status_t* TPM_EnableChannel(TPM_Type *base, *tpm_chnl_t* chnlNumber, uint8_t control)

Enable the channel according to mode and level configs.

This function enable the channel output according to input mode/level config parameters.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- control – The contorl bits value. This is the logical OR of members of the enumeration tpm_chnl_control_bit_mask_t.

Returns

kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

void TPM_SetupInputCapture(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_input_capture_edge_t* captureMode)

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- captureMode – Specifies which edge to capture

status_t TPM_SetupOutputCompare(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_output_compare_mode_t* compareMode, uint32_t compareValue)

Configures the TPM to generate timed pulses.

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- compareMode – Action to take on the channel output when the compare condition is met
- compareValue – Value to be programmed in the CnV register.

Returns

kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnV

void TPM_SetupDualEdgeCapture(TPM_Type *base, *tpm_chnl_t* chnlPairNumber, const *tpm_dual_edge_capture_param_t* *edgeParam, uint32_t filterValue)

Configures the dual edge capture mode of the TPM.

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

- base – TPM peripheral base address
- chnlPairNumber – The TPM channel pair number; options are 0, 1, 2, 3
- edgeParam – Sets up the dual edge capture function
- filterValue – Filter value, specify 0 to disable filter.

void TPM_SetupQuadDecode(TPM_Type *base, const *tpm_phase_params_t* *phaseAParams, const *tpm_phase_params_t* *phaseBParams, *tpm_quad_decode_mode_t* quadMode)

Configures the parameters and activates the quadrature decode mode.

Parameters

- base – TPM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters
- quadMode – Selects encoding mode used in quadrature decoder mode

static inline void TPM_SetChannelPolarity(TPM_Type *base, *tpm_chnl_t* chnlNumber, bool enable)

Set the input and output polarity of each of the channels.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Set the channel polarity to active high; false: Set the channel polarity to active low;

static inline void TPM_EnableChannelExtTrigger(TPM_Type *base, *tpm_chnl_t* chnlNumber, bool enable)

Enable external trigger input to be used by channel.

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

Note: No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

void TPM_EnableInterrupts(TPM_Type *base, uint32_t mask)

Enables the selected TPM interrupts.

Parameters

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

void TPM_DisableInterrupts(TPM_Type *base, uint32_t mask)

Disables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration tpm_interrupt_enable_t

uint32_t TPM_GetEnabledInterrupts(TPM_Type *base)

Gets the enabled TPM interrupts.

Parameters

- base – TPM peripheral base address

Returns

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

void TPM_RegisterCallBack(TPM_Type *base, *tpm_callback_t* callback)

Register callback.

If channel or overflow interrupt is enabled by the user, then a callback can be registered which will be invoked when the interrupt is triggered.

Parameters

- base – TPM peripheral base address
- callback – Callback function

static inline uint32_t TPM_GetChannelValue(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Gets the TPM channel value.

Note: The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns

The channle CnV regisyer value.

static inline uint32_t TPM_GetStatusFlags(TPM_Type *base)

Gets the TPM status flags.

Parameters

- base – TPM peripheral base address

Returns

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

static inline void TPM_ClearStatusFlags(TPM_Type *base, uint32_t mask)

Clears the TPM status flags.

Parameters

- base – TPM peripheral base address

- mask – The status flags to clear. This is a logical OR of members of the enumeration tpm_status_flags_t

static inline *status_t* TPM_SetTimerPeriod(TPM_Type *base, uint32_t ticks)

Sets the timer period in units of ticks.

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note:

a. This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.

b. Call the utility macros provided in the fsl_common.h to convert usec or msec to ticks.

Parameters

- base – TPM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

Returns

kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

static inline uint32_t TPM_GetCurrentTimerCount(TPM_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – TPM peripheral base address

Returns

The current counter value in ticks

static inline void TPM_StartTimer(TPM_Type *base, *tpm_clock_source_t* clockSource)

Starts the TPM counter.

Parameters

- base – TPM peripheral base address
- clockSource – TPM clock source; once clock source is set the counter will start running

static inline *status_t* TPM_StopTimer(TPM_Type *base)

Stops the TPM counter.

Parameters

- base – TPM peripheral base address

Returns

kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

FSL_TPM_DRIVER_VERSION
TPM driver version 2.4.0.

enum _tpm_chnl
List of TPM channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kTPM_Chnl_0
TPM channel number 0

enumerator kTPM_Chnl_1
TPM channel number 1

enumerator kTPM_Chnl_2
TPM channel number 2

enumerator kTPM_Chnl_3
TPM channel number 3

enumerator kTPM_Chnl_4
TPM channel number 4

enumerator kTPM_Chnl_5
TPM channel number 5

enumerator kTPM_Chnl_6
TPM channel number 6

enumerator kTPM_Chnl_7
TPM channel number 7

enum _tpm_pwm_mode
TPM PWM operation modes.

Values:

enumerator kTPM_EdgeAlignedPwm
Edge aligned PWM

enumerator kTPM_CenterAlignedPwm
Center aligned PWM

enumerator kTPM_CombinedPwm
Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

enum _tpm_pwm_level_select
TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

Values:

enumerator kTPM_HighTrue
High true pulses

enumerator kTPM_LowTrue
Low true pulses

enum _tpm_pwm_pause_level_select
TPM PWM output when first enabled or paused: set or clear.

Values:

enumerator kTPM_ClearOnPause
Clear Output when counter first enabled or paused.

enumerator kTPM_SetOnPause
Set Output when counter first enabled or paused.

enum _tpm_chnl_control_bit_mask
List of TPM channel modes and level control bit mask.

Values:

enumerator kTPM_ChnlELSnAMask
Channel ELSA bit mask.

enumerator kTPM_ChnlELSnBMask
Channel ELSB bit mask.

enumerator kTPM_ChnlMSAMask
Channel MSA bit mask.

enumerator kTPM_ChnlMSBMask
Channel MSB bit mask.

enum _tpm_trigger_select
Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

Values:

enumerator kTPM_Trigger_Select_0

enumerator kTPM_Trigger_Select_1

enumerator kTPM_Trigger_Select_2

enumerator kTPM_Trigger_Select_3

enumerator kTPM_Trigger_Select_4

enumerator kTPM_Trigger_Select_5

enumerator kTPM_Trigger_Select_6

enumerator kTPM_Trigger_Select_7

enumerator kTPM_Trigger_Select_8

enumerator kTPM_Trigger_Select_9

enumerator kTPM_Trigger_Select_10

enumerator kTPM_Trigger_Select_11

enumerator kTPM_Trigger_Select_12

enumerator kTPM_Trigger_Select_13

enumerator kTPM_Trigger_Select_14

enumerator kTPM_Trigger_Select_15

enum _tpm_trigger_source

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

Values:

enumerator kTPM_TriggerSource_External

Use external trigger input

enumerator kTPM_TriggerSource_Internal

Use internal trigger (channel pin input capture)

enum _tpm_ext_trigger_polarity

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

Values:

enumerator kTPM_ExtTrigger_Active_High

External trigger input is active high

enumerator kTPM_ExtTrigger_Active_Low

External trigger input is active low

enum _tpm_output_compare_mode

TPM output compare modes.

Values:

enumerator kTPM_NoOutputSignal

No channel output when counter reaches CnV

enumerator kTPM_ToggleOnMatch

Toggle output

enumerator kTPM_ClearOnMatch

Clear output

enumerator kTPM_SetOnMatch

Set output

enumerator kTPM_HighPulseOutput

Pulse output high

enumerator kTPM_LowPulseOutput

Pulse output low

enum _tpm_input_capture_edge

TPM input capture edge.

Values:

enumerator kTPM_RisingEdge

Capture on rising edge only

enumerator kTPM_FallingEdge

Capture on falling edge only

enumerator kTPM_RiseAndFallEdge

Capture on rising or falling edge

enum _tpm_quad_decode_mode

TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

Values:

enumerator kTPM_QuadPhaseEncode

Phase A and Phase B encoding mode

enumerator kTPM_QuadCountAndDir

Count and direction encoding mode

enum _tpm_phase_polarity

TPM quadrature phase polarities.

Values:

enumerator kTPM_QuadPhaseNormal

Phase input signal is not inverted

enumerator kTPM_QuadPhaseInvert

Phase input signal is inverted

enum _tpm_clock_source

TPM clock source selection.

Values:

enumerator kTPM_SystemClock

System clock

enumerator kTPM_ExternalClock

External TPM_EXTCLK pin clock

enumerator kTPM_ExternalInputTriggerClock

Selected external input trigger clock

enum _tpm_clock_prescale

TPM prescale value selection for the clock source.

Values:

enumerator kTPM_Prescale_Divide_1

Divide by 1

enumerator kTPM_Prescale_Divide_2

Divide by 2

enumerator kTPM_Prescale_Divide_4
Divide by 4

enumerator kTPM_Prescale_Divide_8
Divide by 8

enumerator kTPM_Prescale_Divide_16
Divide by 16

enumerator kTPM_Prescale_Divide_32
Divide by 32

enumerator kTPM_Prescale_Divide_64
Divide by 64

enumerator kTPM_Prescale_Divide_128
Divide by 128

enum _tpm_interrupt_enable
List of TPM interrupts.

Values:

enumerator kTPM_Chnl0InterruptEnable
Channel 0 interrupt.

enumerator kTPM_Chnl1InterruptEnable
Channel 1 interrupt.

enumerator kTPM_Chnl2InterruptEnable
Channel 2 interrupt.

enumerator kTPM_Chnl3InterruptEnable
Channel 3 interrupt.

enumerator kTPM_Chnl4InterruptEnable
Channel 4 interrupt.

enumerator kTPM_Chnl5InterruptEnable
Channel 5 interrupt.

enumerator kTPM_Chnl6InterruptEnable
Channel 6 interrupt.

enumerator kTPM_Chnl7InterruptEnable
Channel 7 interrupt.

enumerator kTPM_TimeOverflowInterruptEnable
Time overflow interrupt.

enum _tpm_status_flags
List of TPM flags.

Values:

enumerator kTPM_Chnl0Flag
Channel 0 flag

enumerator kTPM_Chnl1Flag
Channel 1 flag

enumerator kTPM_Chnl2Flag
Channel 2 flag

enumerator kTPM_Chnl3Flag
Channel 3 flag

enumerator kTPM_Chnl4Flag
Channel 4 flag

enumerator kTPM_Chnl5Flag
Channel 5 flag

enumerator kTPM_Chnl6Flag
Channel 6 flag

enumerator kTPM_Chnl7Flag
Channel 7 flag

enumerator kTPM_TimeOverflowFlag
Time overflow flag

typedef enum *_tpm_chnl* tpm_chnl_t
List of TPM channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_tpm_pwm_mode* tpm_pwm_mode_t
TPM PWM operation modes.

typedef enum *_tpm_pwm_level_select* tpm_pwm_level_select_t
TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

typedef enum *_tpm_pwm_pause_level_select* tpm_pwm_pause_level_select_t
TPM PWM output when first enabled or paused: set or clear.

typedef enum *_tpm_chnl_control_bit_mask* tpm_chnl_control_bit_mask_t
List of TPM channel modes and level control bit mask.

typedef struct *_tpm_chnl_pwm_signal_param* tpm_chnl_pwm_signal_param_t
Options to configure a TPM channel's PWM signal.

typedef enum *_tpm_trigger_select* tpm_trigger_select_t
Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

typedef enum *_tpm_trigger_source* tpm_trigger_source_t
Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

typedef enum *_tpm_ext_trigger_polarity* tpm_ext_trigger_polarity_t

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

typedef enum *_tpm_output_compare_mode* tpm_output_compare_mode_t

TPM output compare modes.

typedef enum *_tpm_input_capture_edge* tpm_input_capture_edge_t

TPM input capture edge.

typedef struct *_tpm_dual_edge_capture_param* tpm_dual_edge_capture_param_t

TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_quad_decode_mode* tpm_quad_decode_mode_t

TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_phase_polarity* tpm_phase_polarity_t

TPM quadrature phase polarities.

typedef struct *_tpm_phase_param* tpm_phase_params_t

TPM quadrature decode phase parameters.

typedef enum *_tpm_clock_source* tpm_clock_source_t

TPM clock source selection.

typedef enum *_tpm_clock_prescale* tpm_clock_prescale_t

TPM prescale value selection for the clock source.

typedef struct *_tpm_config* tpm_config_t

TPM config structure.

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

The config struct can be made const so it resides in flash

typedef enum *_tpm_interrupt_enable* tpm_interrupt_enable_t

List of TPM interrupts.

typedef enum *_tpm_status_flags* tpm_status_flags_t

List of TPM flags.

typedef void (*tpm_callback_t)(TPM_Type *base)

TPM callback function pointer.

Param base
TPM peripheral base address.

static inline void TPM_Reset(TPM_Type *base)

Performs a software reset on the TPM module.

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

Note: TPM software reset is available on certain SoC's only

Parameters

- base – TPM peripheral base address

void TPM_DriverIRQHandler(uint32_t instance)

TPM driver IRQ handler common entry.

This function provides the common IRQ request entry for TPM.

Parameters

- instance – TPM instance.

TPM_TIMEOUT

Max loops to wait for writing register.

When writing MOD CnV CnSC and SC register, driver will wait until register is updated. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

TPM_MAX_COUNTER_VALUE(x)

Help macro to get the max counter value.

struct _tpm_chnl_pwm_signal_param

#include <fsl_tpm.h> Options to configure a TPM channel's PWM signal.

Public Members

tpm_chnl_t chnlNumber

TPM channel to configure. In combined mode (available in some SoC's), this represents the channel pair number

tpm_pwm_pause_level_select_t pauseLevel

PWM output level when counter first enabled or paused

tpm_pwm_level_select_t level

PWM output active level select

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

uint8_t firstEdgeDelayPercent

Used only in combined PWM mode to generate asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greate than 100.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

tpm_pwm_pause_level_select_t secPauseLevel

Used only in combined PWM mode. Define the second channel output level when counter first enabled or paused

uint8_t deadTimeValue[2]

The dead time value for channel n and n+1 in combined complementary PWM mode. Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue * 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

struct __tpm__dual__edge__capture__param

#include <fsl_tpm.h> TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

Public Members

bool enableSwap

true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored

tpm_input_capture_edge_t currChanEdgeMode

Input capture edge select for channel n

tpm_input_capture_edge_t nextChanEdgeMode

Input capture edge select for channel n+1

struct __tpm__phase__param

#include <fsl_tpm.h> TPM quadrature decode phase parameters.

Public Members

uint32_t phaseFilterVal

Filter value, filter is disabled when the value is zero

tpm_phase_polarity_t phasePolarity

Phase polarity

struct __tpm__config

#include <fsl_tpm.h> TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the TPM_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

tpm_clock_prescale_t prescale

Select TPM clock prescale value

bool useGlobalTimeBase

true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase

bool syncGlobalTimeBase

true: The TPM counter is synchronized to the global time base; false: disabled

tpm_trigger_select_t triggerSelect
Input trigger to use for controlling the counter operation

tpm_trigger_source_t triggerSource
Decides if we use external or internal trigger.

tpm_ext_trigger_polarity_t extTriggerPolarity
when using external trigger source, need selects the polarity of it.

bool enableDoze
true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode

bool enableDebugMode
true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode

bool enableReloadOnTrigger
true: TPM counter is reloaded on trigger; false: TPM counter not reloaded

bool enableStopOnOverflow
true: TPM counter stops after overflow; false: TPM counter continues running after overflow

bool enableStartOnTrigger
true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately

bool enablePauseOnTrigger
true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

uint8_t chnlPolarity
Defines the input/output polarity of the channels in POL register

2.123 TRDC: Trusted Resource Domain Controller

void TRDC_Init(TRDC_Type *base)
Initializes the TRDC module.

This function enables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

void TRDC_Deinit(TRDC_Type *base)
De-initializes the TRDC module.

This function disables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

static inline uint8_t TRDC_GetCurrentMasterDomainId(TRDC_Type *base)
Gets the domain ID of the current bus master.

Parameters

- base – TRDC peripheral base address.

Returns
Domain ID of current bus master.

void TRDC_GetHardwareConfig(TRDC_Type *base, *trdc_hardware_config_t* *config)

Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

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

static inline void TRDC_SetDacGlobalValid(TRDC_Type *base)

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline void TRDC_LockMasterDomainAssignment(TRDC_Type *base, uint8_t master)

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.

static inline void TRDC_SetMasterDomainAssignmentValid(TRDC_Type *base, uint8_t master, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.
- valid – True to set valid, false to set invalid.

void TRDC_GetDefaultProcessorDomainAssignment(*trdc_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId         = 0U;
assignment->domainIdSelect   = kTRDC_DidMda;
assignment->lock             = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void TRDC_GetDefaultNonProcessorDomainAssignment(*trdc_non_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId             = 0U;
assignment->privilegeAttr        = kTRDC_ForceUser;
assignment->secureAttr       = kTRDC_ForceSecure;
assignment->bypassDomainId       = 0U;
assignment->lock                 = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, const *trdc_processor_domain_assignment_t* *domainAssignment)

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;

TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the assignment structure.

static inline void TRDC_EnableProcessorDomainAssignment(TRDC_Type *base, bool enable)

Enables the processor bus master domain assignment.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const *trdc_non_processor_domain_assignment_t* *domainAssignment)

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;
```

(continues on next page)

(continued from previous page)

```
TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to trdc_master_t in processor header file.
- domainAssignment – Pointer to the assignment structure.

void TRDC_GetDefaultIDAUConfig(*trdc_idau_config_t* *idauConfiguration)

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```
config->lockSecureVTOR    = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU     = false;
config->lockNonsecureMPU  = false;
config->lockSAU           = false;
```

Parameters

- idauConfiguration – Pointer to the configuration structure.

void TRDC_SetIDAU(TRDC_Type *base, const *trdc_idau_config_t* *idauConfiguration)

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
idauConfiguration.lockNonsecureMPU     = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- idauConfiguration – Pointer to the configuration structure.

static inline void TRDC_EnableFlashLogicalWindow(TRDC_Type *base, bool enable)

Enables/disables the FLW(flash logical window) function.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

static inline void TRDC_LockFlashLogicalWindow(TRDC_Type *base)

Locks FLW registers. Once locked the registers can noy be updated until next reset.

Parameters

- base – TRDC peripheral base address.

static inline uint32_t TRDC_GetFlashLogicalWindowPbase(TRDC_Type *base)

Gets the FLW physical base address.

Parameters

- base – TRDC peripheral base address.

Returns
Physical address of the FLW function.

static inline void TRDC_GetSetFlashLogicalWindowSize(TRDC_Type *base, uint16_t size)

Sets the FLW size.

Parameters

- base – TRDC peripheral base address.
- size – Size of the FLW in unit of 32k bytes.

void TRDC_GetDefaultFlashLogicalWindowConfig(*trdc_flw_config_t* *flwConfiguration)

Gets the default FLW(Flsh Logical Window) configuration.

```
config->blockCount    = false;
config->arrayBaseAddr = false;
config->lock      = false;
config->enable    = false;
```

Parameters

- flwConfiguration – Pointer to the configuration structure.

void TRDC_SetFlashLogicalWindow(TRDC_Type *base, const *trdc_flw_config_t* *flwConfiguration)

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xXXXXXXXX;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- flwConfiguration – Pointer to the configuration structure.

status_t TRDC_GetAndClearFirstDomainError(TRDC_Type *base, *trdc_domain_error_t* *error)

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.

Returns
If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_NoData.

status_t TRDC_GetAndClearFirstSpecificDomainError(TRDC_Type *base, *trdc_domain_error_t* *error, uint8_t domainId)

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.
- domainId – The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_NoData.

static inline void TRDC_SetMrcGlobalValid(TRDC_Type *base)

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline uint8_t TRDC_GetMrcRegionNumber(TRDC_Type *base, uint8_t mrcIdx)

Gets the TRDC MRC(Memory Region Checkers) region number valid.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.

Returns

the region number of the given MRC instance

void TRDC_MrcSetMemoryAccessConfig(TRDC_Type *base, const *trdc_memory_access_control_config_t* *config, uint8_t mrcIdx, uint8_t regIdx)

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mrcIdx – MRC index.
- regIdx – Register number.

void TRDC_MrcEnableDomainNseUpdate(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask, bool enable)

Enables the update of the selected domains.

After the domains' update are enabled, their regions' NSE bits can be set or clear.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

void TRDC_MrcRegionNseSet(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

void TRDC_MrcRegionNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to clear.

void TRDC_MrcDomainNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains whose NSE bits to clear.

void TRDC_MrcSetRegionDescriptorConfig(TRDC_Type *base, const *trdc_mrc_region_descriptor_config_t* *config)

Sets the configuration for one of the region descriptor per domain per MRC instnce.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to region descriptor configuration structure.

static inline void TRDC_SetMbcGlobalValid(TRDC_Type *base)

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

void TRDC_GetMbcHardwareConfig(TRDC_Type *base, *trdc_slave_memory_hardware_config_t* *config, uint8_t mbcIdx, uint8_t slvIdx)

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

void TRDC_MbcSetNseUpdateConfig(TRDC_Type *base, const *trdc_mbc_nse_update_config_t* *config, uint8_t mbcIdx)

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be updateby NSE set/clear.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

void TRDC_MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

void TRDC_MbcWordNseClear(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

void TRDC_MbcNseClearAll(TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slaveMask)

Clears all configuration words' NSE bits of the selected domain and memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

void TRDC_MbcSetMemoryAccessConfig(TRDC_Type *base, const *trdc_memory_access_control_config_t* *config, uint8_t mbcIdx, uint8_t rgdIdx)

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

void TRDC_MbcSetMemoryBlockConfig(TRDC_Type *base, const *trdc_mbc_memory_block_config_t* *config)

Sets the configuration for one of the memory block per domain per MBC instnce.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

enum _trdc_did_sel

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

Values:

enumerator kTRDC_DidMda

Use MDAn[2:0] as DID.

enumerator kTRDC_DidInput

Use the input DID (DID_in) as DID.

enumerator kTRDC_DidMdaAndInput

Use MDAn[2] concatenated with DID_in[1:0] as DID.

enumerator kTRDC_DidReserved

Reserved.

enum _trdc_secure_attr

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

Values:

enumerator kTRDC_ForceSecure

Force the bus attribute for this master to secure.

enumerator kTRDC_ForceNonSecure

Force the bus attribute for this master to non-secure.

enumerator kTRDC_MasterSecure

Use the bus master's secure/nonsecure attribute directly.

enumerator kTRDC_MasterSecure1

Use the bus master's secure/nonsecure attribute directly.

enum _trdc_privilege_attr

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

Values:

enumerator kTRDC_ForceUser

Force the bus attribute for this master to user.

enumerator kTRDC_ForcePrivilege

Force the bus attribute for this master to privileged.

enumerator kTRDC_MasterPrivilege

Use the bus master's attribute directly.

enumerator kTRDC_MasterPrivilege1

Use the bus master's attribute directly.

enum _trdc_controller

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

Values:

enumerator kTRDC_MemBlockController0

Memory block checker 0.

enumerator kTRDC_MemBlockController1

Memory block checker 1.

enumerator kTRDC_MemBlockController2

Memory block checker 2.

enumerator kTRDC_MemBlockController3

Memory block checker 3.

enumerator kTRDC_MemRegionChecker0

Memory region checker 0.

enumerator kTRDC_MemRegionChecker1

Memory region checker 1.

enumerator kTRDC_MemRegionChecker2

Memory region checker 2.

enumerator kTRDC_MemRegionChecker3
Memory region checker 3.

enumerator kTRDC_MemRegionChecker4
Memory region checker 4.

enumerator kTRDC_MemRegionChecker5
Memory region checker 5.

enumerator kTRDC_MemRegionChecker6
Memory region checker 6.

enum _trdc_error_state
TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or TRDC_MRCn_DERR_W1[EST].

Values:

enumerator kTRDC_ErrorStateNone
No access violation detected.

enumerator kTRDC_ErrorStateNone1
No access violation detected.

enumerator kTRDC_ErrorStateSingle
Single access violation detected.

enumerator kTRDC_ErrorStateMulti
Multiple access violation detected.

enum _trdc_error_attr
TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

Values:

enumerator kTRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kTRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kTRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorSecurePrivilegeData
Secure privileged mode, data access.

enumerator kTRDC_ErrorNonSecureUserInst
NonSecure user mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecureUserData
NonSecure user mode, data access.

enumerator kTRDC_ErrorNonSecurePrivilegeInst
NonSecure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecurePrivilegeData
NonSecure privileged mode, data access.

enum _trdc_error_type
TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

Values:

enumerator kTRDC_ErrorTypeRead
Error occurs on read reference.

enumerator kTRDC_ErrorTypeWrite
Error occurs on write reference.

enum _trdc_region_descriptor
The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

Values:

enumerator kTRDC_RegionDescriptor0
Region descriptor 0.

enumerator kTRDC_RegionDescriptor1
Region descriptor 1.

enumerator kTRDC_RegionDescriptor2
Region descriptor 2.

enumerator kTRDC_RegionDescriptor3
Region descriptor 3.

enumerator kTRDC_RegionDescriptor4
Region descriptor 4.

enumerator kTRDC_RegionDescriptor5
Region descriptor 5.

enumerator kTRDC_RegionDescriptor6
Region descriptor 6.

enumerator kTRDC_RegionDescriptor7
Region descriptor 7.

enumerator kTRDC_RegionDescriptor8
Region descriptor 8.

enumerator kTRDC_RegionDescriptor9
Region descriptor 9.

enumerator kTRDC_RegionDescriptor10
Region descriptor 10.

enumerator kTRDC_RegionDescriptor11
Region descriptor 11.

enumerator kTRDC_RegionDescriptor12
Region descriptor 12.

enumerator kTRDC_RegionDescriptor13
Region descriptor 13.

enumerator kTRDC_RegionDescriptor14
Region descriptor 14.

enumerator kTRDC_RegionDescriptor15
Region descriptor 15.

enum _trdc_MRC_domain

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

Values:

enumerator kTRDC_MrcDomain0
Domain 0.

enumerator kTRDC_MrcDomain1
Domain 1.

enumerator kTRDC_MrcDomain2
Domain 2.

enumerator kTRDC_MrcDomain3
Domain 3.

enumerator kTRDC_MrcDomain4
Domain 4.

enumerator kTRDC_MrcDomain5
Domain 5.

enumerator kTRDC_MrcDomain6
Domain 6.

enumerator kTRDC_MrcDomain7
Domain 7.

enumerator kTRDC_MrcDomain8
Domain 8.

enumerator kTRDC_MrcDomain9
Domain 9.

enumerator kTRDC_MrcDomain10
Domain 10.

enumerator kTRDC_MrcDomain11
Domain 11.

enumerator kTRDC_MrcDomain12
Domain 12.

enumerator kTRDC_MrcDomain13
Domain 13.

enumerator kTRDC_MrcDomain14
Domain 14.

enumerator kTRDC_MrcDomain15
Domain 15.

enum _trdc_MBC_domain

The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

Values:

enumerator kTRDC_MbcDomain0
Domain 0.

enumerator kTRDC_MbcDomain1
Domain 1.

enumerator kTRDC_MbcDomain2
Domain 2.

enumerator kTRDC_MbcDomain3
Domain 3.

enumerator kTRDC_MbcDomain4
Domain 4.

enumerator kTRDC_MbcDomain5
Domain 5.

enumerator kTRDC_MbcDomain6
Domain 6.

enumerator kTRDC_MbcDomain7
Domain 7.

enum _trdc_MBC_memory
The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

Values:

enumerator kTRDC_MbcSlaveMemory0
Memory 0.

enumerator kTRDC_MbcSlaveMemory1
Memory 1.

enumerator kTRDC_MbcSlaveMemory2
Memory 2.

enumerator kTRDC_MbcSlaveMemory3
Memory 3.

enum _trdc_MBC_bit
The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

Values:

enumerator kTRDC_MbcBit0
Bit 0.

enumerator kTRDC_MbcBit1
Bit 1.

enumerator kTRDC_MbcBit2
Bit 2.

enumerator kTRDC_MbcBit3
Bit 3.

enumerator kTRDC_MbcBit4
Bit 4.

enumerator kTRDC_MbcBit5
Bit 5.

enumerator kTRDC_MbcBit6
Bit 6.

enumerator kTRDC_MbcBit7
Bit 7.

enumerator kTRDC_MbcBit8
Bit 8.

enumerator kTRDC_MbcBit9
Bit 9.

enumerator kTRDC_MbcBit10
Bit 10.

enumerator kTRDC_MbcBit11
Bit 11.

enumerator kTRDC_MbcBit12
Bit 12.

enumerator kTRDC_MbcBit13
Bit 13.

enumerator kTRDC_MbcBit14
Bit 14.

enumerator kTRDC_MbcBit15
Bit 15.

enumerator kTRDC_MbcBit16
Bit 16.

enumerator kTRDC_MbcBit17
Bit 17.

enumerator kTRDC_MbcBit18
Bit 18.

enumerator kTRDC_MbcBit19
Bit 19.

enumerator kTRDC_MbcBit20
Bit 20.

enumerator kTRDC_MbcBit21
Bit 21.

enumerator kTRDC_MbcBit22
Bit 22.

enumerator kTRDC_MbcBit23
Bit 23.

enumerator kTRDC_MbcBit24
Bit 24.

enumerator kTRDC_MbcBit25
Bit 25.

enumerator kTRDC_MbcBit26
Bit 26.

enumerator kTRDC_MbcBit27
Bit 27.

enumerator kTRDC_MbcBit28
Bit 28.

enumerator kTRDC_MbcBit29
Bit 29.

enumerator kTRDC_MbcBit30
Bit 30.

enumerator kTRDC_MbcBit31
Bit 31.

typedef struct *_trdc_hardware_config* trdc_hardware_config_t
TRDC hardware configuration.

typedef struct *_trdc_slave_memory_hardware_config* trdc_slave_memory_hardware_config_t
Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *_trdc_did_sel* trdc_did_sel_t
TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

typedef enum *_trdc_secure_attr* trdc_secure_attr_t
TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

typedef struct *_trdc_processor_domain_assignment* trdc_processor_domain_assignment_t
Domain assignment for the processor bus master.

typedef enum *_trdc_privilege_attr* trdc_privilege_attr_t
TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

typedef struct *_trdc_non_processor_domain_assignment*
trdc_non_processor_domain_assignment_t
Domain assignment for the non-processor bus master.

typedef struct *_trdc_idau_config* trdc_idau_config_t
IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct *_trdc_flw_config* trdc_flw_config_t
FLW(Flash Logical Window) configuration.

typedef enum *_trdc_controller* trdc_controller_t
TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

typedef enum *_trdc_error_state* trdc_error_state_t
TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or TRDC_MRCn_DERR_W1[EST].

typedef enum *_trdc_error_attr* trdc_error_attr_t
TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

typedef enum *_trdc_error_type* trdc_error_type_t
TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

typedef struct *_trdc_domain_error* trdc_domain_error_t
TRDC domain error definition.

typedef struct *_trdc_memory_access_control_config* trdc_memory_access_control_config_t
Memory access control configuration for MBC/MRC.

typedef struct *_trdc_mrc_region_descriptor_config* trdc_mrc_region_descriptor_config_t
The configuration of each region descriptor per domain per MRC instance.

typedef struct *_trdc_mbc_nse_update_config* trdc_mbc_nse_update_config_t
The configuration of MBC NSE update.

typedef struct *_trdc_mbc_memory_block_config* trdc_mbc_memory_block_config_t
The configuration of each memory block per domain per MBC instance.

FSL_TRDC_DRIVER_VERSION

struct _trdc_hardware_config
#include <fsl_trdc.h> TRDC hardware configuration.

Public Members

uint8_t masterNumber
Number of bus masters.

uint8_t domainNumber
Number of domains.

uint8_t mbcNumber
Number of MBCs.

uint8_t mrcNumber
Number of MRCs.

struct _trdc_slave_memory_hardware_config
#include <fsl_trdc.h> Hardware configuration of the two slave memories within each MBC(memory block checker).

Public Members

uint32_t blockNum
Number of blocks.

uint32_t blockSize
Block size.

struct _trdc_processor_domain_assignment
#include <fsl_trdc.h> Domain assignment for the processor bus master.

Public Members

uint32_t domainId
Domain ID.

uint32_t domainIdSelect
Domain ID select method, see trdc_did_sel_t.

uint32_t ___pad0___
Reserved.

uint32_t secureAttr
Secure attribute, see trdc_secure_attr_t.

uint32_t ___pad1___
Reserved.

uint32_t lock
Lock the register.

uint32_t ___pad2___
Reserved.

struct __trdc__non__processor__domain__assignment
#include <fsl_trdc.h> Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId
Domain ID.

uint32_t privilegeAttr
Privileged attribute, see trdc_privilege_attr_t.

uint32_t secureAttr
Secure attribute, see trdc_secure_attr_t.

uint32_t bypassDomainId
Bypass domain ID.

uint32_t ___pad0___
Reserved.

uint32_t lock
Lock the register.

uint32_t ___pad1___
Reserved.

struct __trdc__idau__config
#include <fsl_trdc.h> IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

Public Members

uint32_t ___pad0___
Reserved.

uint32_t lockSecureVTOR
Disable writes to secure VTOR(Vector Table Offset Register).

uint32_t lockNonsecureVTOR
Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32_t lockSecureMPU
Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32_t lockNonsecureMPU
Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32_t lockSAU
Disable writes to SAU(Security Attribution Unit) registers.

uint32_t ___pad1___
Reserved.

struct __trdc__flw__config
#include <fsl_trdc.h> FLW(Flash Logical Window) configuration.

Public Members

uint16_t blockCount
Block count of the Flash Logic Window in 32KByte blocks.

uint32_t arrayBaseAddr
Flash array base address of the Flash Logical Window.

bool lock
Disable writes to FLW registers.

bool enable
Enable FLW function.

struct __trdc__domain__error
#include <fsl_trdc.h> TRDC domain error definition.

Public Members

trdc_controller_t controller
Which controller captured access violation.

uint32_t address
Access address that generated access violation.

trdc_error_state_t errorState
Error state.

trdc_error_attr_t errorAttr
Error attribute.

trdc_error_type_t errorType
Error type.

uint8_t errorPort
Error port.

uint8_t domainId
Domain ID.

uint8_t slaveMemoryIdx
The slave memory index. Only apply when violation in MBC.

struct __trdc__memory__access__control__config
#include <fsl_trdc.h> Memory access control configuration for MBC/MRC.

Public Members

uint32_t nonsecureUsrX
Allow nonsecure user execute access.

uint32_t nonsecureUsrW
Allow nonsecure user write access.

uint32_t nonsecureUsrR
: Allow nonsecure user read access.

uint32_t ___pad0___
: Reserved.

uint32_t nonsecurePrivX
: Allow nonsecure privilege execute access.

uint32_t nonsecurePrivW
: Allow nonsecure privilege write access.

uint32_t nonsecurePrivR
: Allow nonsecure privilege read access.

uint32_t ___pad1___
: Reserved.

uint32_t secureUsrX
: Allow secure user execute access.

uint32_t secureUsrW
: Allow secure user write access.

uint32_t secureUsrR
: Allow secure user read access.

uint32_t ___pad2___
: Reserved.

uint32_t securePrivX
: Allownsecure privilege execute access.

uint32_t securePrivW
: Allownsecure privilege write access.

uint32_t securePrivR
: Allownsecure privilege read access.

uint32_t ___pad3___
: Reserved.

uint32_t lock
: Lock the configuration until next reset, only apply to access control register 0.

struct __trdc__mrc__region__descriptor__config
: *#include <fsl_trdc.h>* The configuration of each region descriptor per domain per MRC instance.

Public Members

uint8_t memoryAccessControlSelect
: Select one of the 8 access control policies for this region, for access cotrol policies see trdc_memory_access_control_config_t.

uint32_t startAddr
: Physical start address.

bool valid
: Lock the register.

bool nseEnable
: Enable non-secure accesses and disable secure accesses.

uint32_t endAddr
: Physical start address.

uint8_t mrcIdx
: The index of the MRC for this configuration to take effect.

uint8_t domainIdx
: The index of the domain for this configuration to take effect.

uint8_t regionIdx
: The index of the region for this configuration to take effect.

struct __trdc_mbc_nse_update_config
#include <fsl_trdc.h> The configuration of MBC NSE update.

Public Members

uint32_t ___pad0___
: Reserved.

uint32_t wordIdx
: MBC configuration word index to be updated.

uint32_t ___pad1___
: Reserved.

uint32_t memorySelect
: Bit mask of the selected memory to be updated. _trdc_MBC_memory.

uint32_t ___pad2___
: Reserved.

uint32_t domianSelect
: Bit mask of the selected domain to be updated. _trdc_MBC_domain.

uint32_t ___pad3___
: Reserved.

uint32_t autoIncrement
: Whether to increment the word index after current word is updated using this configuration.

struct __trdc_mbc_memory_block_config
#include <fsl_trdc.h> The configuration of each memory block per domain per MBC instance.

Public Members

uint32_t memoryAccessControlSelect
: Select one of the 8 access control policies for this memory block, for access cotrol policies see trdc_memory_access_control_config_t.

uint32_t nseEnable
: Enable non-secure accesses and disable secure accesses.

uint32_t mbcIdx
: The index of the MBC for this configuration to take effect.

uint32_t domainIdx
The index of the domain for this configuration to take effect.

uint32_t slaveMemoryIdx
The index of the slave memory for this configuration to take effect.

uint32_t memoryBlockIdx
The index of the memory block for this configuration to take effect.

2.124 TSTMR: Timestamp Timer Driver

void TSTMR_Init(TSTMR_Type *base)
Init TSTMR.

This function initializes the TSTMR module.

Parameters

- base – TSTMR peripheral base address.

void TSTMR_Deinit(TSTMR_Type *base)
Deinit TSTMR.

This function deinitializes the TSTMR module.

Parameters

- base – TSTMR peripheral base address.

FSL_TSTMR_DRIVER_VERSION
Version 2.1.0

static inline uint64_t TSTMR_ReadTimeStamp(TSTMR_Type *base)
Reads the time stamp.

This function reads the low and high registers and returns the 56-bit free running counter value. This can be read by software at any time to determine the software ticks. TSTMR registers can be read with 32-bit accesses only. The TSTMR LOW read should occur first, followed by the TSTMR HIGH read.

Parameters

- base – TSTMR peripheral base address.

Returns
The 56-bit time stamp value.

void TSTMR_DelayUs(TSTMR_Type *base, uint64_t delayInUs)
Delays for a specified number of microseconds.

This function repeatedly reads the timestamp register and waits for the user-specified delay value.

Parameters

- base – TSTMR peripheral base address.
- delayInUs – Delay value in microseconds.

2.125 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.

2.126 VREF: Voltage Reference Driver

void VREF_Init(VREF_Type *base, const *vref_config_t* *config)
Enables the clock gate and configures the VREF module according to the configuration structure.

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up vref_config_t parameters and how to call the VREF_Init function by passing in these parameters.

```
vref_config_t vrefConfig;
VREF_GetDefaultConfig(VREF, &vrefConfig);
vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
VREF_Init(VREF, &vrefConfig);
```

Parameters

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

void VREF_Deinit(VREF_Type *base)

Stops and disables the clock for the VREF module.

This function should be called to shut down the module. This is an example.

```
vref_config_t vrefUserConfig;
VREF_GetDefaultConfig(VREF, &vrefUserConfig);
VREF_Init(VREF, &vrefUserConfig);
...
VREF_Deinit(VREF);
```

Parameters

- base – VREF peripheral address.

void VREF_GetDefaultConfig(*vref_config_t* *config)

Initializes the VREF configuration structure.

This function initializes the VREF configuration structure to default values. This is an example.

```
config->bufferMode = kVREF_ModeHighPowerBuffer;
config->enableInternalVoltageRegulator = true;
config->enableChopOscillator           = true;
config->enableHCBandgap                = true;
config->enableCurvatureCompensation    = true;
config->enableLowPowerBuff             = true;
```

Parameters

- config – Pointer to the initialization structure.

void VREF_SetVrefTrimVal(VREF_Type *base, uint8_t trimValue)

Sets a TRIM value for the accurate 1.0V bandgap output.

This function sets a TRIM value for the reference voltage. It will trim the accurate 1.0V bandgap by 0.5mV each step.

Parameters

- base – VREF peripheral address.
- trimValue – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

void VREF_SetTrim21Val(VREF_Type *base, uint8_t trim21Value)

Sets a TRIM value for the accurate buffered VREF output.

This function sets a TRIM value for the reference voltage. If buffer mode be set to other values (Buf21 enabled), it will trim the VREF_OUT by 0.1V each step from 1.0V to 2.1V.

Note: When Buf21 is enabled, the value of UTRIM[TRIM2V1] should be ranged from 0b0000 to 0b1011 in order to trim the output voltage from 1.0V to 2.1V, other values will make the VREF_OUT to default value, 1.0V.

Parameters

- base – VREF peripheral address.
- trim21Value – Value of the trim register to set the output reference voltage (maximum 0xF (4-bit)).

uint8_t VREF_GetVrefTrimVal(VREF_Type *base)

Reads the trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[VREFTRIM] (13:8)

Parameters

- base – VREF peripheral address.

Returns

6-bit value of trim setting.

uint8_t VREF_GetTrim21Val(VREF_Type *base)

Reads the VREF 2.1V trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[TRIM2V1] (3:0),

Parameters

- base – VREF peripheral address.

Returns

4-bit value of trim setting.

static inline void VREF_SetTestUlockValue(VREF_Type *base, uint16_t value)

Set VREF test unlock value.

Parameters

- base – VREF peripheral address.
- value – VREF test unlock value.

static inline uint16_t VREF_GetTestUlockValue(VREF_Type *base)

Get VREF test unlock value.

Parameters

- base – VREF peripheral address.

Returns

VREF test unlock value.

static inline void VREF_SetTestUlockEnable(VREF_Type *base, bool enable)

Decides whether to enable/disable VREF test unlock.

Parameters

- base – VREF peripheral address.
- enable – VREF test unlock enable/disable.
  - **true** VREF test unlock enable.
  - **false** VREF test unlock disable.

static inline void VREF_SetAmpPolValue(VREF_Type *base, uint16_t reverse)

Decides whether to reverse VREF amplifier polarity.

Parameters

- base – VREF peripheral address.
- reverse – VREF amplifier polarity reverse enable/disable.
  - **true** VREF amplifier polarity reverse enable.
  - **false** VREF amplifier polarity reverse disable.

static inline void VREF_SetP7TrimValue(VREF_Type *base, uint8_t value)

Set VREF P7 trim value.

Parameters

- base – VREF peripheral address.
- value – VREF P7 trim value.

static inline uint8_t VREF_GetP7TrimValue(VREF_Type *base)

Get VREF P7 trim value.

Parameters

- base – VREF peripheral address.

Returns

VREF P7 trim value.

static inline void VREF_SetChopOscTrimValue(VREF_Type *base, uint8_t value)

Set VREF chop osc trim value.

Parameters

- base – VREF peripheral address.
- value – VREF chop osc trim value.

static inline uint8_t VREF_GetChopOscTrimValue(VREF_Type *base)

Get VREF chop osc trim value.

Parameters

- base – VREF peripheral address.

Returns

VREF chop osc trim value.

static inline void VREF_SetBandgapMsb(VREF_Type *base, uint8_t value)

Set VREF bandgap MSB value.

Parameters

- base – VREF peripheral address.
- value – VREF bandgap MSB value.

static inline uint8_t VREF_GetBandgapMsb(VREF_Type *base)

Get VREF bandgap MSB value.

Parameters

- base – VREF peripheral address.

Returns

VREF bandgap MSB value.

static inline void VREF_SetBandgapLsb(VREF_Type *base, uint8_t value)

Set VREF bandgap LSB value.

Parameters

- base – VREF peripheral address.
- value – VREF bandgap LSB value.

static inline uint8_t VREF_GetBandgapLsb(VREF_Type *base)

Get VREF bandgap LSB value.

Parameters

- base – VREF peripheral address.

Returns
VREF bandgap LSB value.

static inline void VREF_SetTempCompMsb(VREF_Type *base, uint8_t value)
Set VREF temperature compensation MSB value.

Parameters

- base – VREF peripheral address.
- value – VREF temperature compensation MSB value.

static inline uint8_t VREF_GetTempCompMsb(VREF_Type *base)
Get VREF temperature compensation MSB value.

Parameters

- base – VREF peripheral address.

Returns
VREF temperature compensation MSB value.

static inline void VREF_SetTempCompLsb(VREF_Type *base, uint8_t value)
Set VREF temperature compensation LSB value.

Parameters

- base – VREF peripheral address.
- value – VREF temperature compensation LSB value.

static inline uint8_t VREF_GetTempCompLsb(VREF_Type *base)
Get VREF temperature compensation LSB value.

Parameters

- base – VREF peripheral address.

Returns
VREF temperature compensation LSB value.

FSL_VREF_DRIVER_VERSION
Version 2.4.0.

enum _vref_buffer_mode
VREF buffer modes.

Values:

enumerator kVREF_ModeBandgapOnly
Bandgap enabled/standby.

enumerator kVREF_ModeLowPowerBuffer
Low-power buffer mode enabled

enumerator kVREF_ModeHighPowerBuffer
High-power buffer mode enabled

typedef enum *_vref_buffer_mode* vref_buffer_mode_t
VREF buffer modes.

typedef struct *_vref_config* vref_config_t
The description structure for the VREF module.

struct _vref_config
#include <fsl_vref.h> The description structure for the VREF module.

Public Members

vref_buffer_mode_t bufferMode
Buffer mode selection

bool enableInternalVoltageRegulator
Provide additional supply noise rejection.

bool enableChopOscillator
Enable Chop oscillator.

bool enableHCBandgap
Enable High-Accurate bandgap.

bool enableCurvatureCompensation
Enable second order curvature compensation.

bool enableLowPowerBuff
Provides bias current for other peripherals.

2.127 Xbar

void XBAR_Init(xbar_instance_t xbarInstance)
Initializes the XBAR modules.

This function un-gates the XBAR clock.

Parameters

- xbarInstance – XBAR peripheral address.

void XBAR_Deinit(xbar_instance_t xbarInstance)
Shutdown the XBAR modules.

This function disables XBAR clock.

Parameters

- xbarInstance – XBAR peripheral address.

status_t XBAR_SetSignalsConnection(xbar_input_signal_t input, xbar_output_signal_t output)
Set connection between the selected XBAR_IN[*] input and the XBAR_OUT[*] output signal.

This function connects the XBAR input to the selected XBAR output. If more than one XBAR module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBAR_SetSignalsConnection(kXBAR_DSC1_InputLogicLow, kXBAR_DSC1_OutputTriggerSyncIn0);
```

Parameters

- input – XBAR input signal.
- output – XBAR output signal.

Return values

- kStatus_Success – Signal connection set successfully.
- kStatus_InvalidArgument – Failed because of invalid argument.

status_t XBAR_ClearOutputStatusFlag(xbar_output_signal_t output)

Clears the edge detection status flags.

Parameters

- output – XBAR output signal.

Return values

- kStatus_Success – Signal connection set successfully.
- kStatus_InvalidArgument – Failed because of invalid argument.

status_t XBAR_GetOutputStatusFlag(xbar_output_signal_t output, bool *flag)

Gets the active edge detection status.

This function gets the active edge detect status of all XBAR_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBAR_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

- output – XBAR output signal.
- flag – get XBAR output status flag.

Return values

- kStatus_Success – Signal connection set successfully.
- kStatus_InvalidArgument – Failed because of invalid argument.

status_t XBAR_SetOutputSignalConfig(xbar_output_signal_t output, const *xbar_control_config_t* *controlConfig)

Configures the XBAR control register.

This function configures an XBAR control register. The active edge detection and the DMA/IRQ function on the corresponding XBAR output can be set.

Example:

```
xbar_control_config_t userConfig;
userConfig.activeEdge = kXBAR_EdgeRising;
userConfig.requestType = kXBAR_RequestInterruptEnable;
XBAR_SetOutputSignalConfig(kXBARA_OutputDMAMUX18, &userConfig);
```

Parameters

- output – XBAR output signal.
- controlConfig – Pointer to structure that keeps configuration of control register.

Return values

- kStatus_Success – Signal connection set successfully.
- kStatus_InvalidArgument – Failed because of invalid argument.

enum _xbar_active_edge

XBAR active edge for detection.

Values:

enumerator kXBAR_EdgeNone

Edge detection status bit never asserts.

enumerator kXBAR_EdgeRising
Edge detection status bit asserts on rising edges.

enumerator kXBAR_EdgeFalling
Edge detection status bit asserts on falling edges.

enumerator kXBAR_EdgeRisingAndFalling
Edge detection status bit asserts on rising and falling edges.

enum _xbar_request
Defines the XBAR DMA and interrupt configurations.

Values:

enumerator kXBAR_RequestDisable
Interrupt and DMA are disabled.

enumerator kXBAR_RequestDMAEnable
DMA enabled, interrupt disabled.

enumerator kXBAR_RequestInterruptEnable
Interrupt enabled, DMA disabled.

typedef uint16_t xbar_reg_t

typedef enum *_xbar_active_edge* xbar_active_edge_t
XBAR active edge for detection.

typedef enum *_xbar_request* xbar_request_t
Defines the XBAR DMA and interrupt configurations.

typedef struct *_xbar_control_config* xbar_control_config_t
Defines the configuration structure of the XBAR control register.

This structure keeps the configuration of XBAR control register for one output. Control registers are available only for a few outputs. Not every XBAR module has control registers.

FSL_XBAR_DRIVER_VERSION

struct xbar_info_t
#include <fsl_xbar.h> Find the instance index from base address and register offset mappings.

struct _xbar_control_config
#include <fsl_xbar.h> Defines the configuration structure of the XBAR control register.

This structure keeps the configuration of XBAR control register for one output. Control registers are available only for a few outputs. Not every XBAR module has control registers.

Public Members

xbar_active_edge_t activeEdge
Active edge to be detected.

xbar_request_t requestType
Selects DMA/Interrupt request.

2.128 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.

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

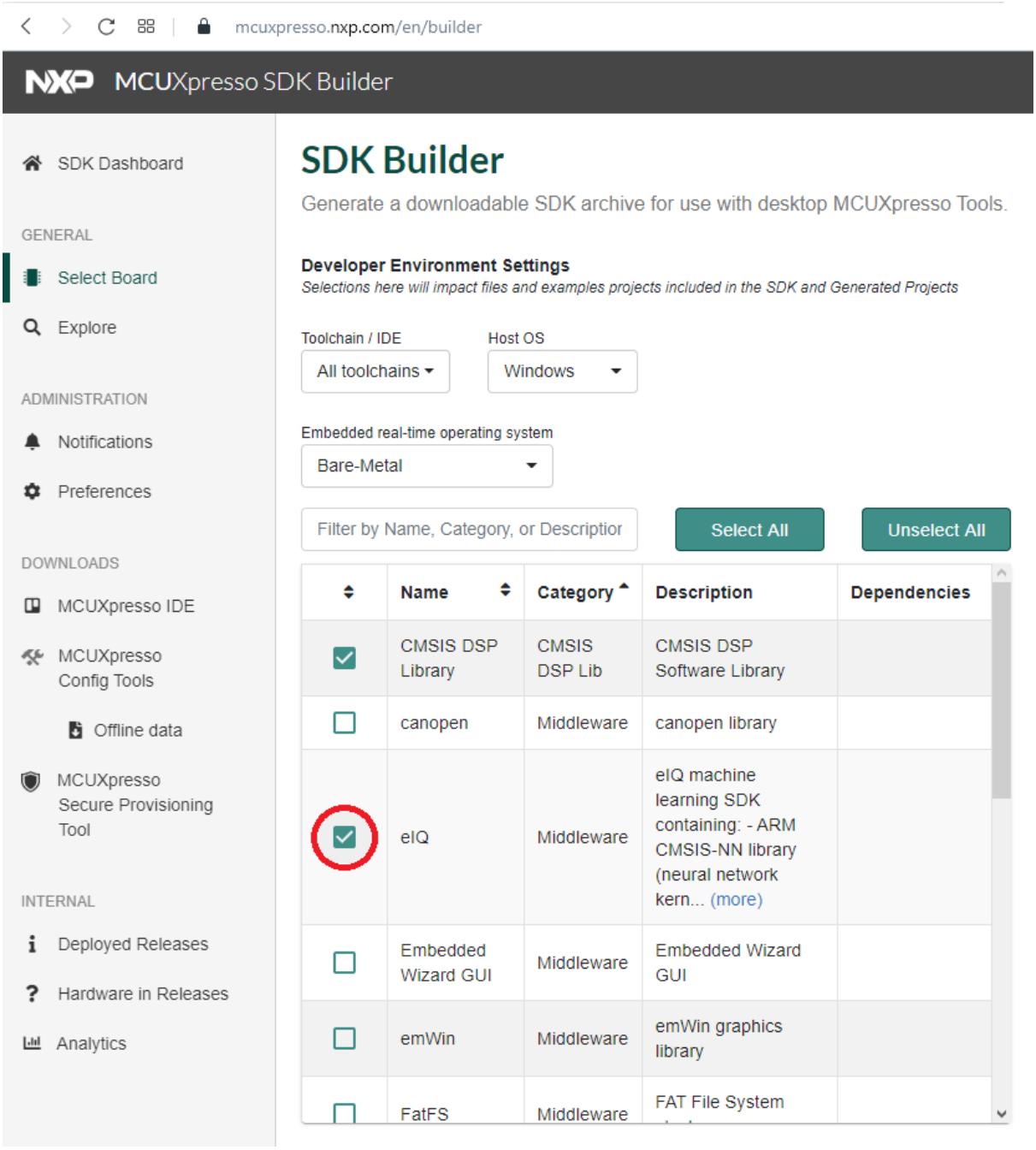
- *Continuous Integration*
- *Benchmarks*
- *Profiling*
- *Memory Management*
- *Logging*
- *Porting Reference Kernels from TfLite to TFLM*
- *Optimized Kernel Implementations*
- *New Platform Support*
- Platform/IP support
  - *Arm IP support*
- *Software Emulation with Renode*
- *Software Emulation with QEMU*

- *Python Dev Guide*
- *Automatically Generated Files*
- *Python Interpreter Guide*

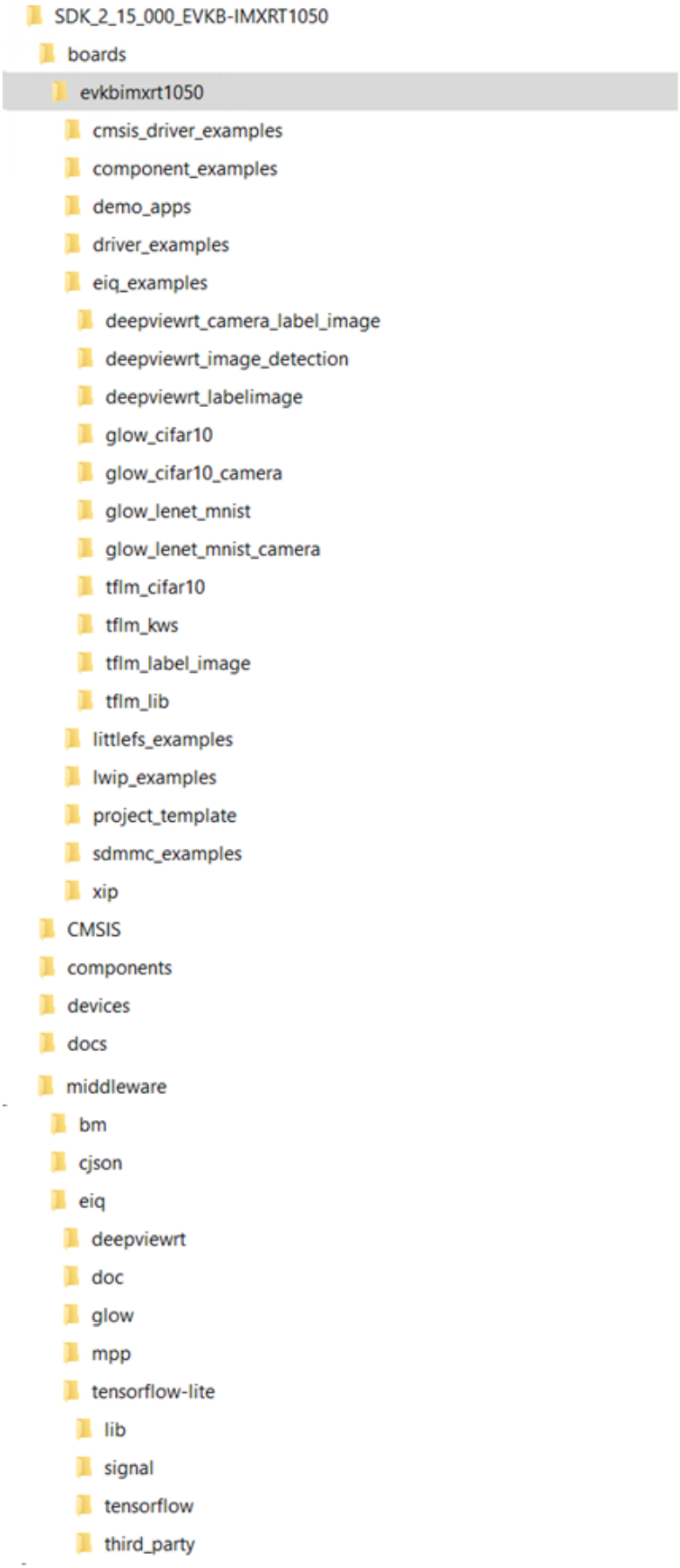
RFCs

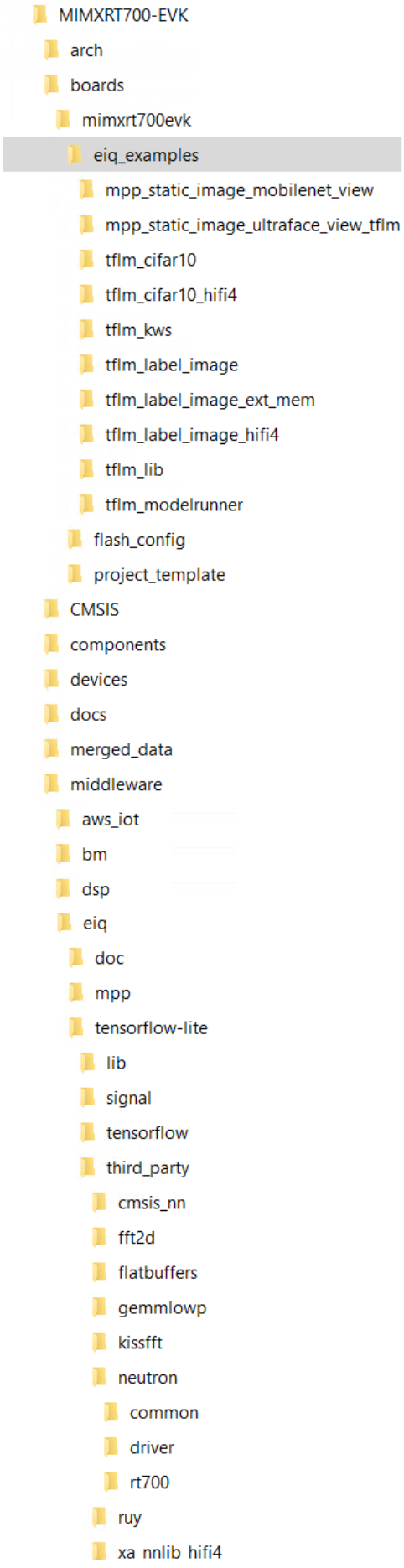
1. *Pre-allocated tensors*
2. *TensorFlow Lite for Microcontrollers Port of 16x8 Quantized Operators*

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

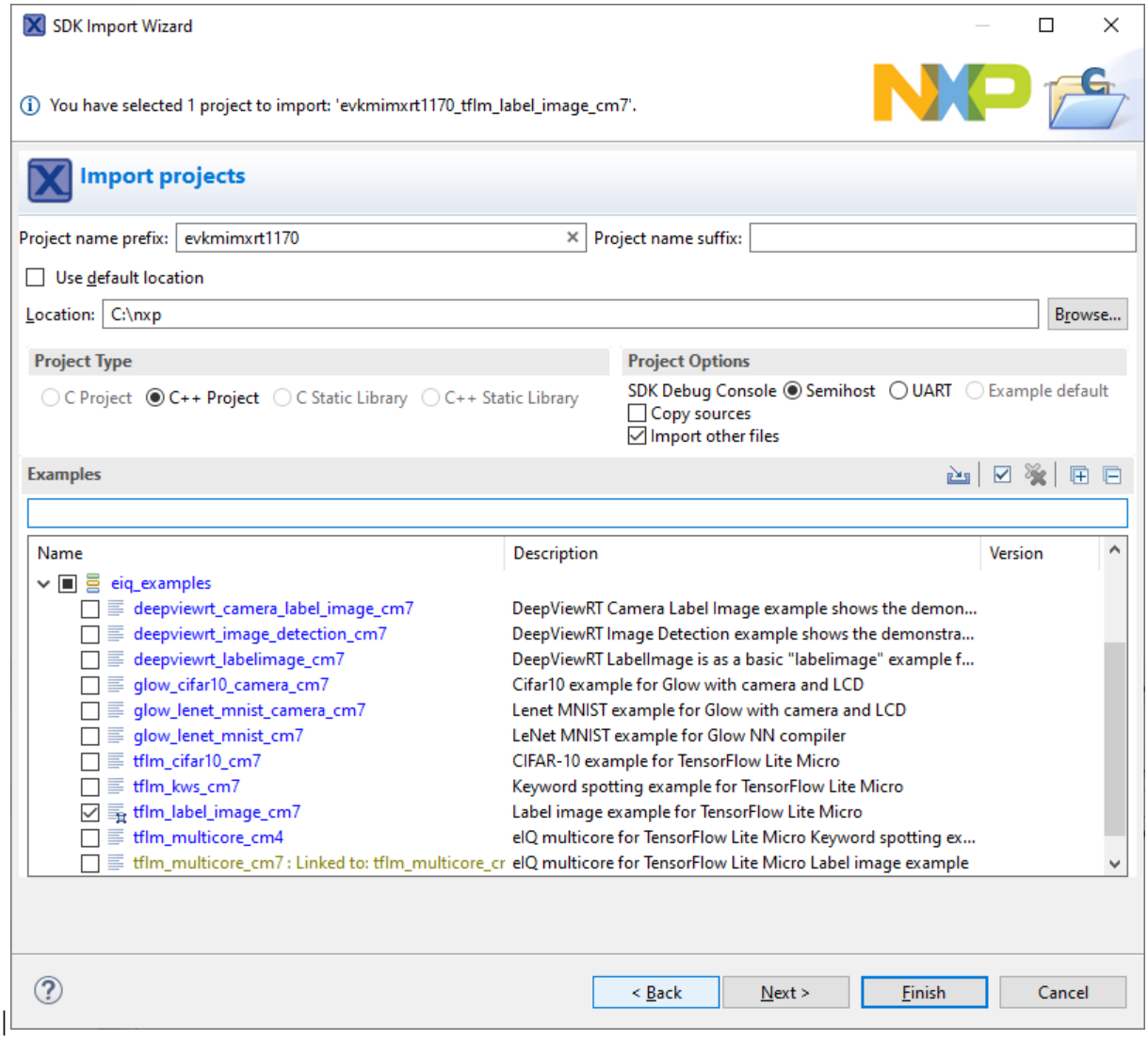
|

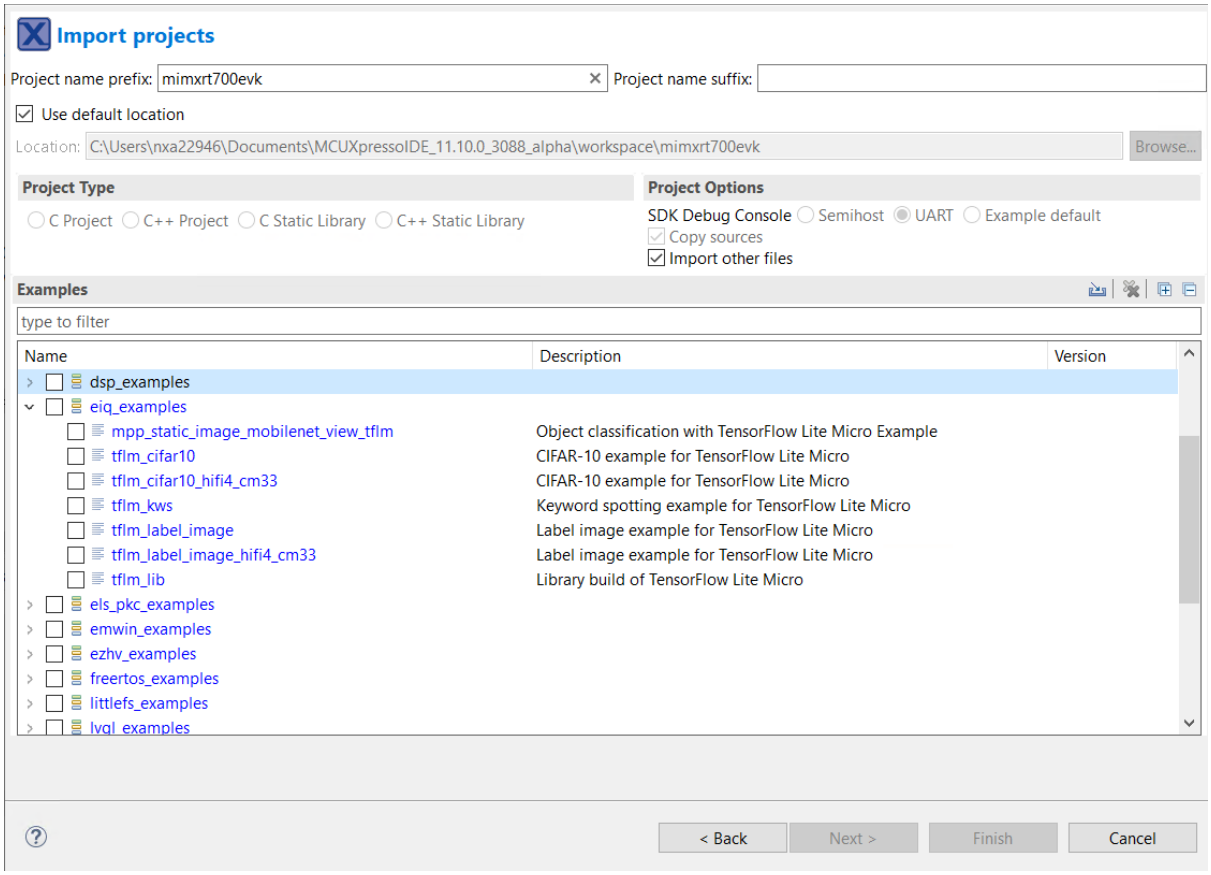
The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

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.

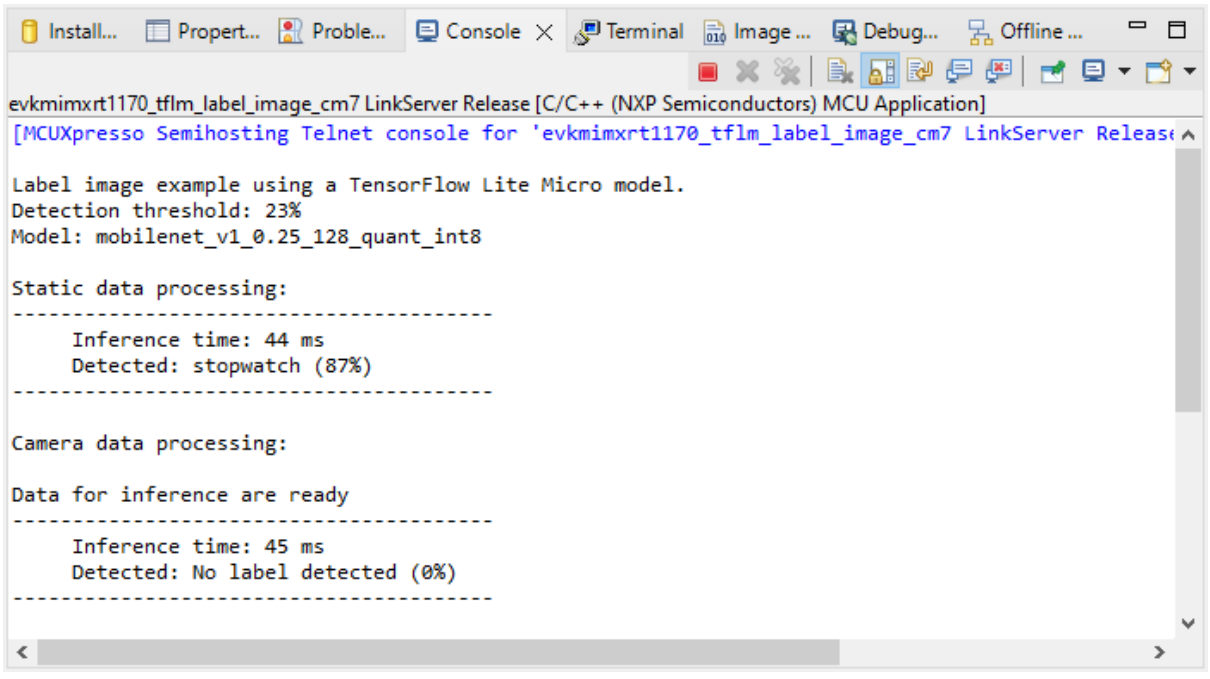
Name	Description	Availability
tflm_c	CIFAR-10 classification of 32 × 32 RGB pixel images into 10 categories using a small Convolutional Neural Network (CNN).	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (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.

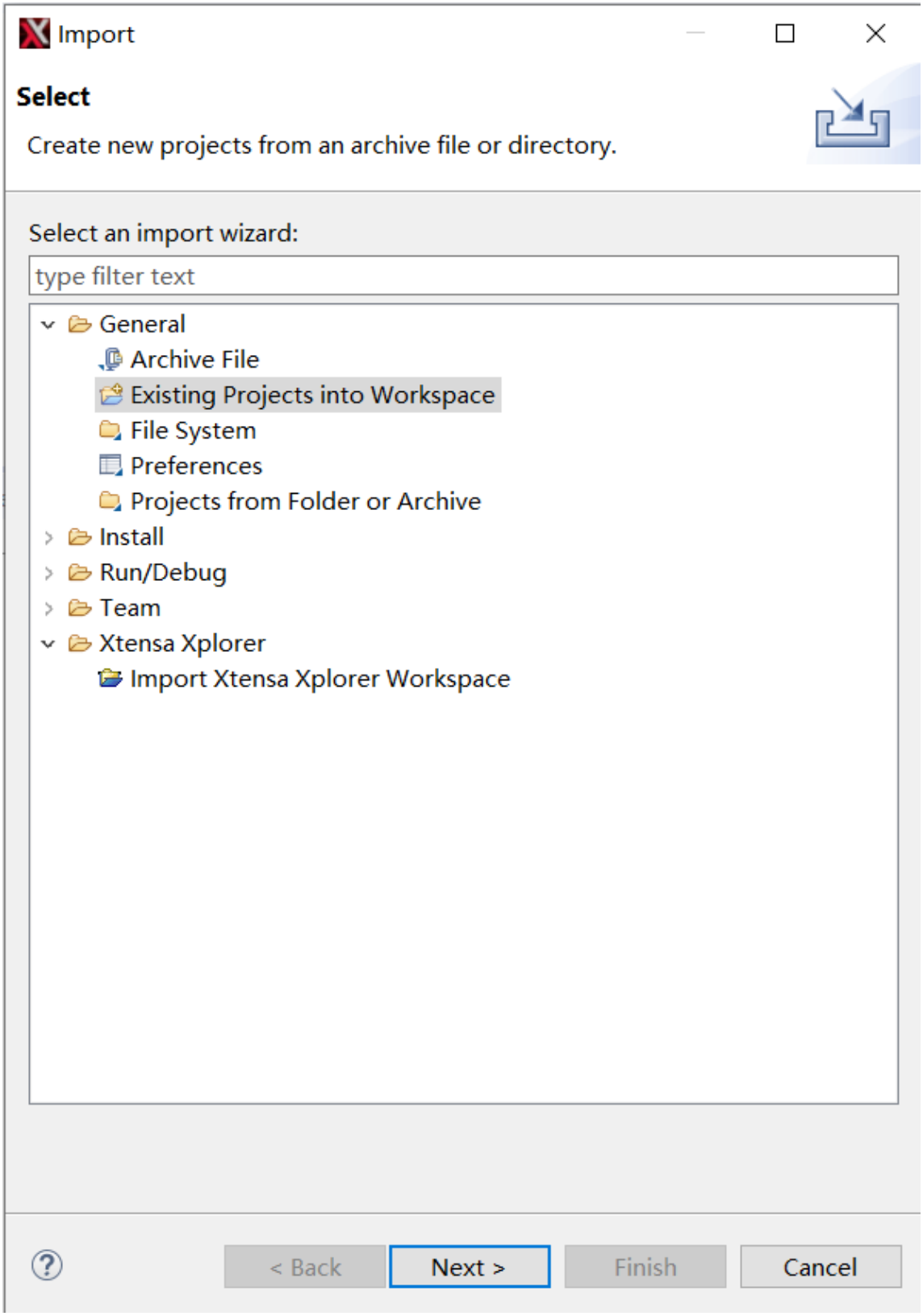
Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

Prepare DSP core for the examples

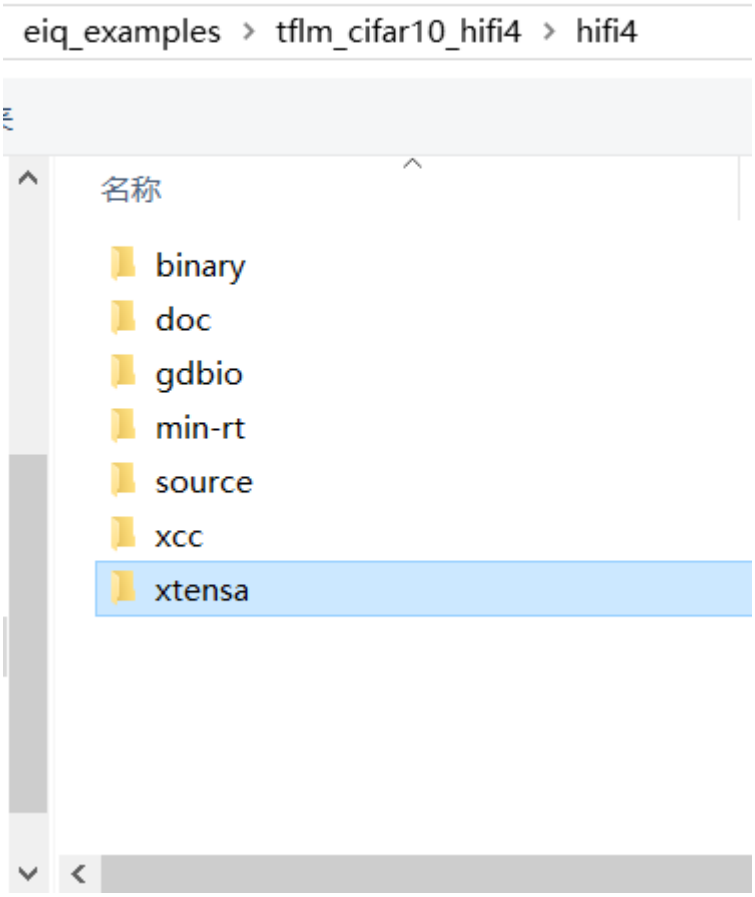
The projects for different supported toolchains are built. The “xcc” project builds on the command line and the “xtensa” directory is an Xplorer IDE project.

To run the tflm_cifar10_hifi4 example, import the SDK sources into the Xplorer IDE.

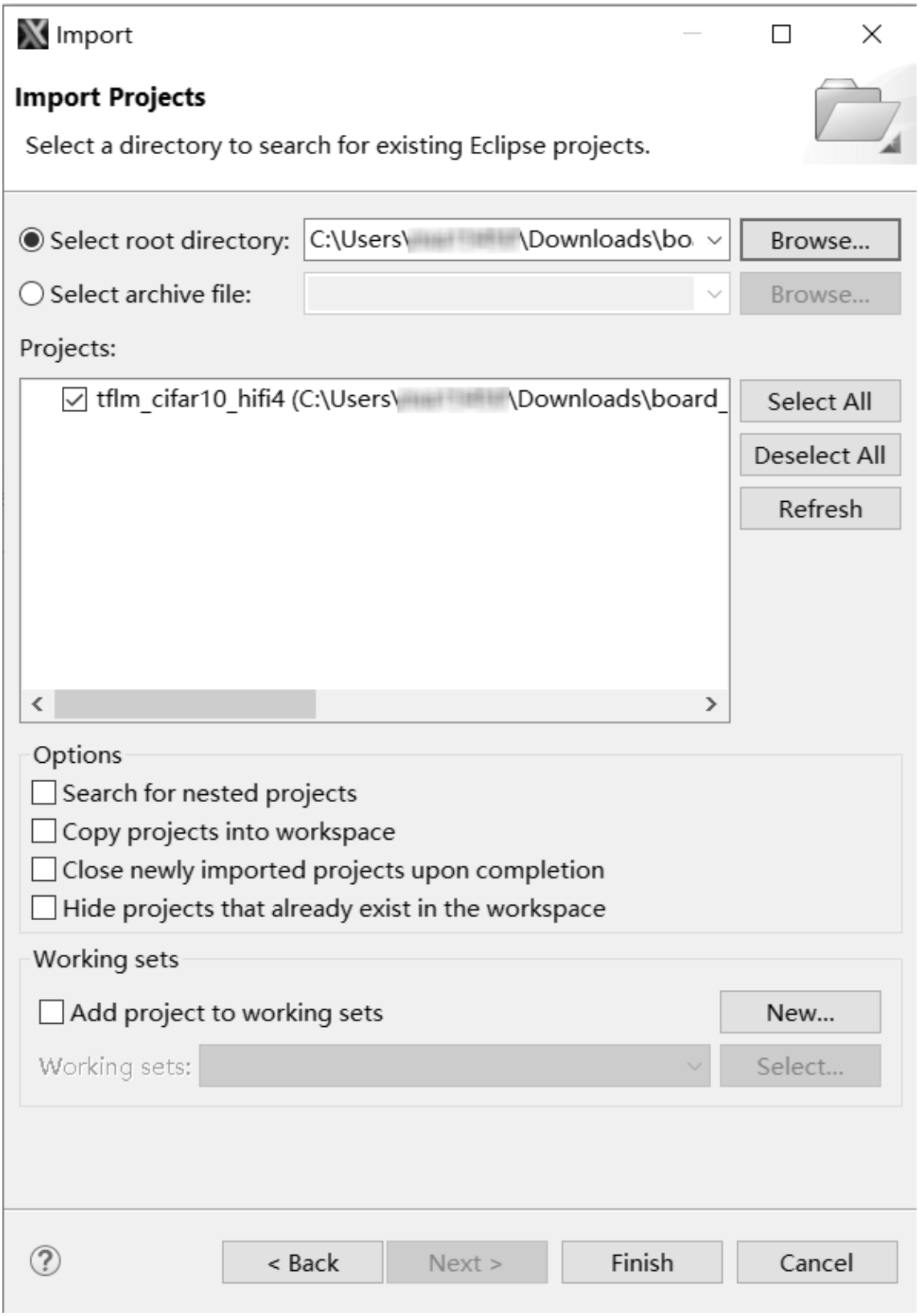
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Click **Next**.
3. Select the SDK directory/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/xtensa as the root directory.

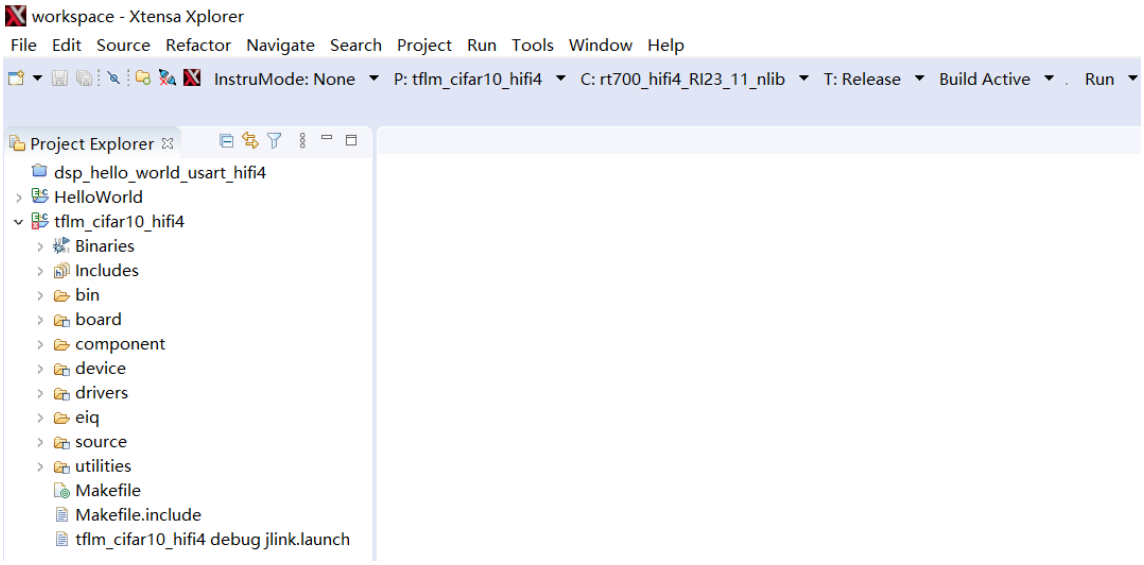


4. Click **Select Folder**.
5. Leave all the other options check boxes blank.

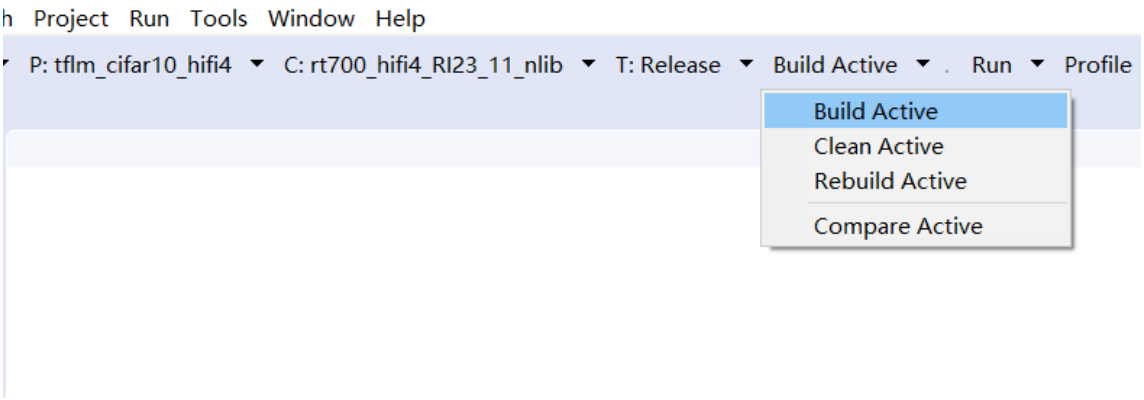


Once imported, the tflm_cifar10_hifi4 example appears in the **Project Explorer**.

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.

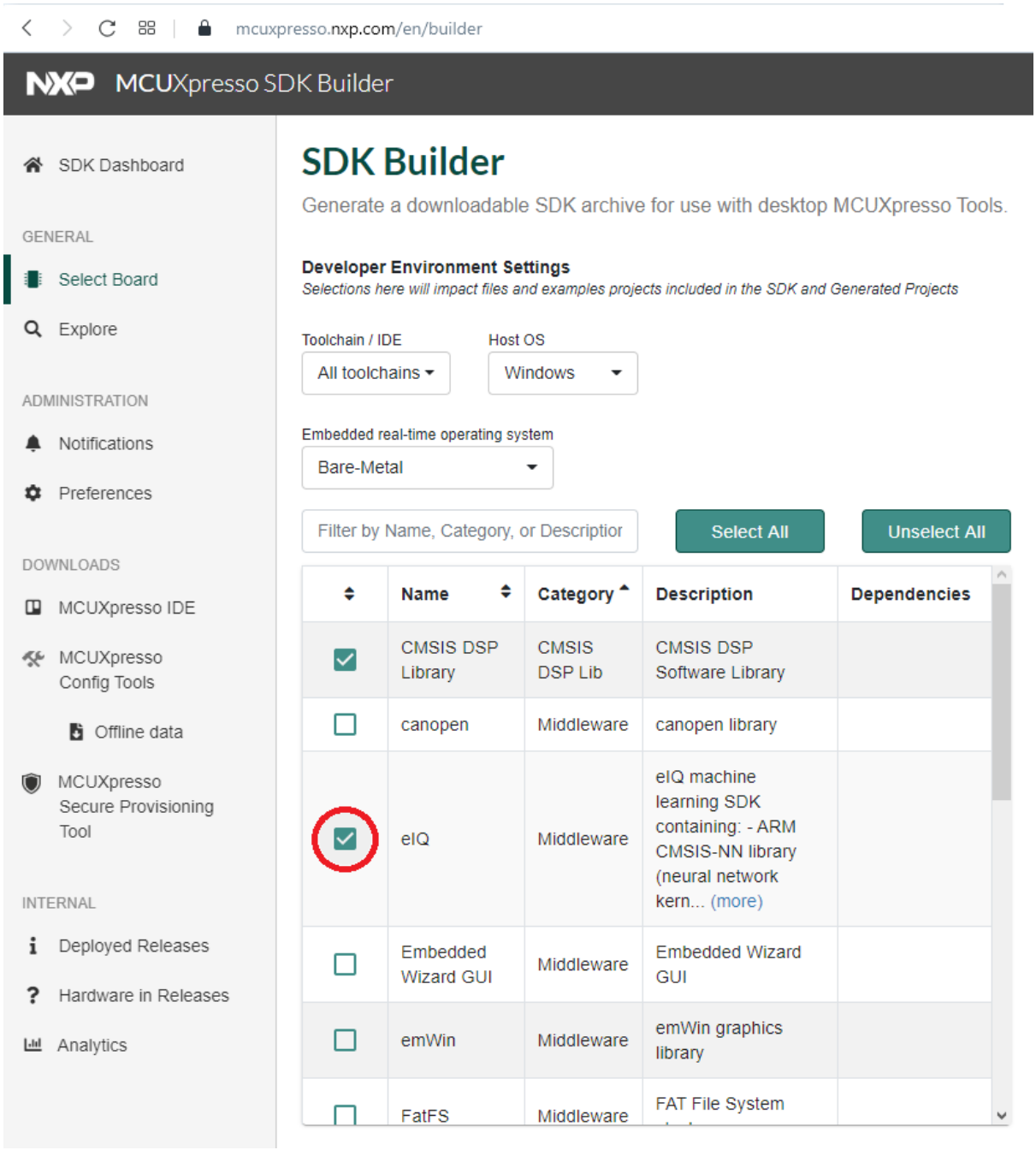
You can obtain the MCUXpresso SDK from MCUXpresso SDK Builder including the IDE. See the *getting_mcuxpress* for details.

In the MCUXpresso SDK, there are 2 projects available related to ExecuTorch:

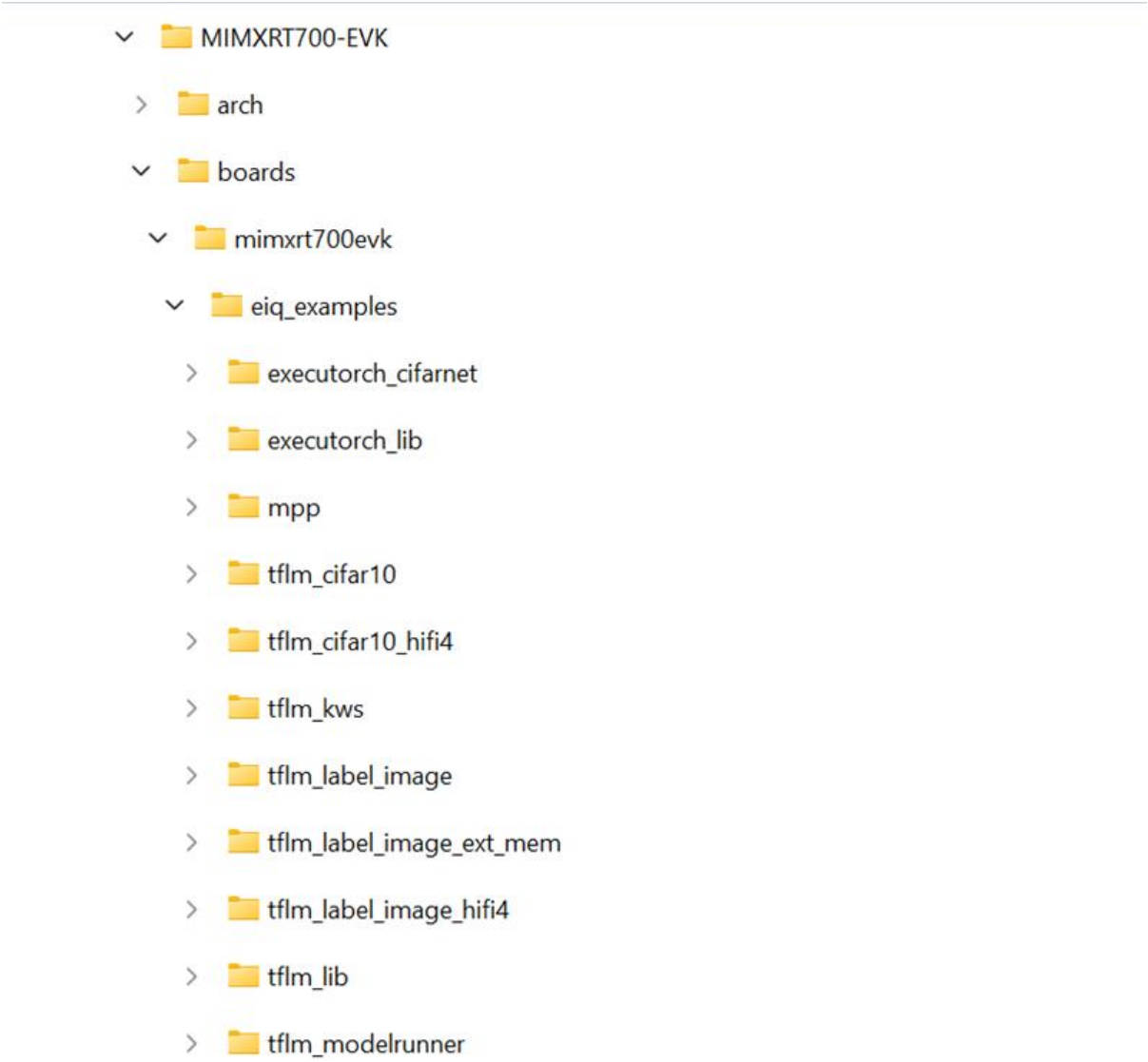
- executorch_lib
- executorch_cifarnet

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

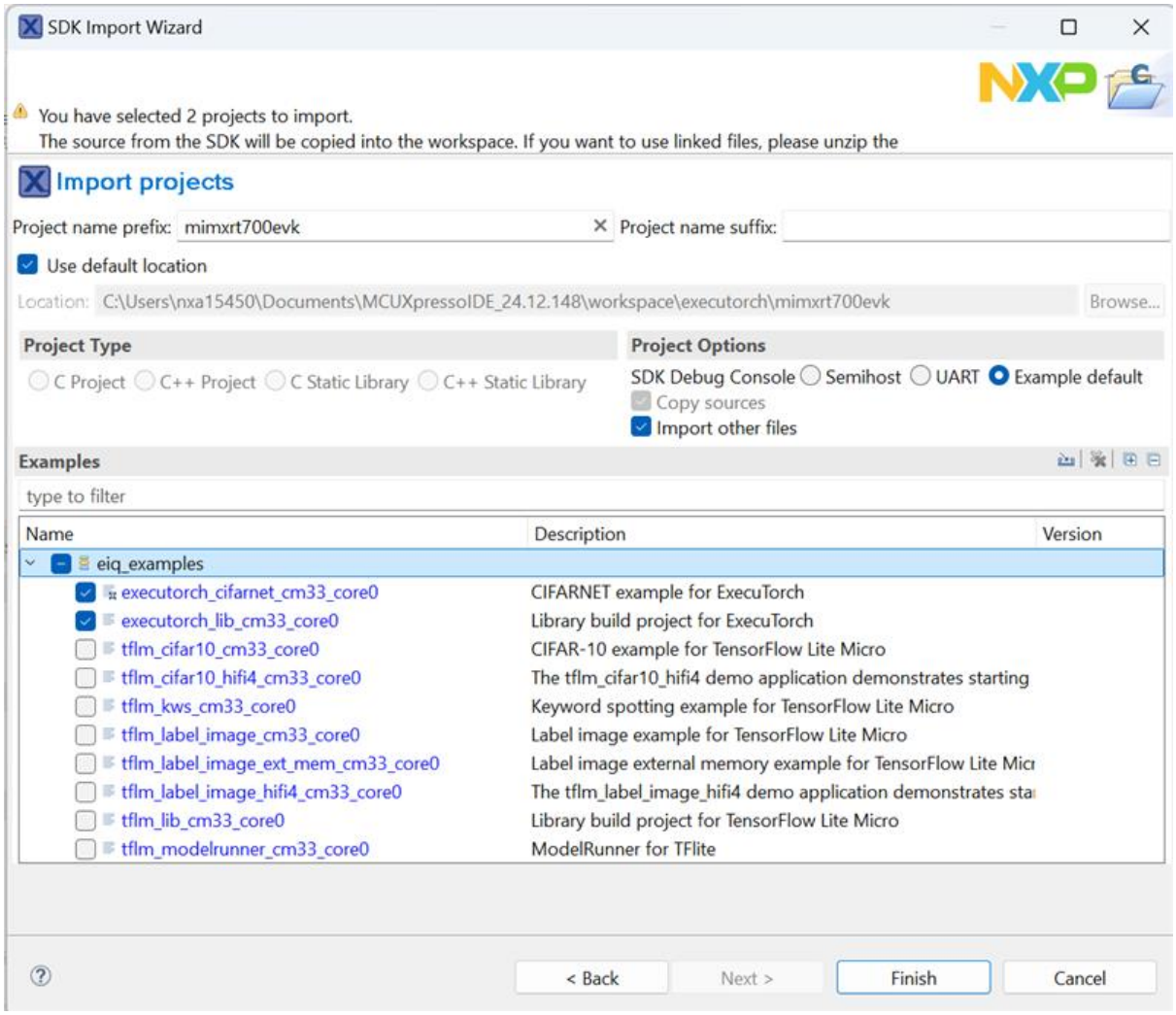
1. When using ARMGCC toolchain, the example application can be built as below. After building the example application, download it to the target with JLink as shown in *Figure 3*, an output message displays on the connected terminal as *Figure 4*.

```
$ 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.
```

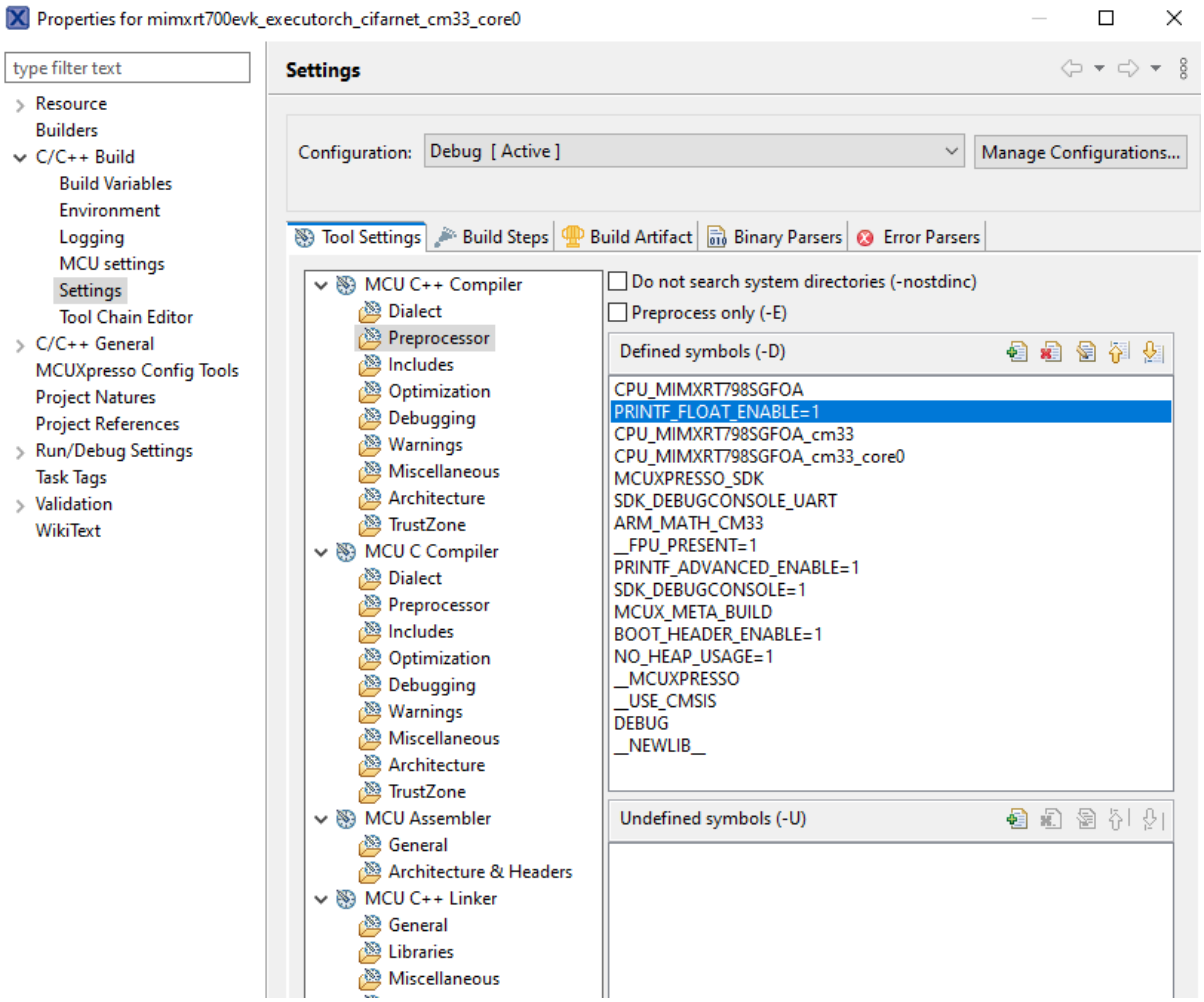
2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 5*.



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.

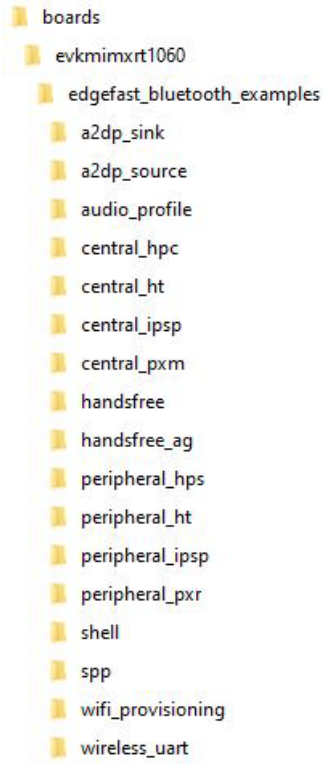
The APIs of the EdgeFast Bluetooth Protocol Abstraction Layer host stack are described in the EdgeFast Bluetooth Protocol Abstraction Layer RM document.

Note: The online document of the Zephyr Bluetooth Host stack is available here: https://docs.zephyrproject.org/latest/reference/bluetooth/index.html.

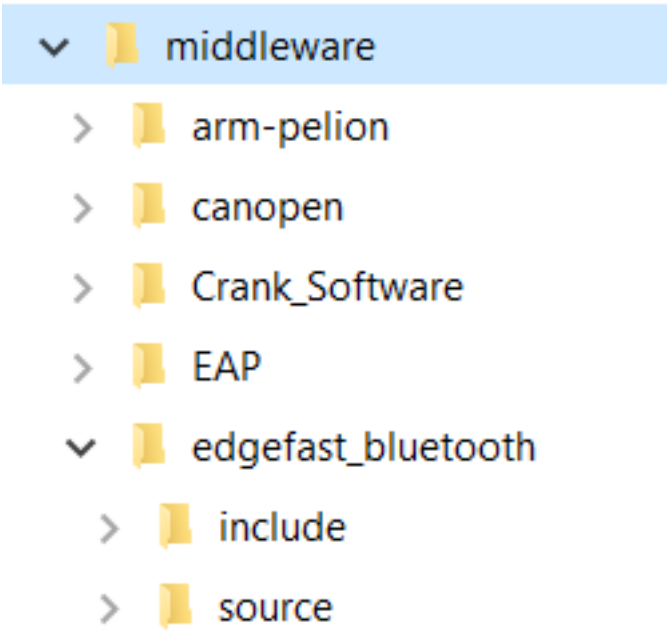
Parent topic:*Introduction*

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.

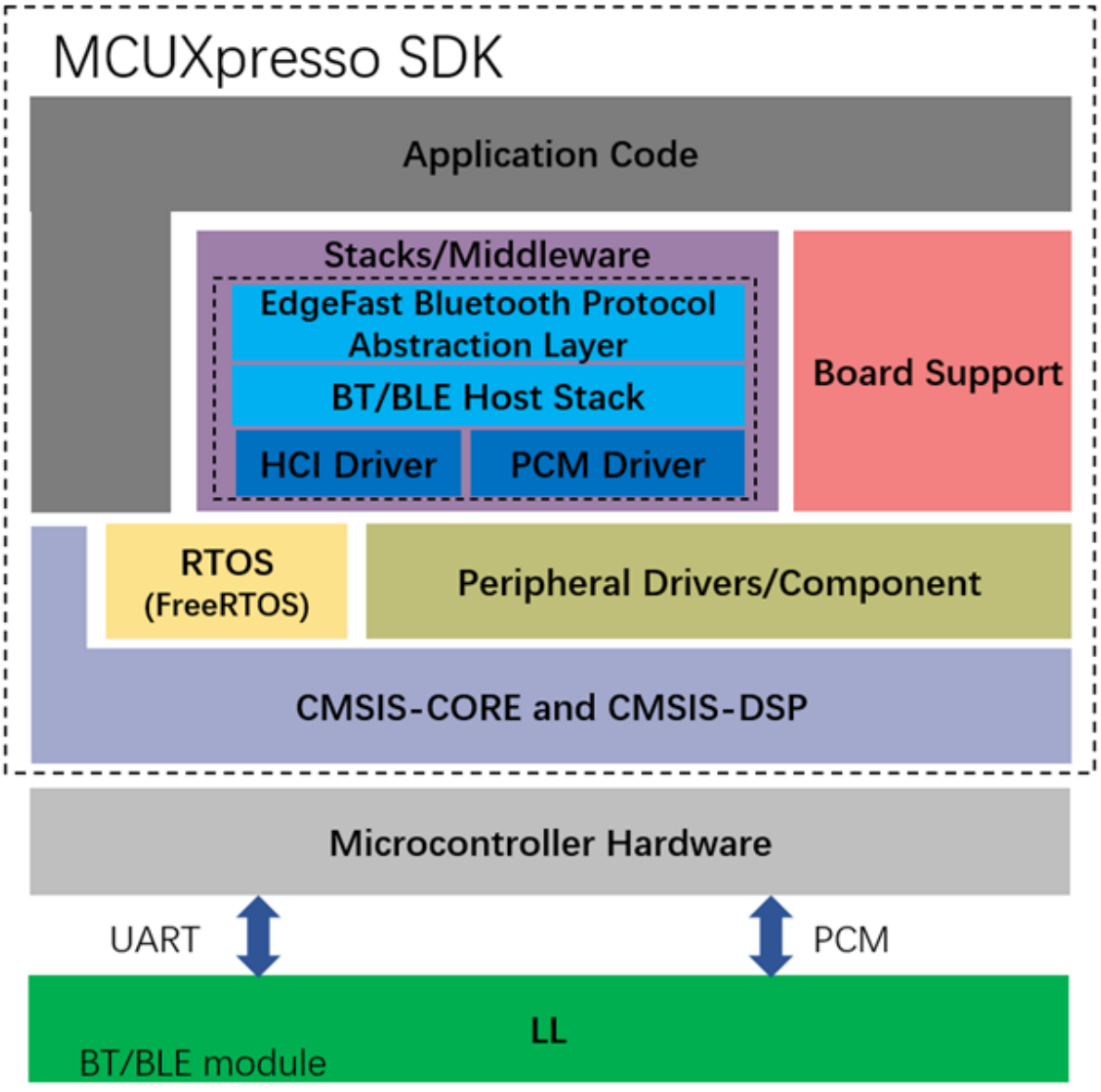
Parent topic:*Overview*

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.

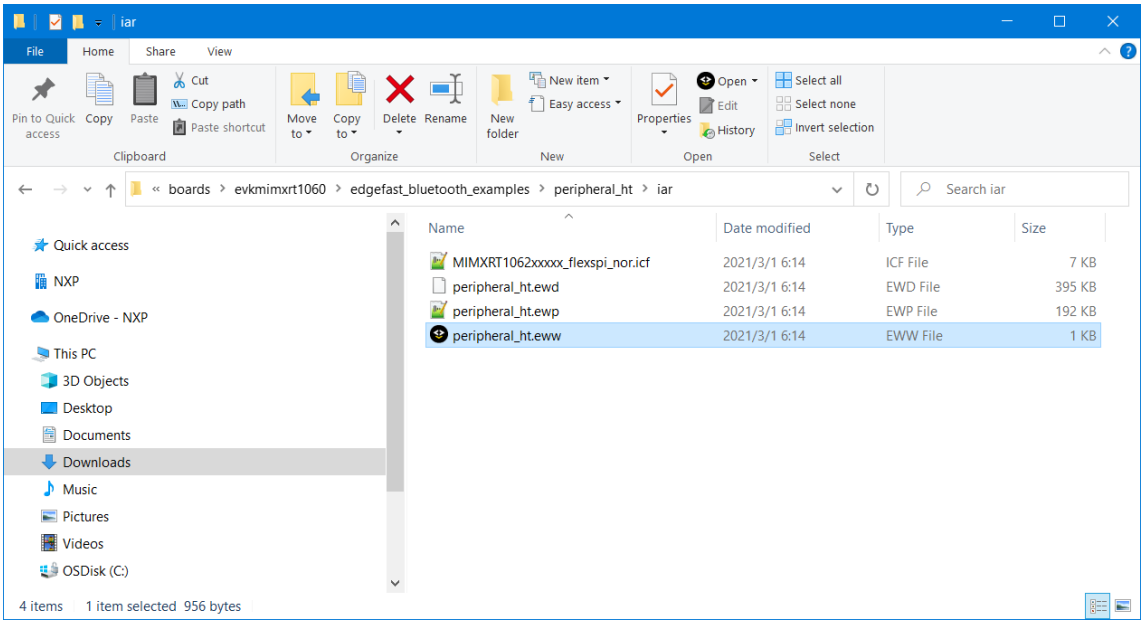
Run a demo application using IAR This document uses EVKRT1060 EdgeFast Bluetooth Protocol Abstraction Layer API example to describe the steps to open a project, build an example, and run a project. For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide*(document MCUXSDKGSUG) 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

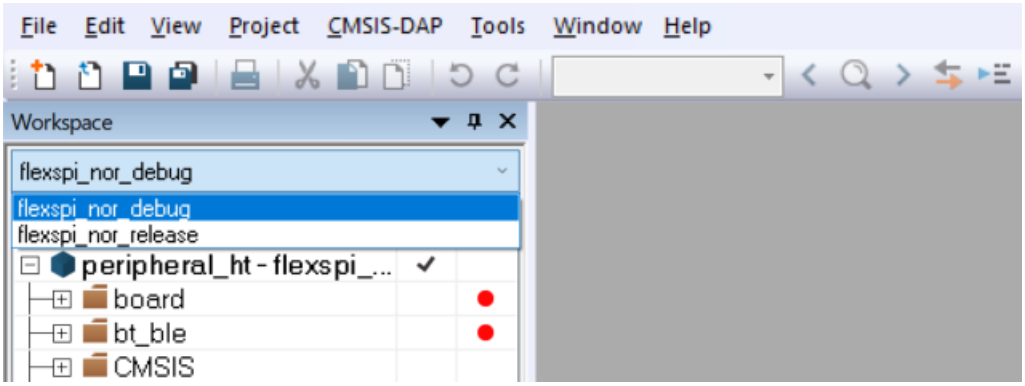
2. Open the IAR workspace file. For example, the highlighted *.eww format file



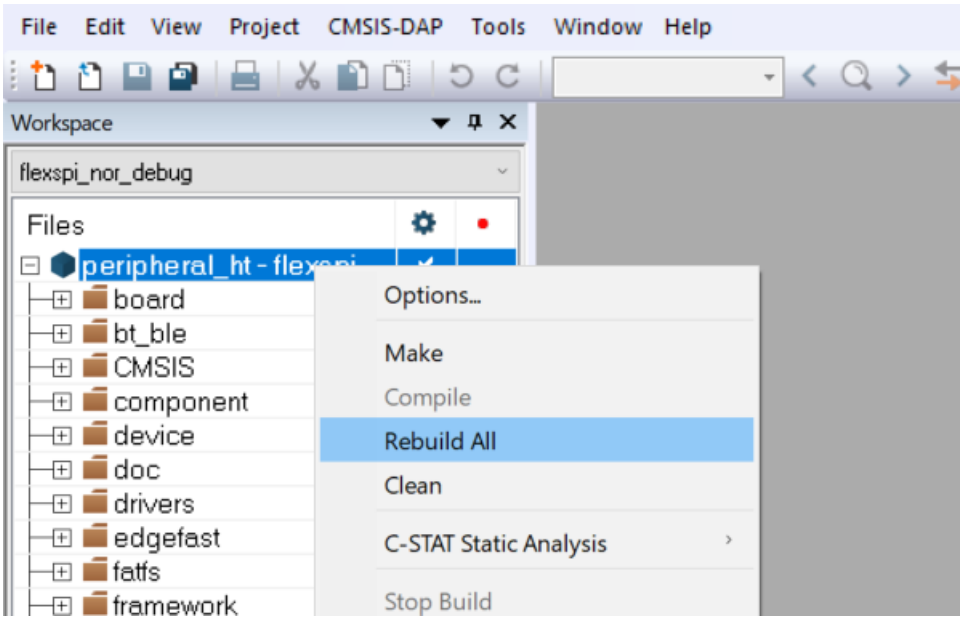
Parent topic: Run a demo application using IAR

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.

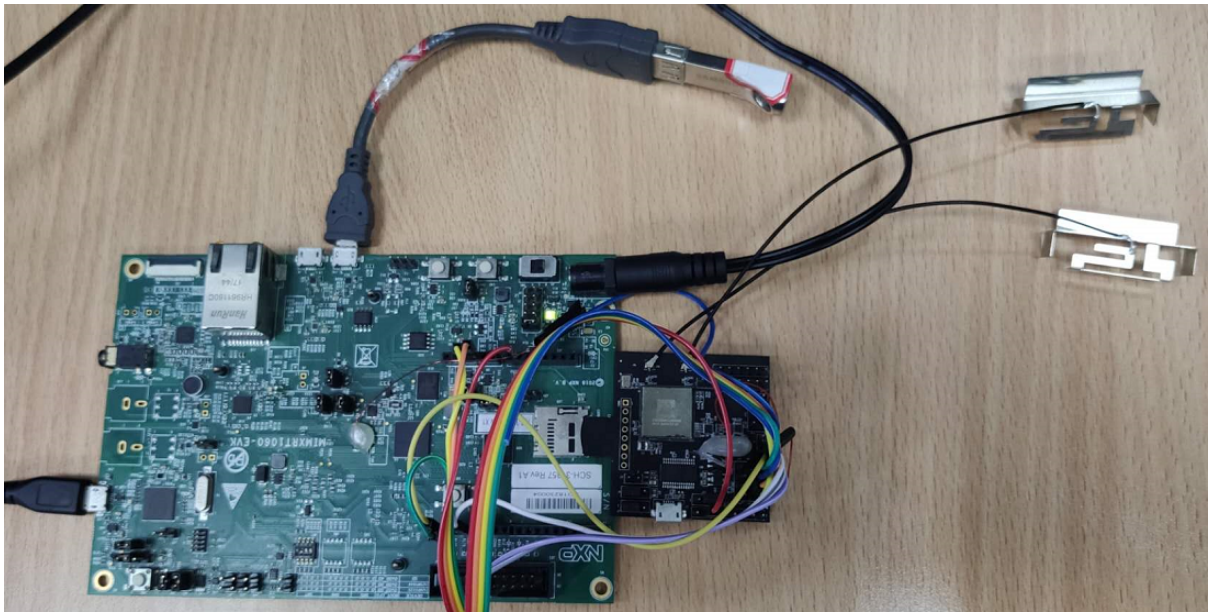


Note: Wireless module does not have flash hardware and requires 512 KB image loaded from board (such as RT1060) on system startup. The 512 KB image is kept on RT1060 side and only flexspi_nor target is supported for Bluetooth examples. Other targets are not supported because memory size limit.

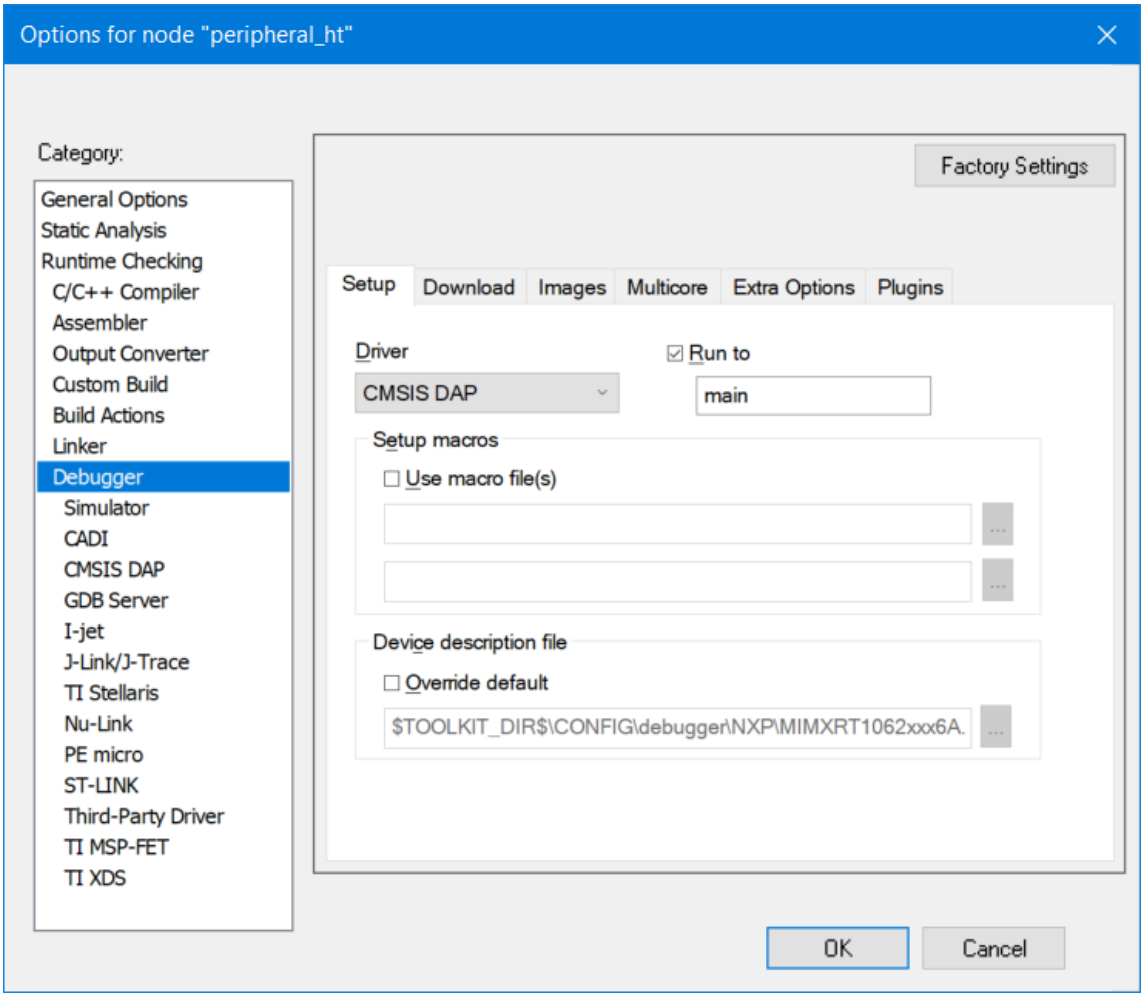
Parent topic: Run a demo application using IAR

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.

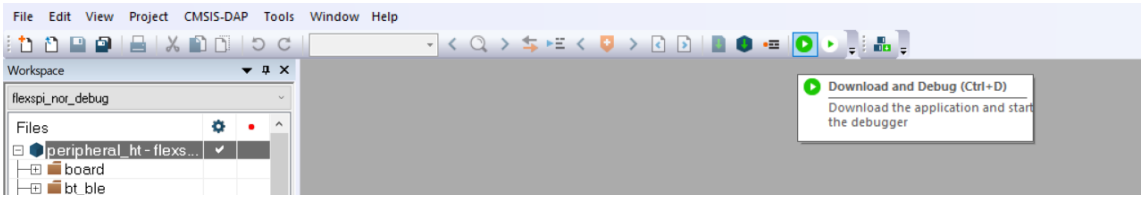
The following figure shows the connection of RT1060 and the uSD wireless module.



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

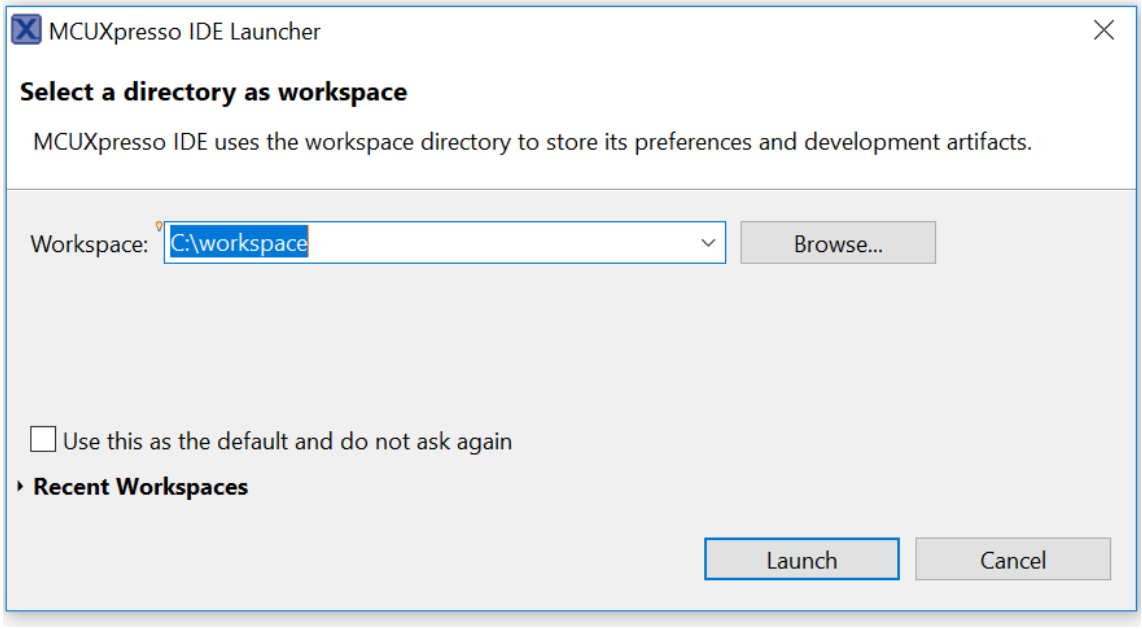
Parent topic:*Demo*

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.

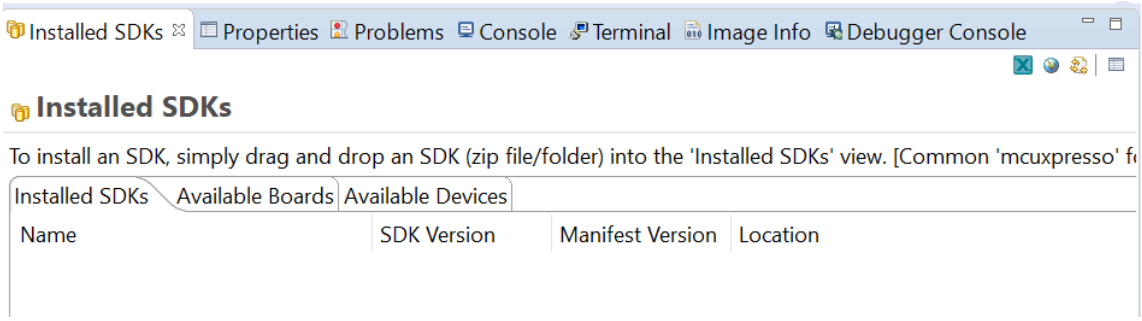
For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf* and refer to the readme file in the corresponding demo's directory.

Open an MCUXpresso IDE example

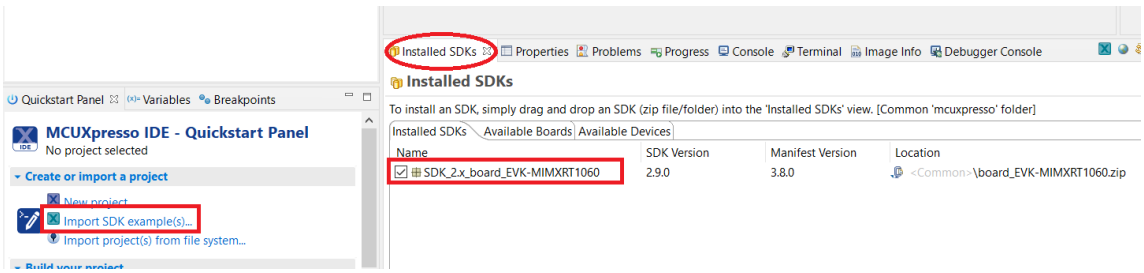
1. Open MCUXpresso IDE and open an existing or a new workspace location.



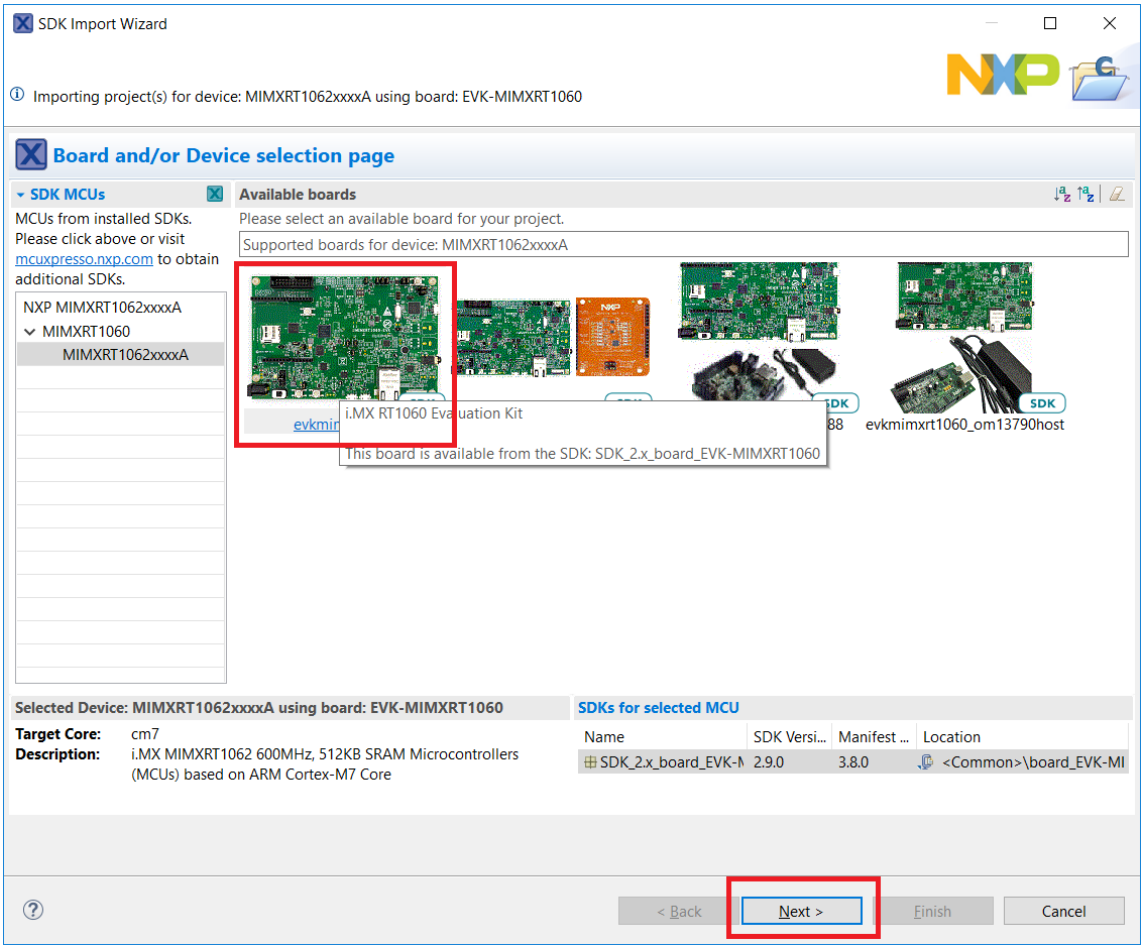
2. Drag and drop the package archive into the MCUXpresso Installed SDKs area in the lower right of the main window.



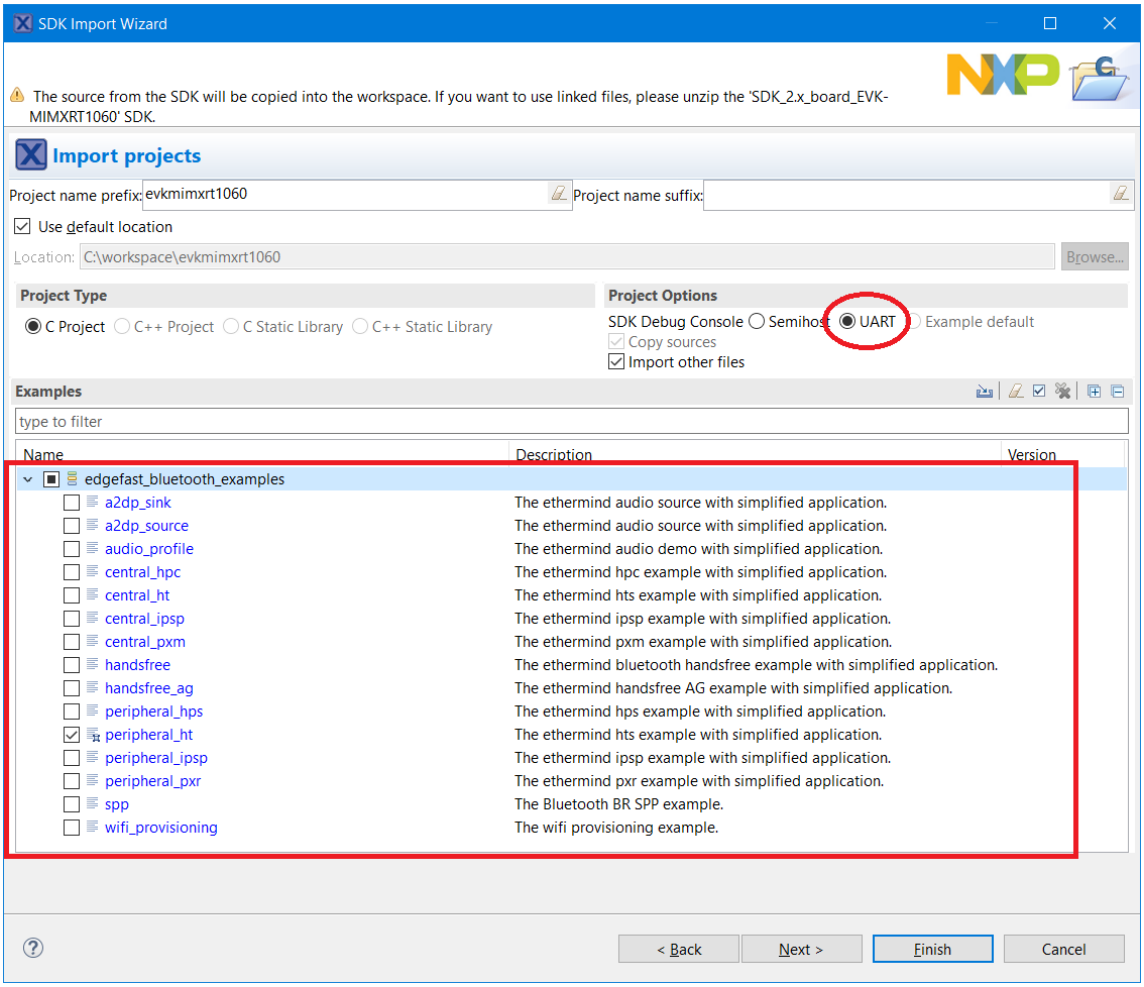
3. After the SDK is loaded successfully, select the **Import the SDK examples(s)...** to add examples to your workspace.



4. Select the evkmimxrt1060 board and click the **Next** button to select the desired example(s).



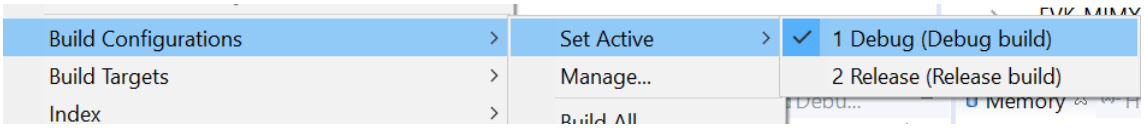
5. Select the evkmimxrt1060 board EdgeFast Bluetooth example. For example, peripheral_ht.
6. Ensure to change SDK debug console from **Semihost** to **UART**.
7. Click **Finish**.



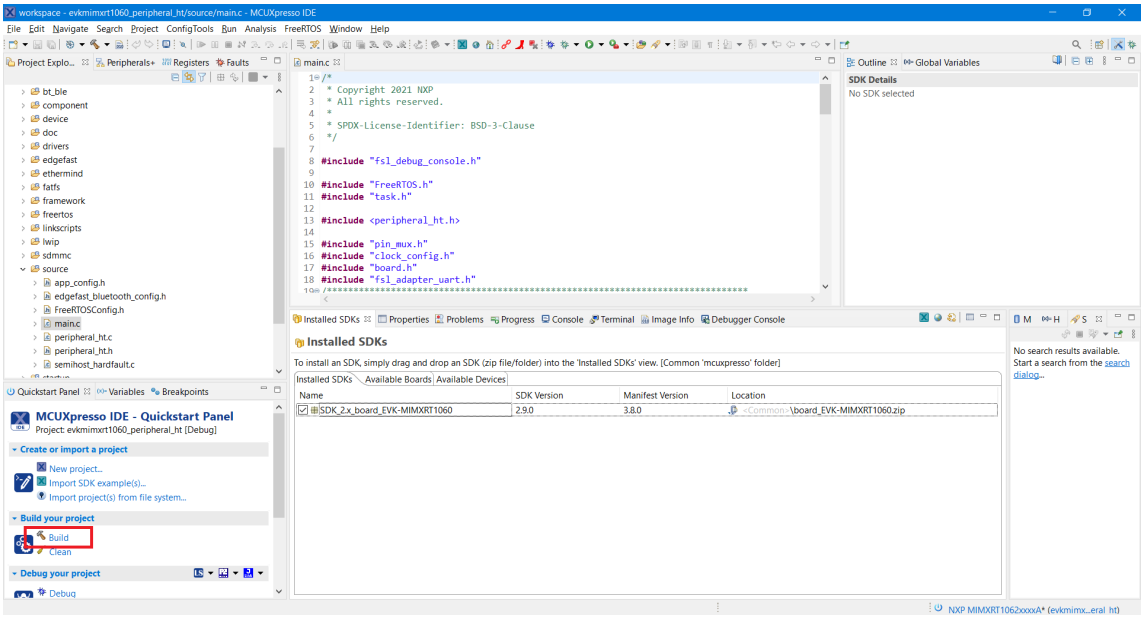
Parent topic: Run a demo application using MCUXpresso IDE

Build an MCUXpresso IDE example

1. Select desired target for your project.



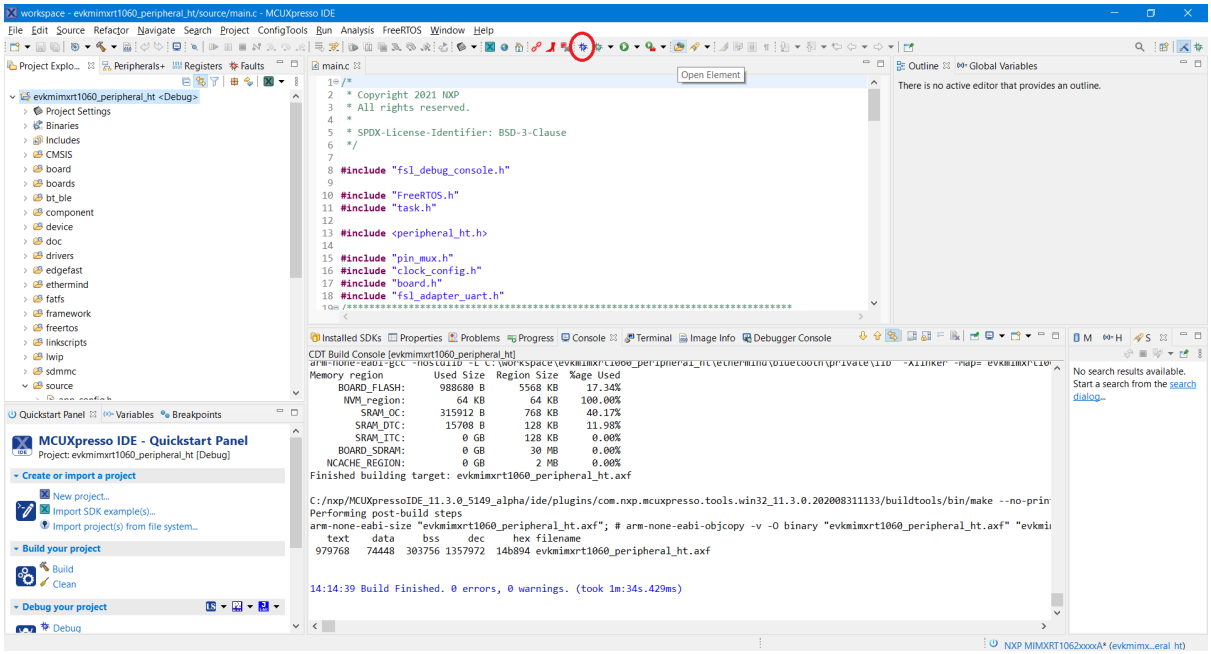
2. Build MCUXpresso IDE EdgeFast Bluetooth Protocol Abstraction Layer project.



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

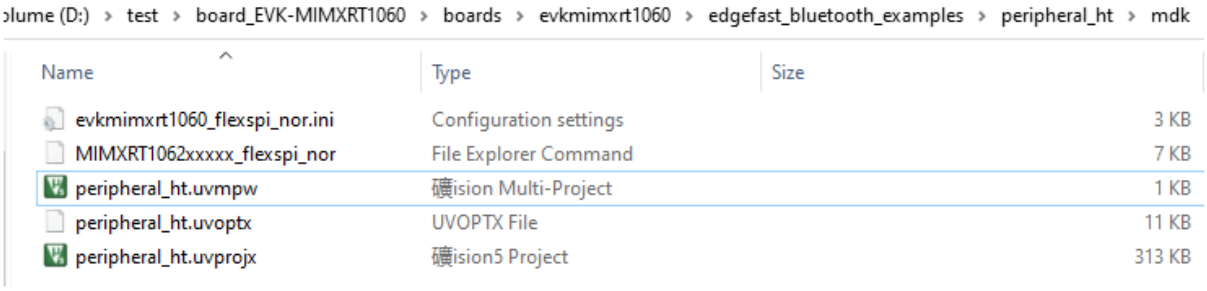
Parent topic:*Demo*

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.

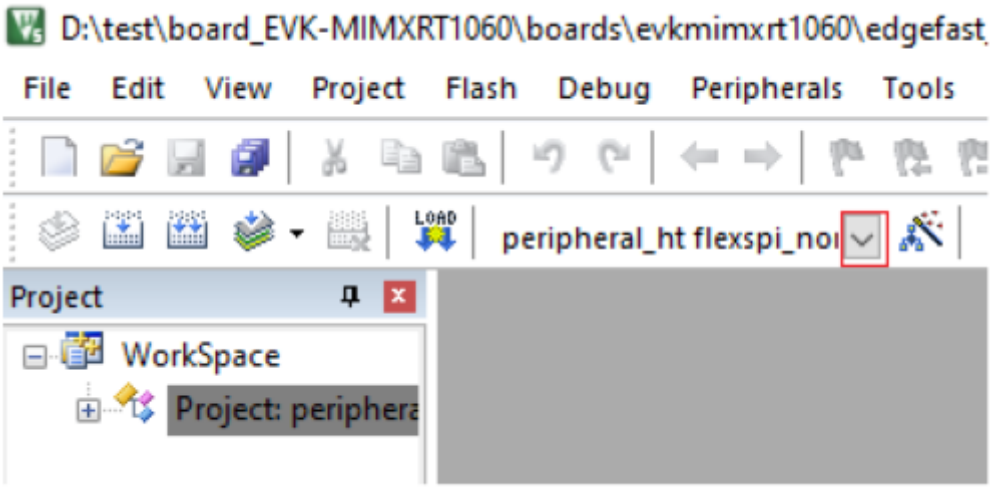
1. The example projects are available at: <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/mdk.
2. Open the mdk workspace file. For example, the highlighted *.uvmpw format file.



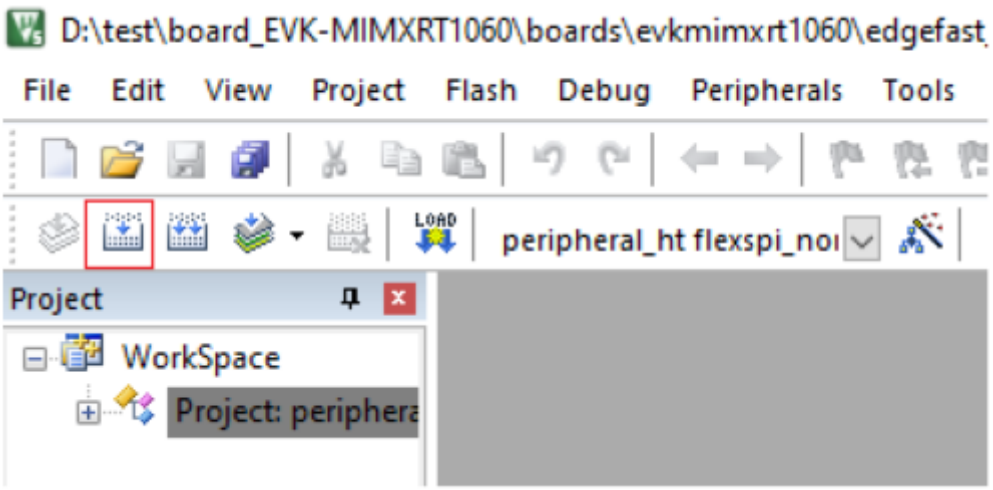
Parent topic: Run a demo application using MDK

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.



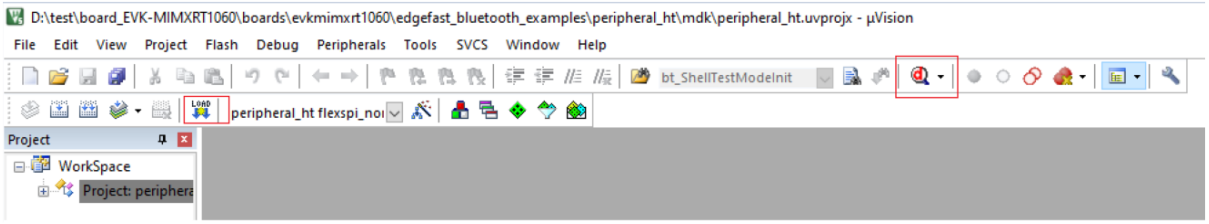
2. Click the highlighted icon to build the EdgeFast Bluetooth Protocol Abstraction Layer project.



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*.

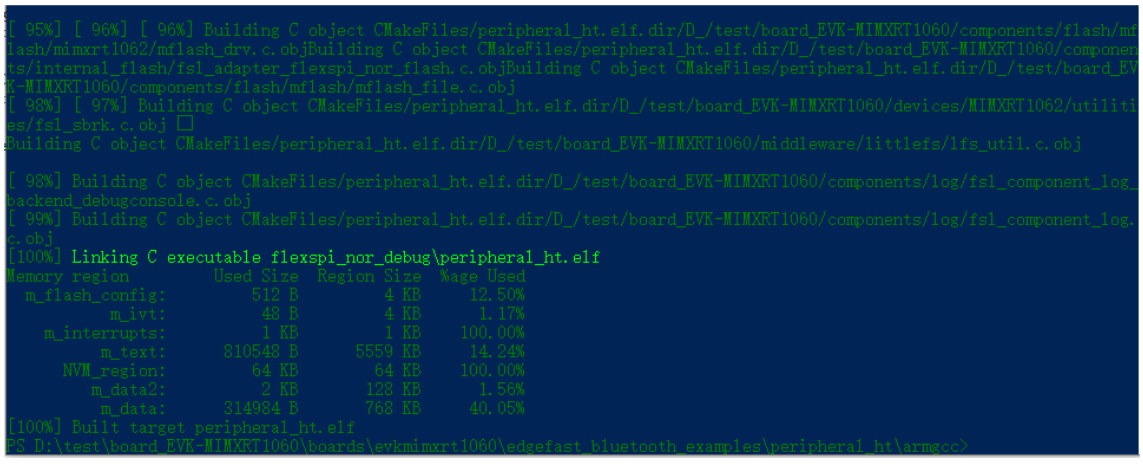
Parent topic: Run a demo application using Arm GCC

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

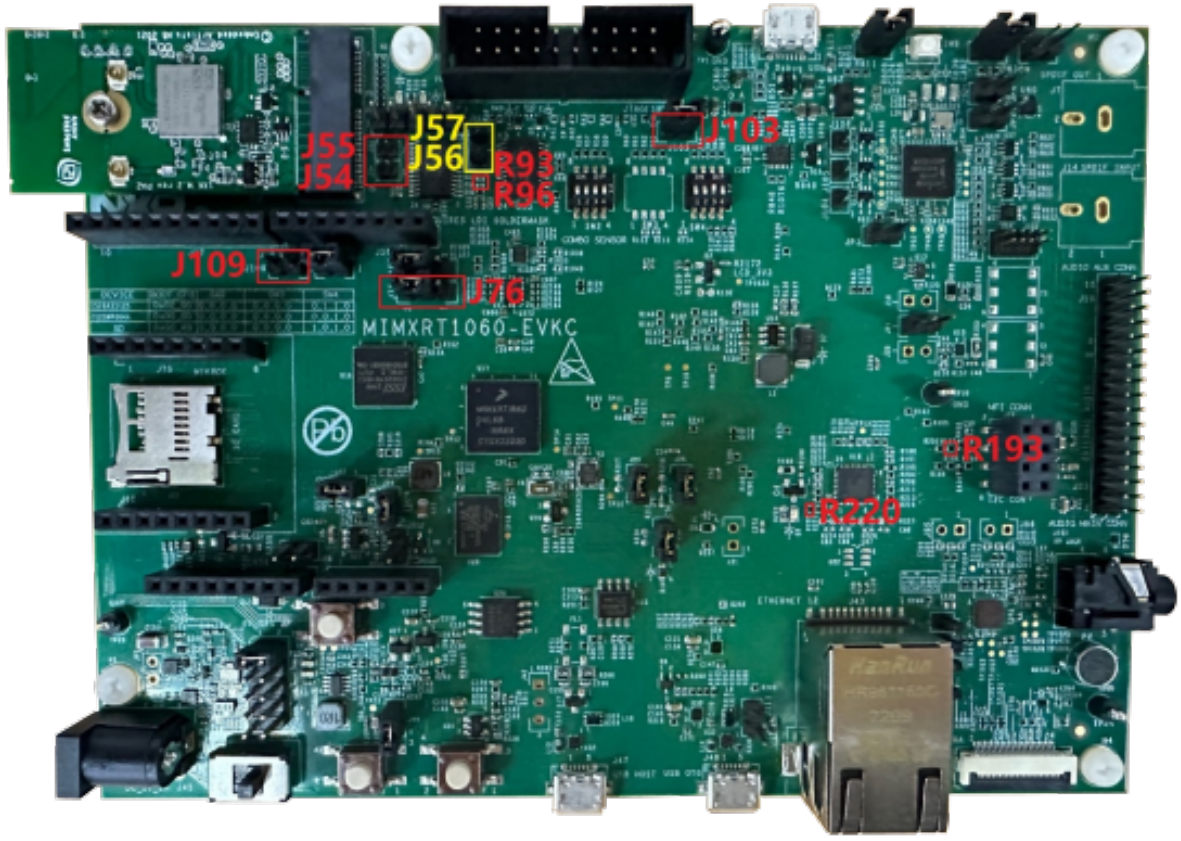
- **HCI UART rework**

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

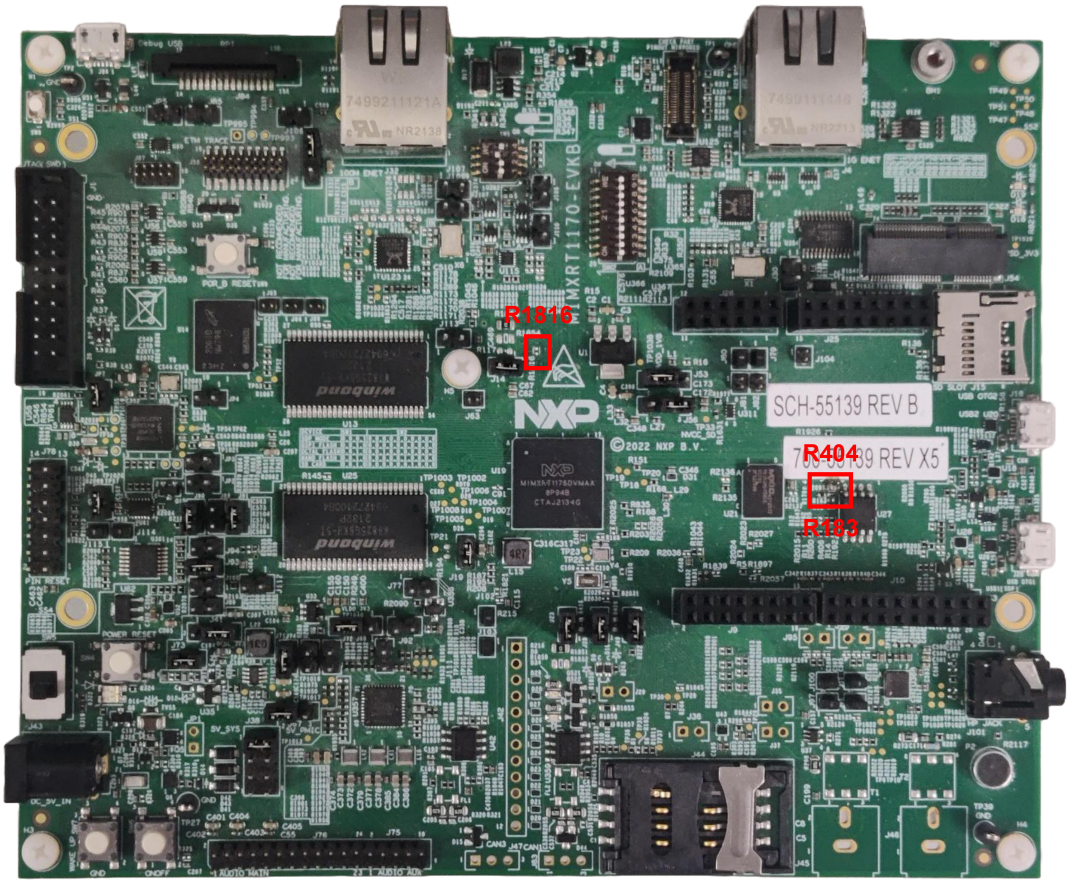
This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

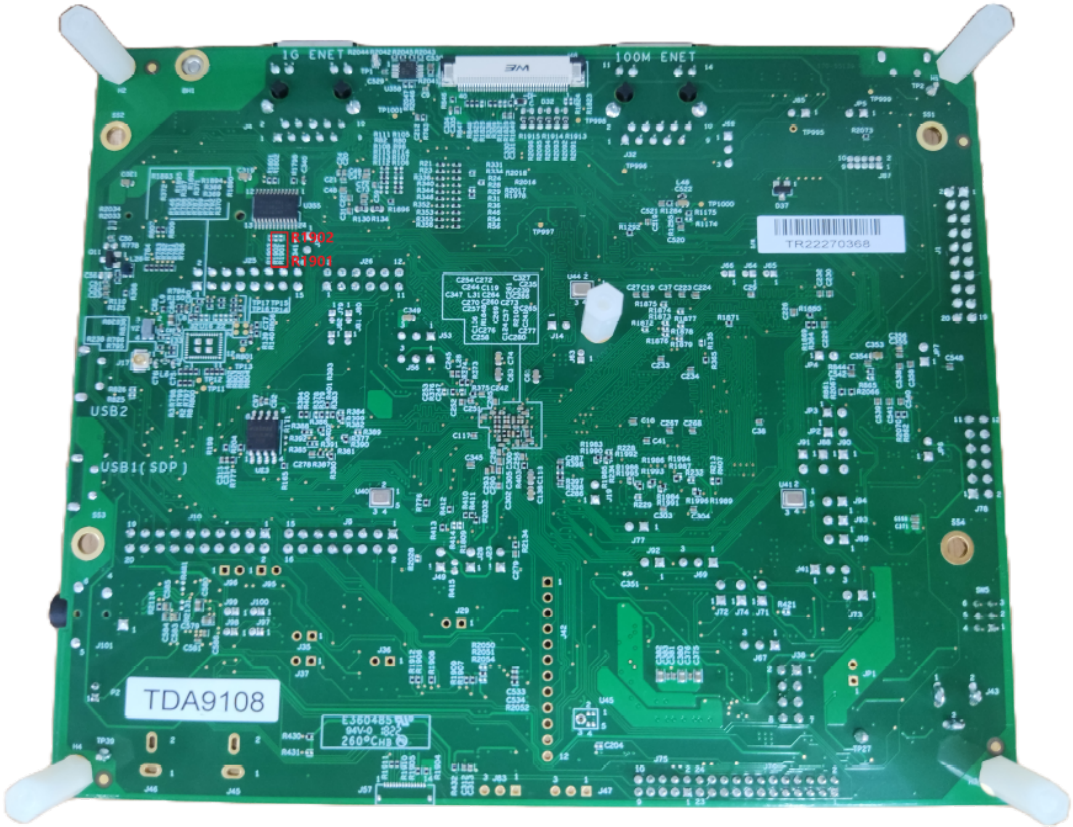
The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

Hardware rework

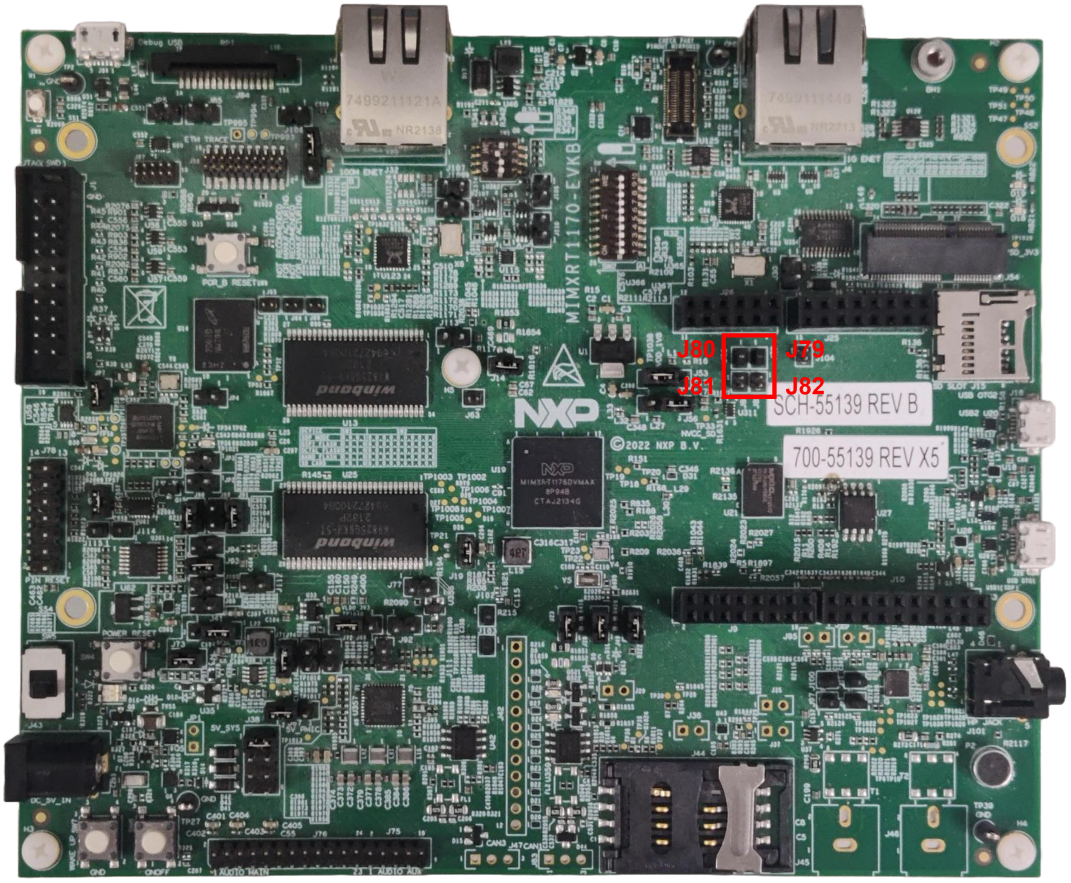
- **HCI UART rework**
  1. Remove resistors R183 and R1816.
  2. Solder 0 ohm resistor to R404, R1901, and R1902.

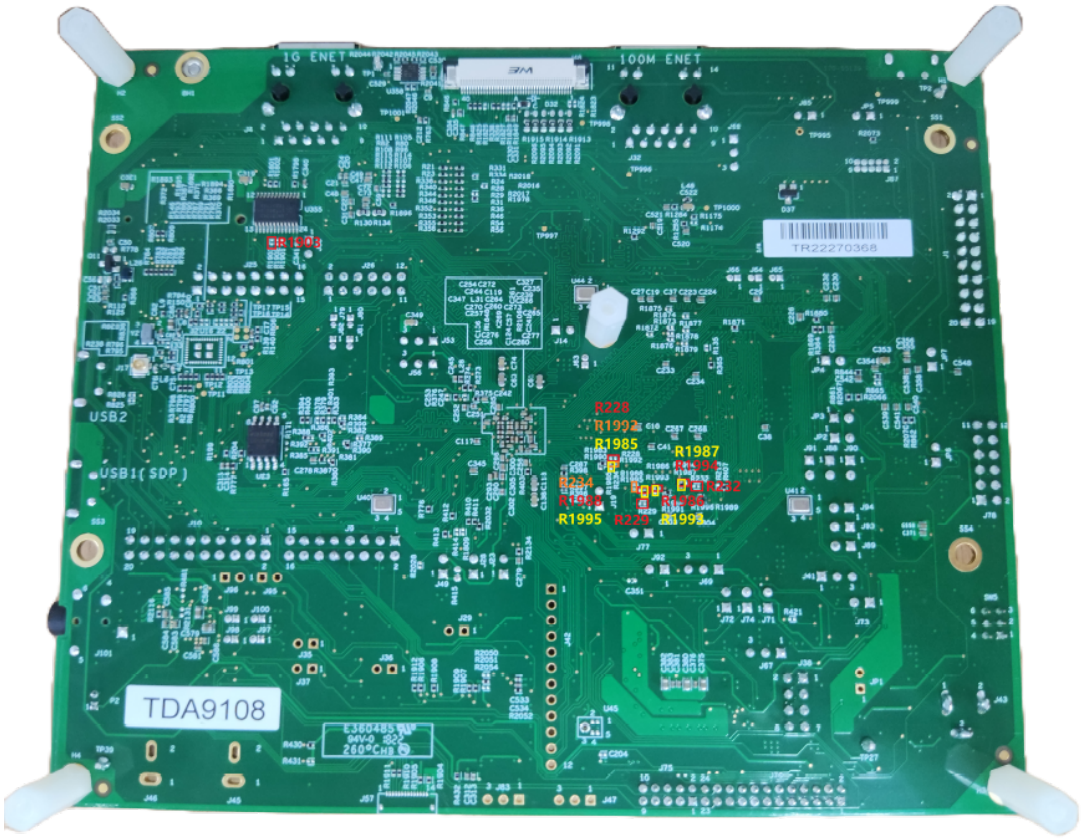




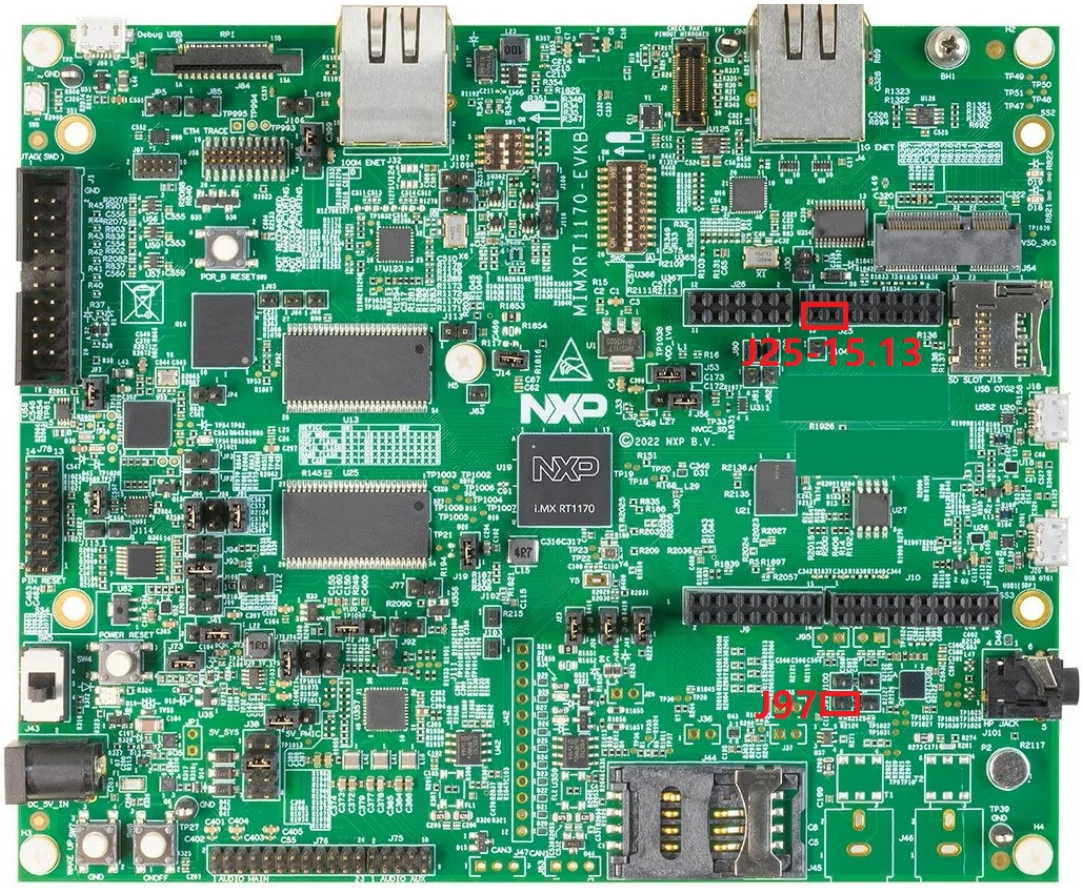
- **PCM interface rework**
  1. Disconnect header J79 and J80.

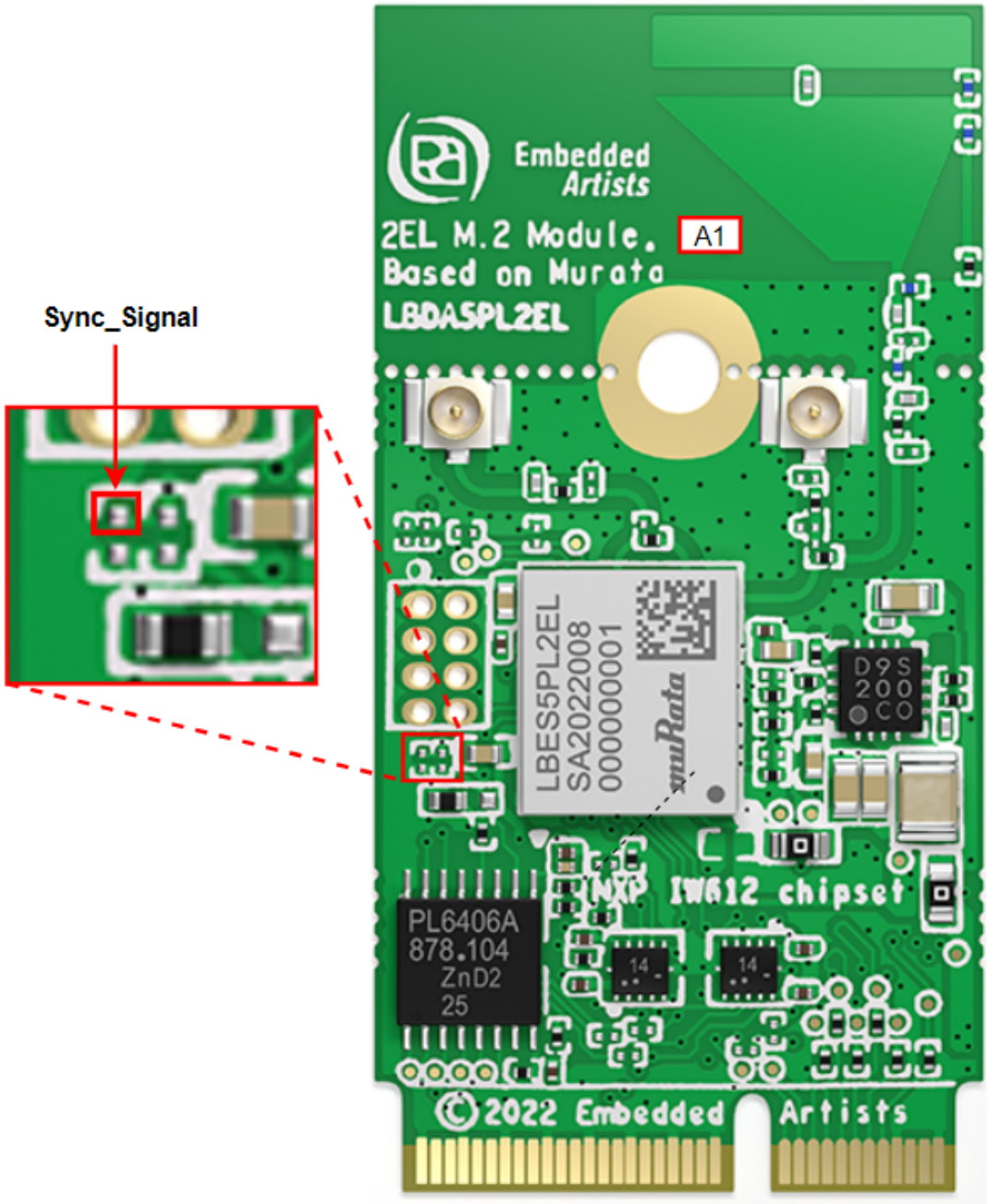
2. Connect header J81 and J82.
3. Remove resistors R1985, R1986, R1987, R1988, R1992, R1993, R1994, and R1995.
4. Solder 0 ohm resistor to R228, R229, R232, R234, and R1903.





- 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


J25 15.13
J97
MIMXRT1170-EVKB
i.MX RT1170
2022 NXP B.V.



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:

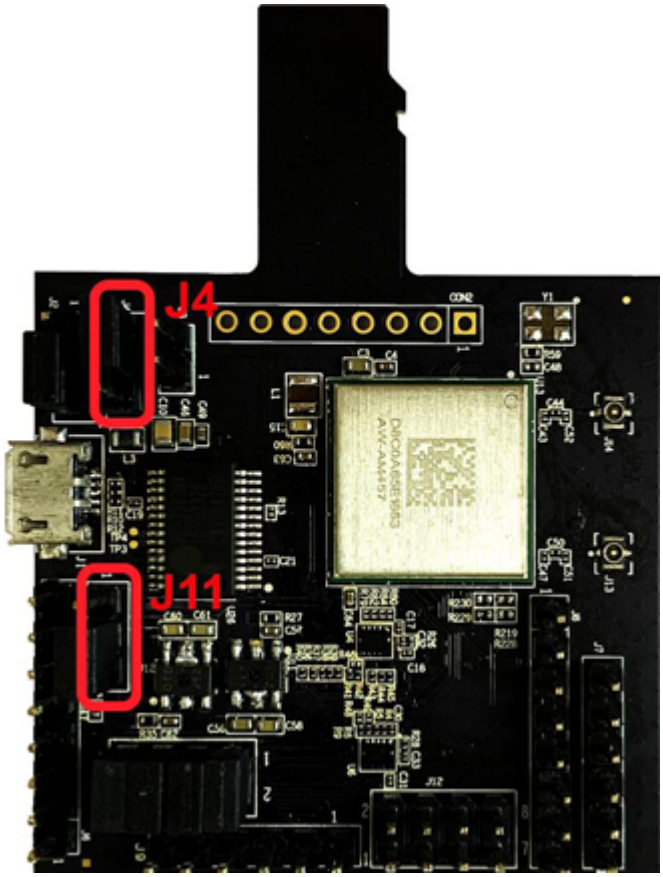
- HCI UART rework

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

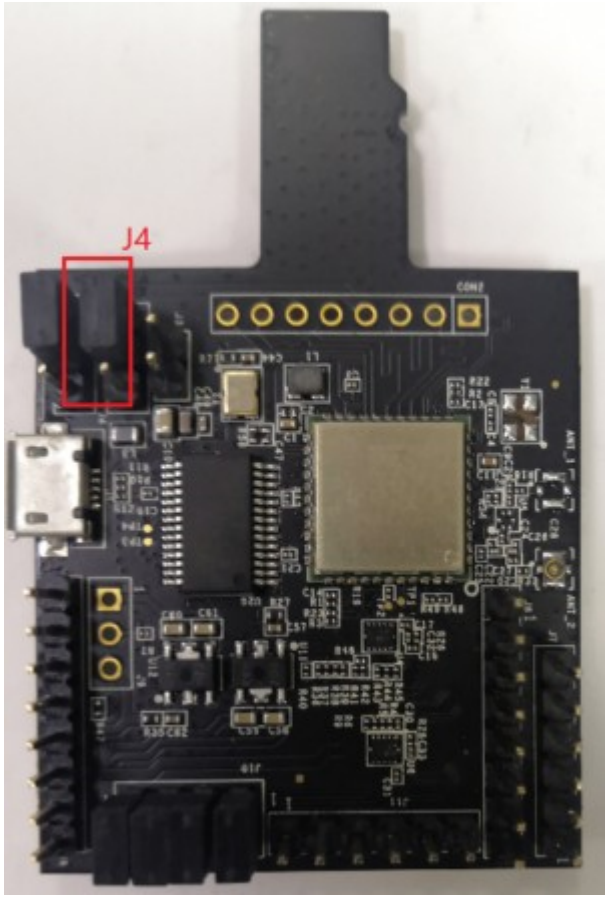
Hardware rework **HCI UART rework**

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:

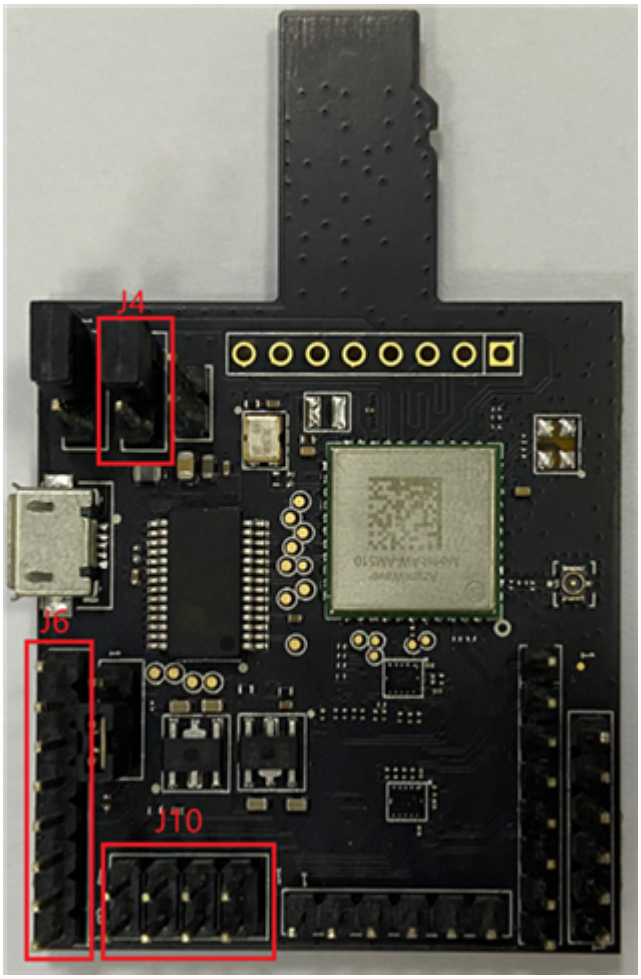
- HCI UART rework

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



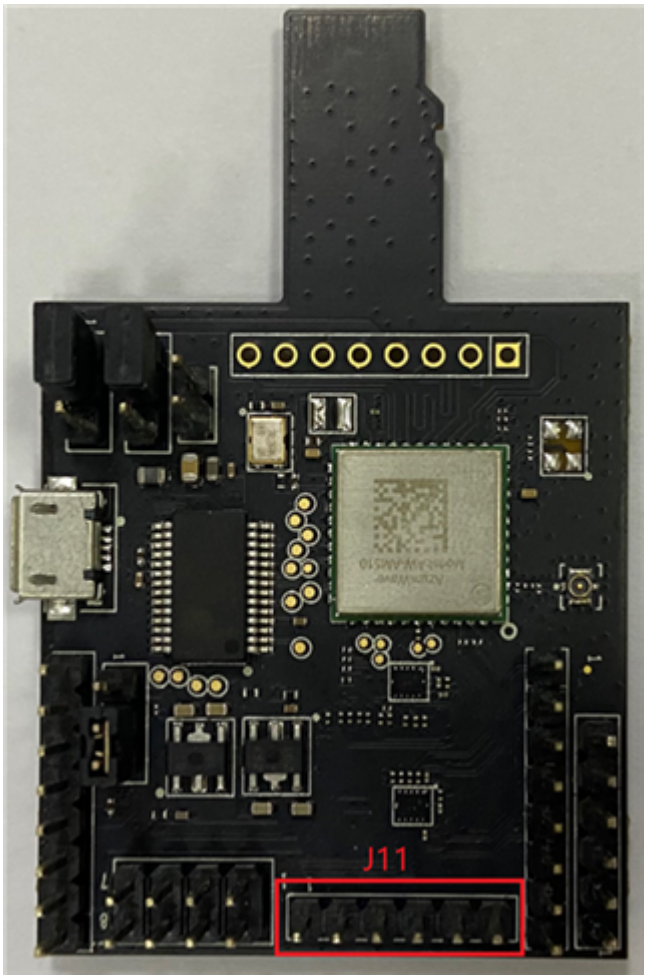
Jumper Setting:

– 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.

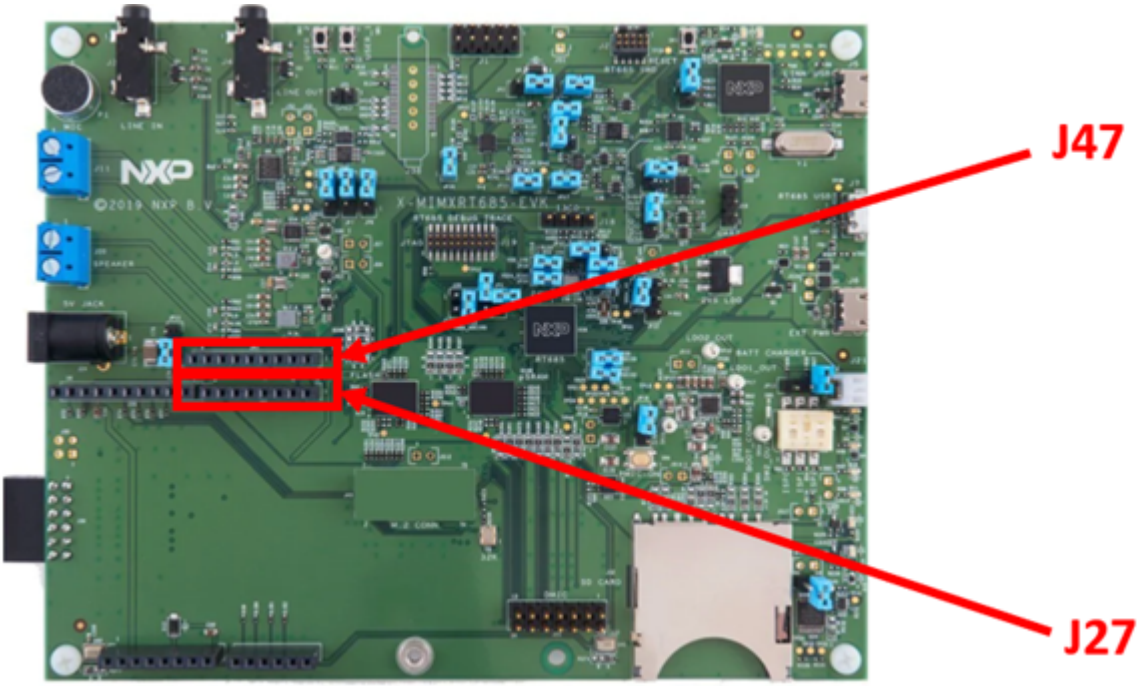
The hardware rework has one part:

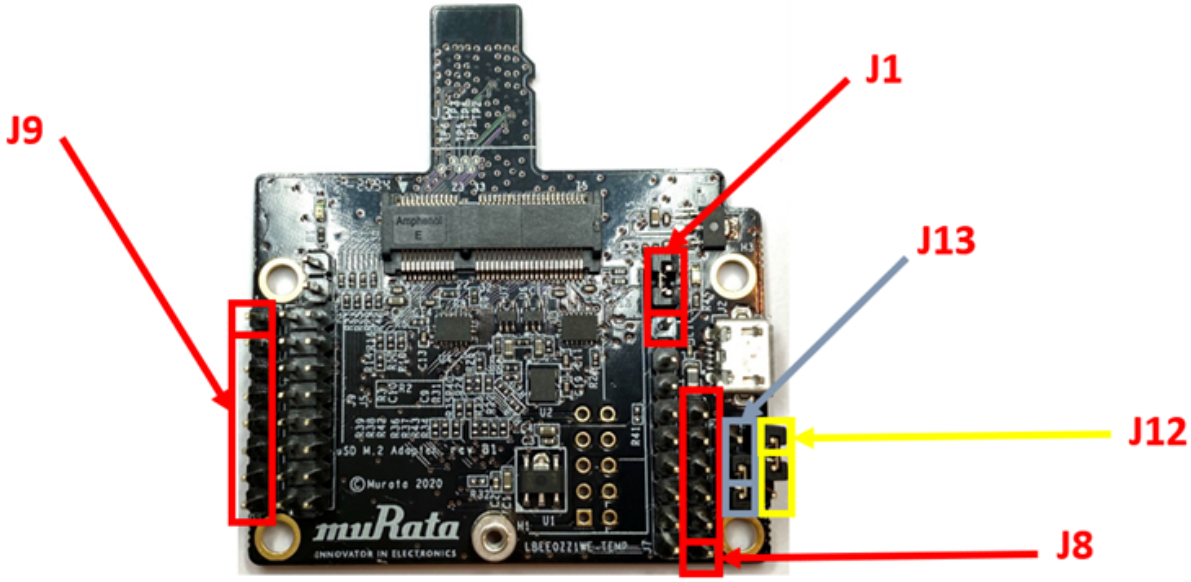
- HCI UART rework

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:

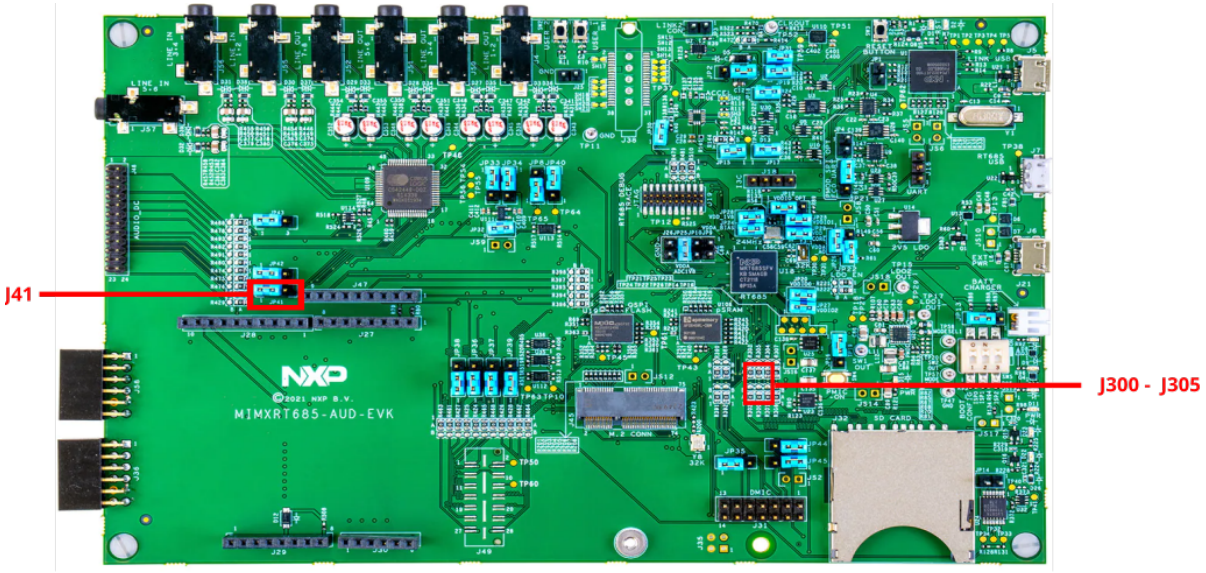
- HCI UART rework

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**

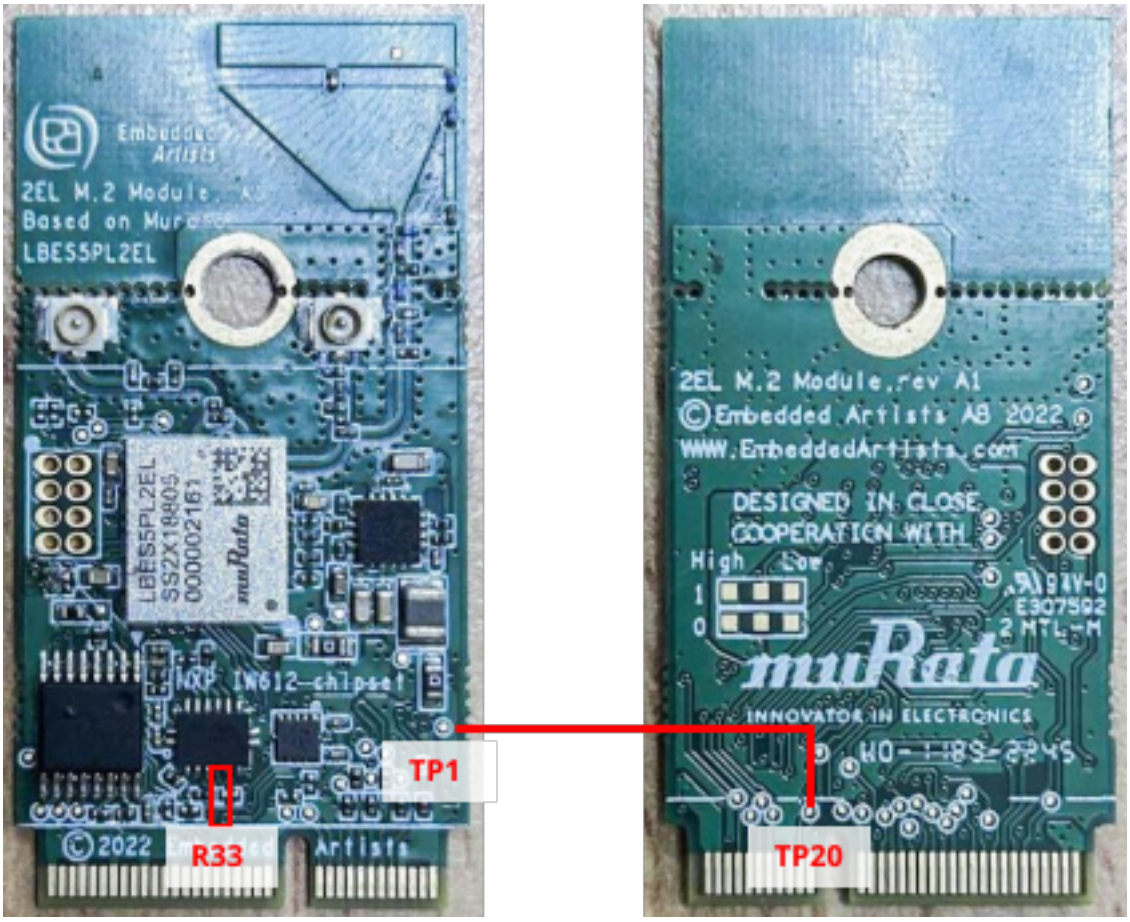
  For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

- **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.

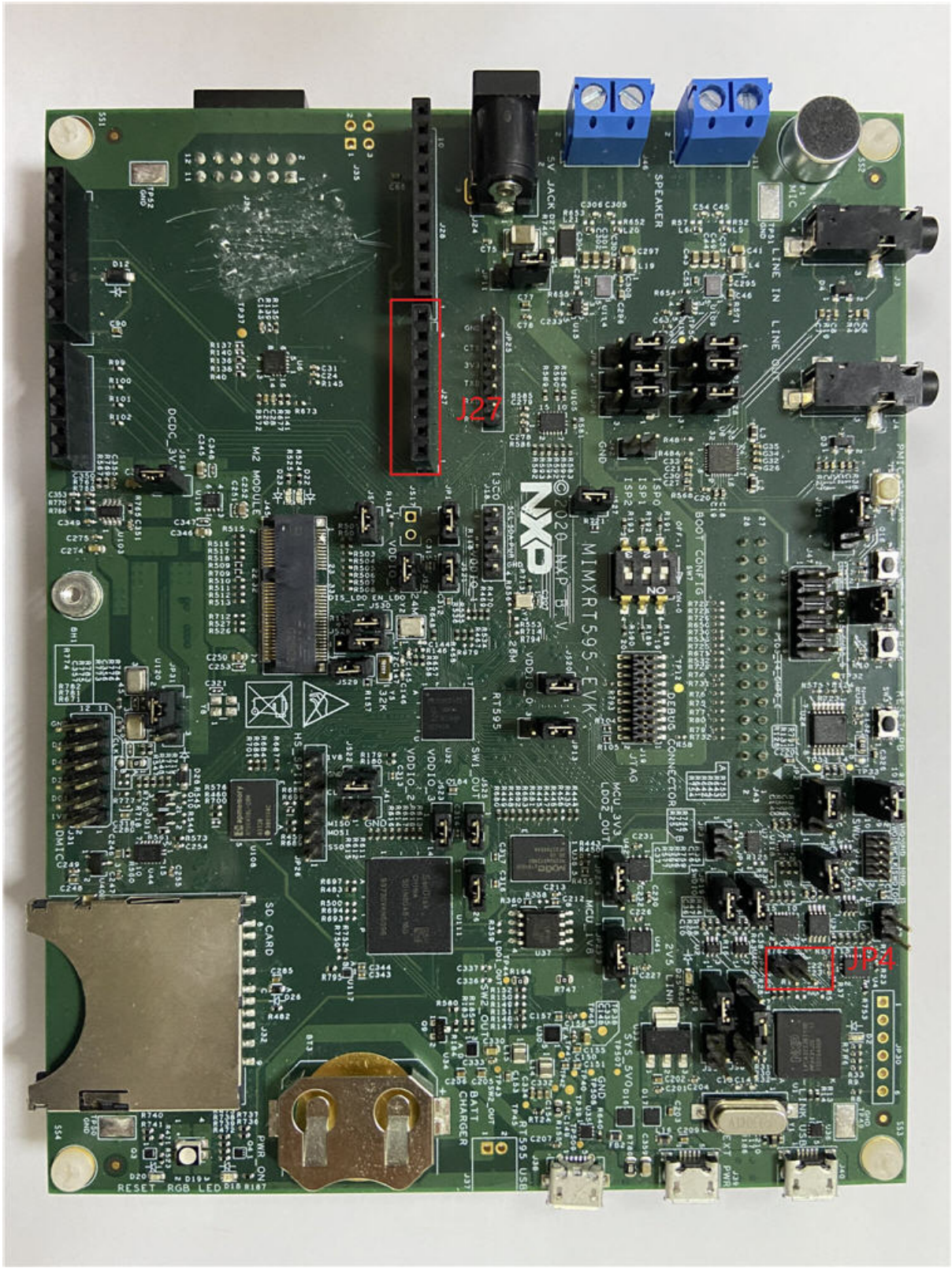
The hardware rework has one part:

- 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:

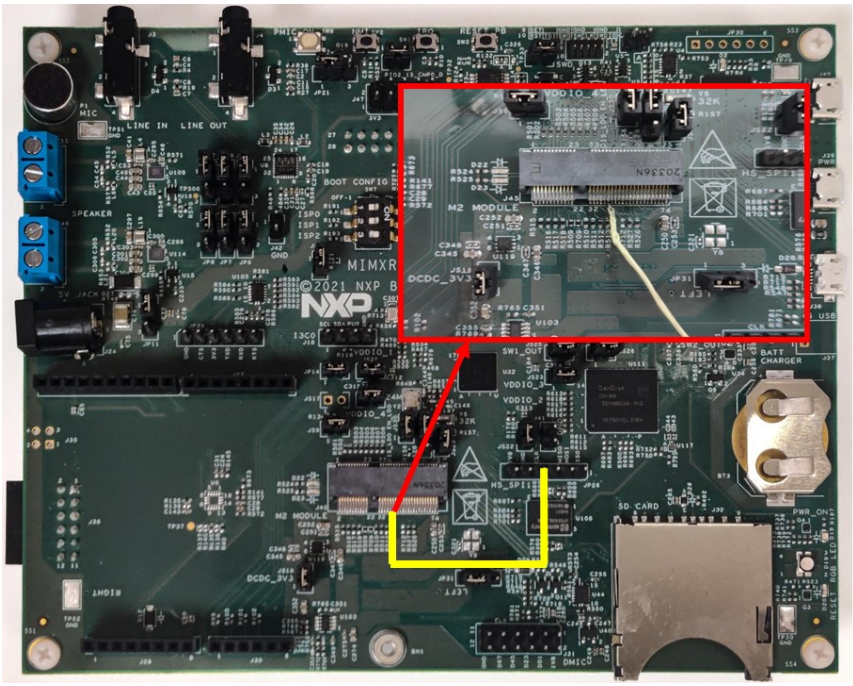
- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework **Debug console serial rework:**

For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

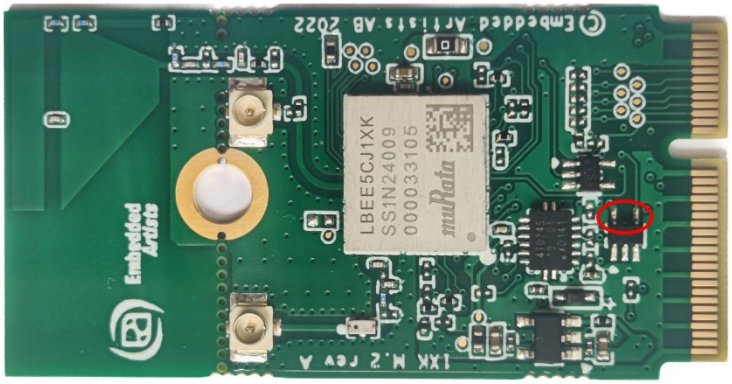
Host wake-up controller pin rework:

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.



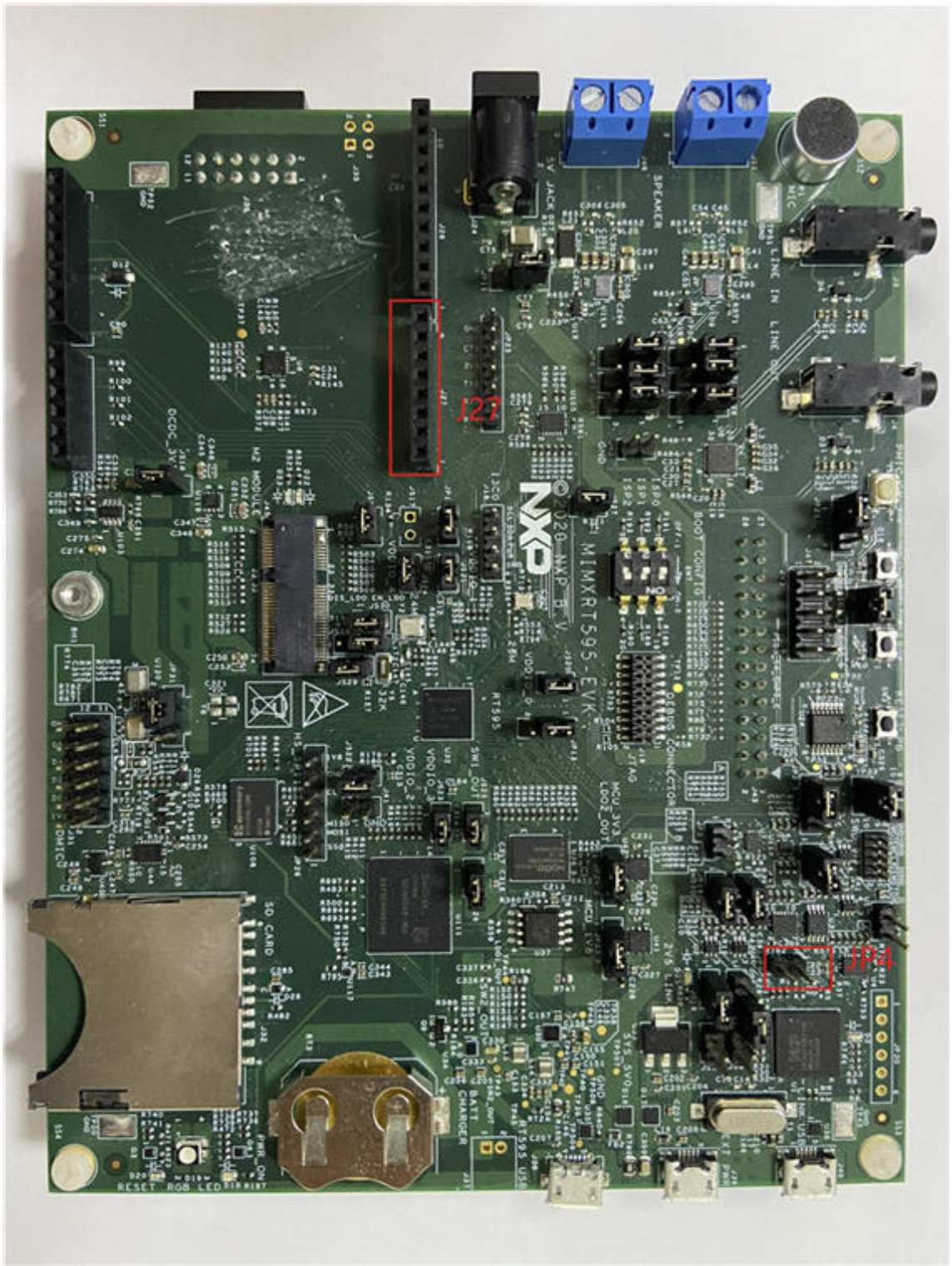
Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA*

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

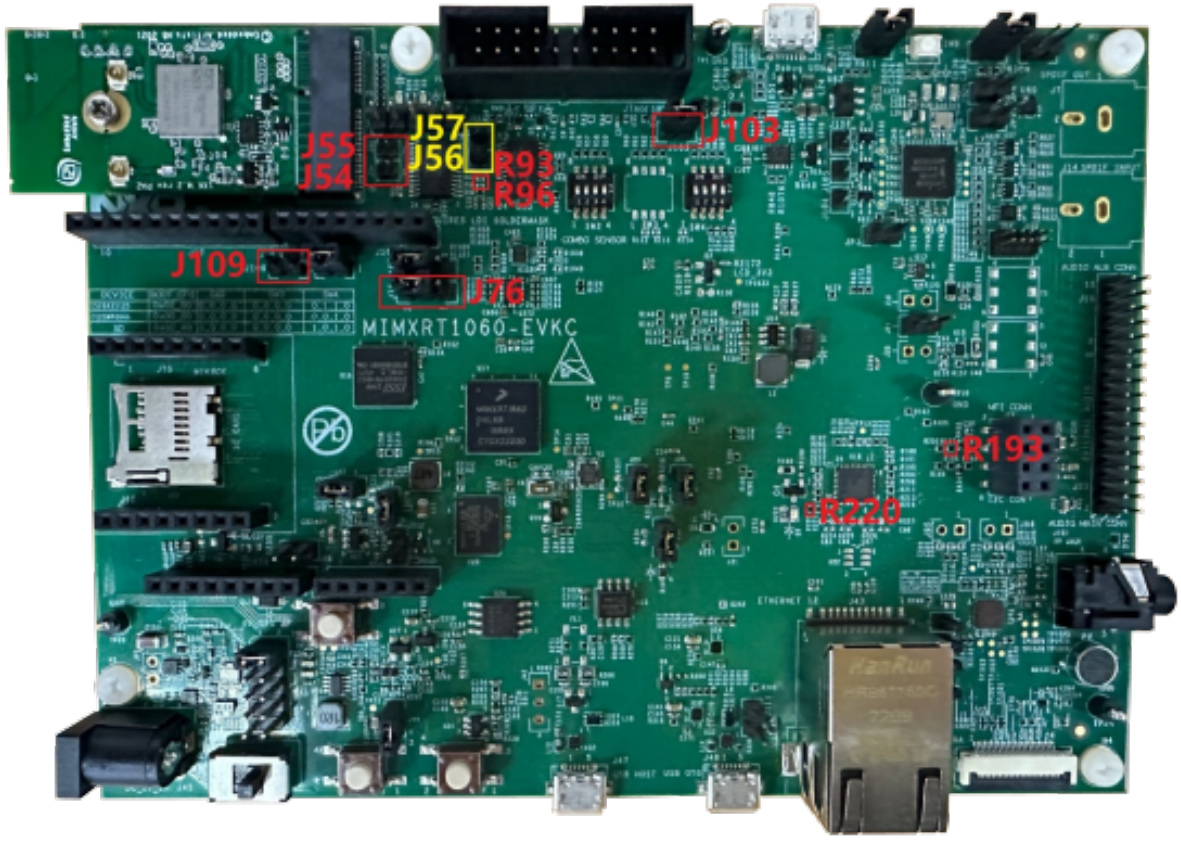
- **HCI UART rework**

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

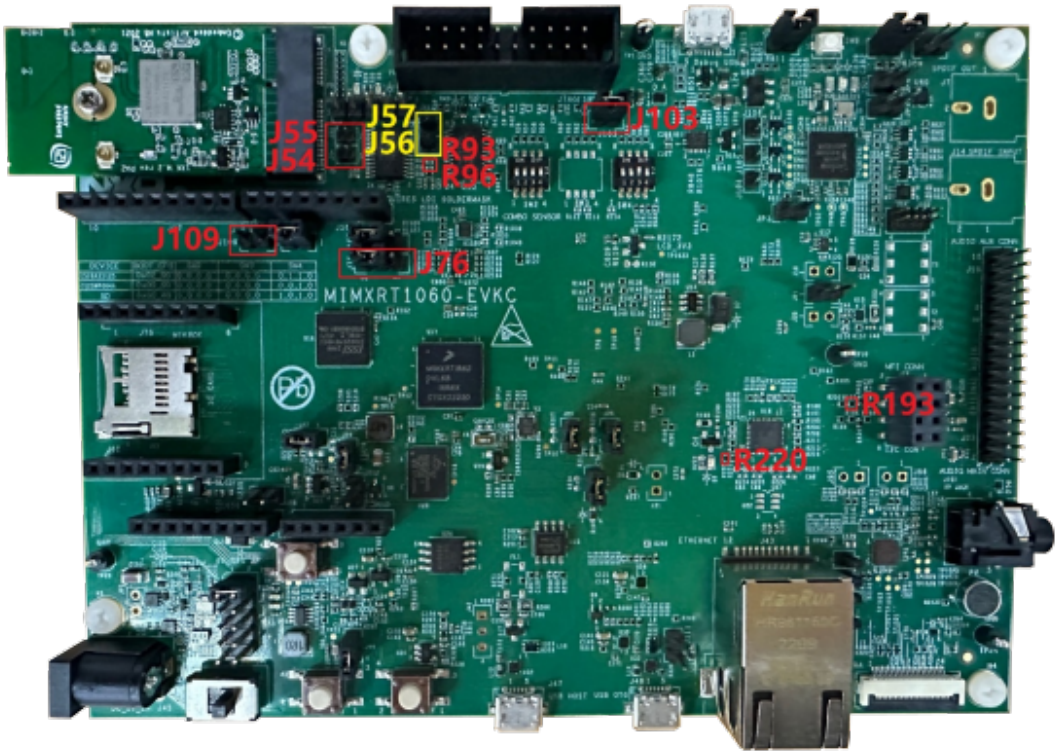
- HCI UART rework

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

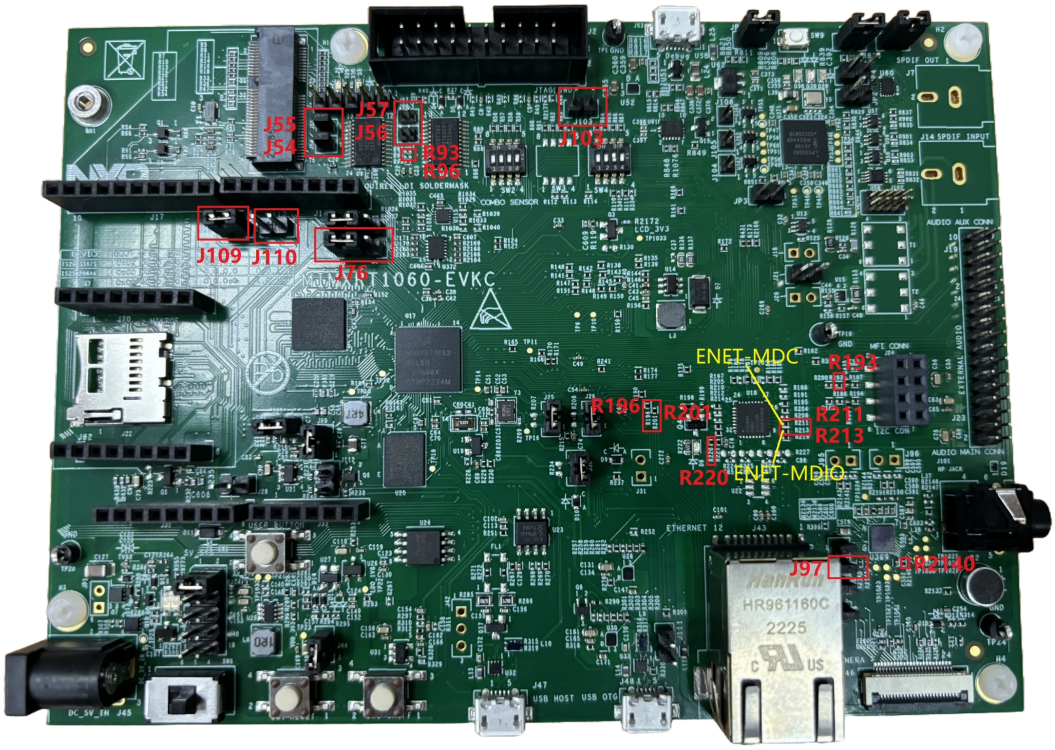
- PCM interface rework

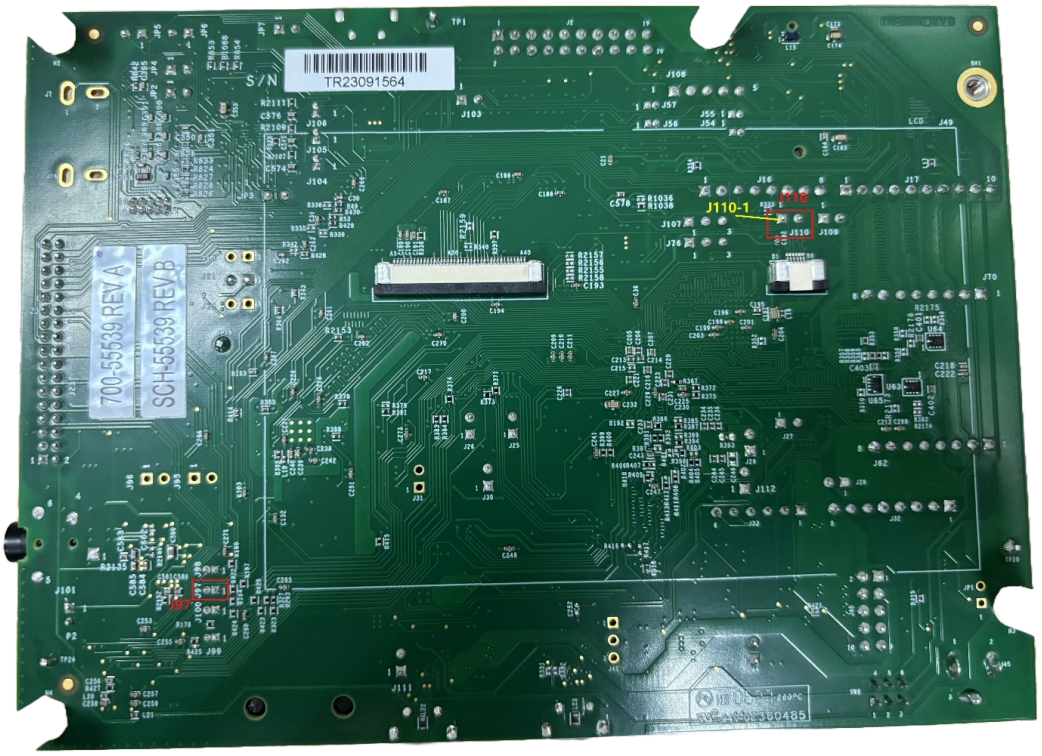
1. Remove J54 and J55, connect J56, and J57.
2. Remove R220.
3. Connect J103.

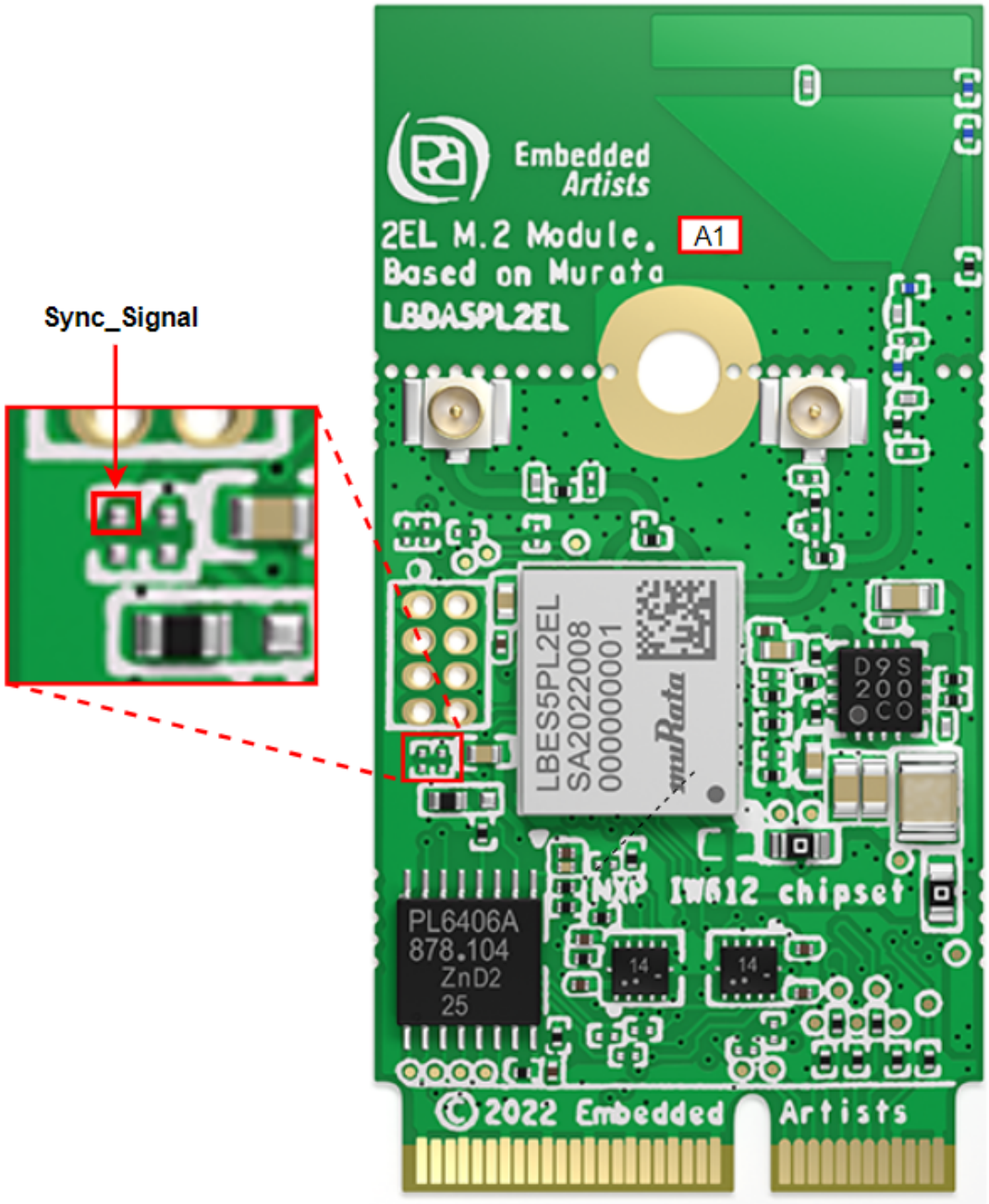
   Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



- 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.

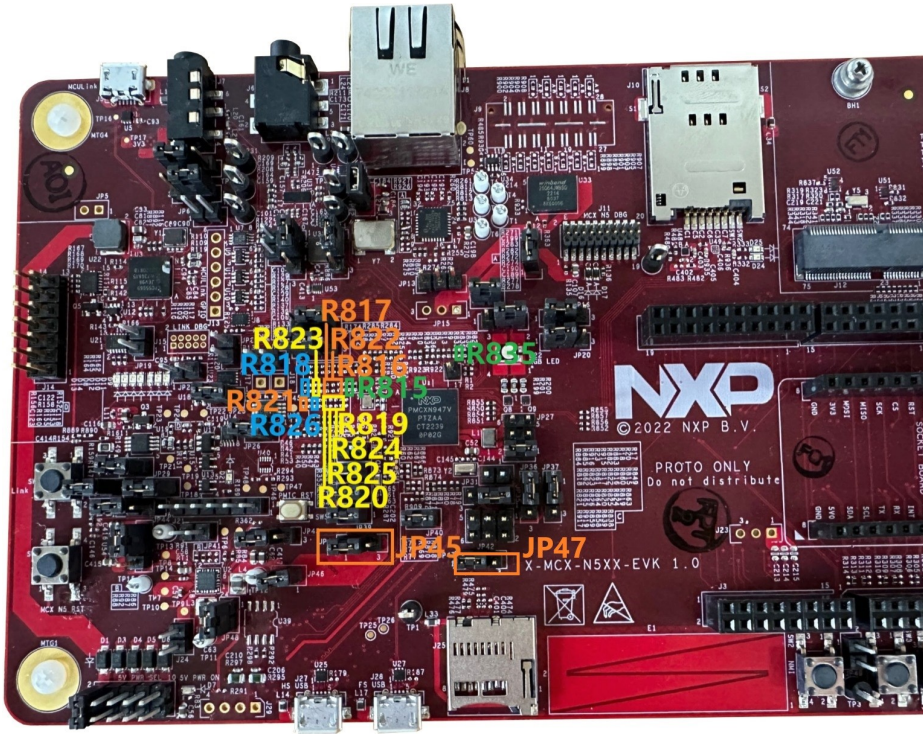
The hardware rework consists of two parts:

- 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.

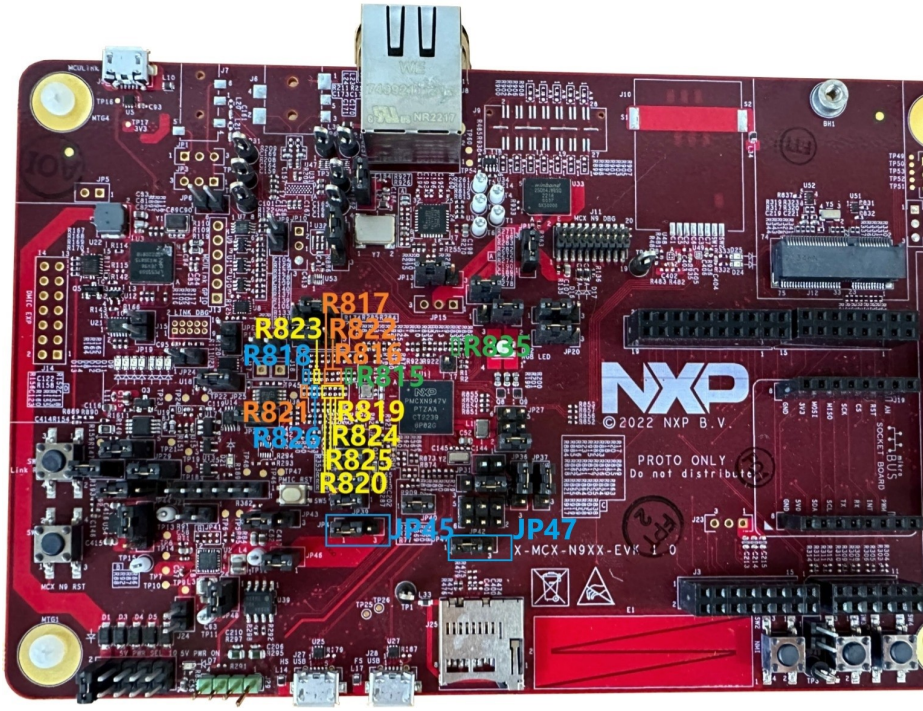
The hardware rework consists of two parts:

- 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*

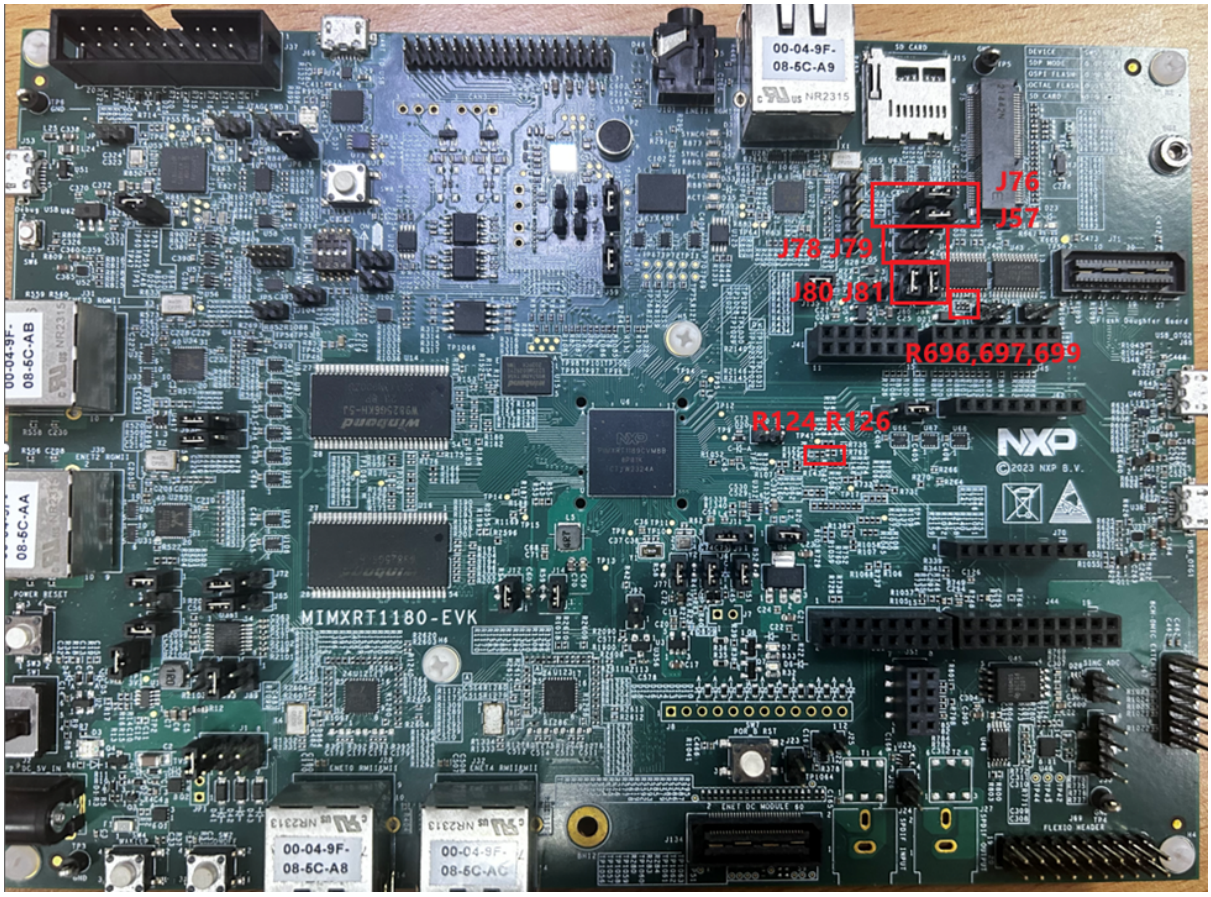
Hardware Rework Guide for MIMXRT1180 and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MIMXRT1180 board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) 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



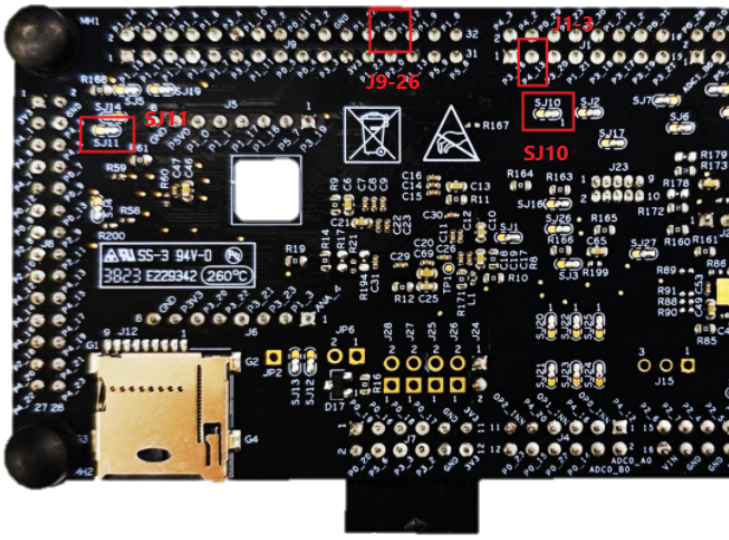
Hardware Rework Guide for FRDM-MCXN947 and X-FRDM-WIFI-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and X-FRDM-WIFI-M.2 or the Murata's 2LL EAR00500 (2LL) M.2 modules solution.

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



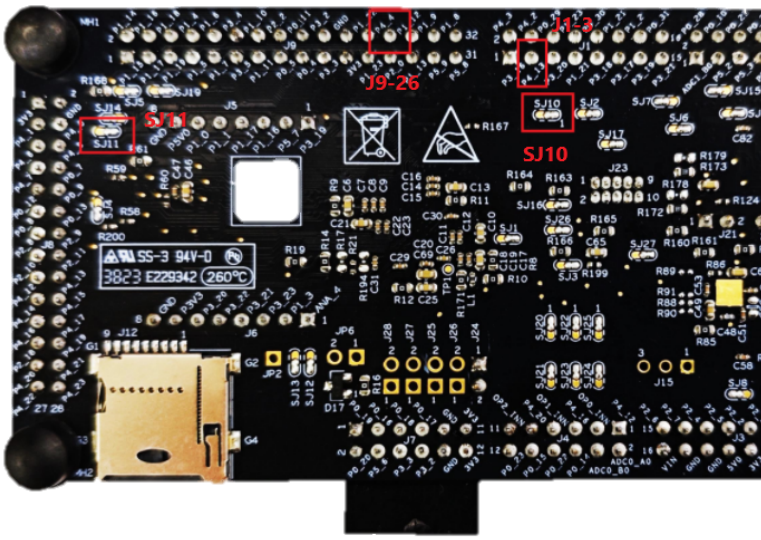
– Connect J18(On X-FRDM-WIFI-M.2) for 3.3V

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:

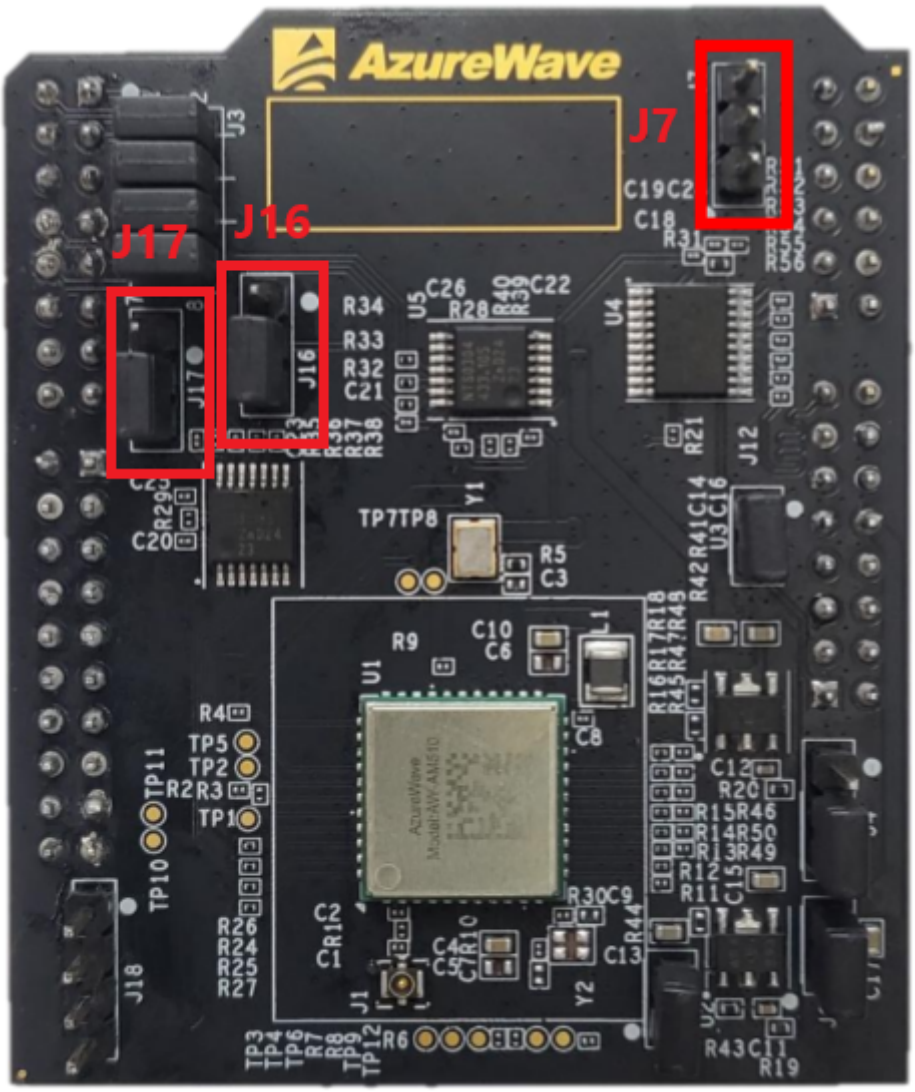
- UART interface rework
- FRDM-IW416-AW-AM510

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.

```
#define BOARD_FLASH_SIZE (0x800000U)
```

```
#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.

```
define symbol m_interrupts_start       = 0x60002000;
define symbol m_interrupts_end         = 0x600023FF;

define symbol m_text_start             = 0x60002400;
define symbol _ROM_END_                = 0x6057FFFF;

define exported symbol m_boot_hdr_conf_start = 0x60000000;
define symbol m_boot_hdr_ivt_start           = 0x60001000;
define symbol m_boot_hdr_boot_data_start     = 0x60001020;
define symbol m_boot_hdr_dcd_data_start      = 0x60001030;

BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash secto
BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x60800000 - EDGEFAST_BT_
***/
```

```
define symbol m_interrupts_start       = 0x70002000;
define symbol m_interrupts_end         = 0x700023FF;

define symbol m_text_start             = 0x70002400;
define symbol _ROM_END_                = 0x7017FFFF;

define exported symbol m_boot_hdr_conf_start = 0x70000000;
define symbol m_boot_hdr_ivt_start           = 0x70001000;
define symbol m_boot_hdr_boot_data_start     = 0x70001020;
define symbol m_boot_hdr_dcd_data_start      = 0x70001030;

BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash sect
BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x70400000 - EDGEFAST_BT
***/
```

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.

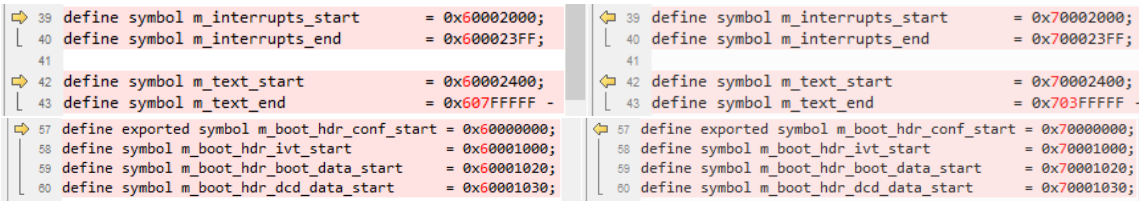
Parent topic:*Migrate examples from RT1060EVKC to RT1064*

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

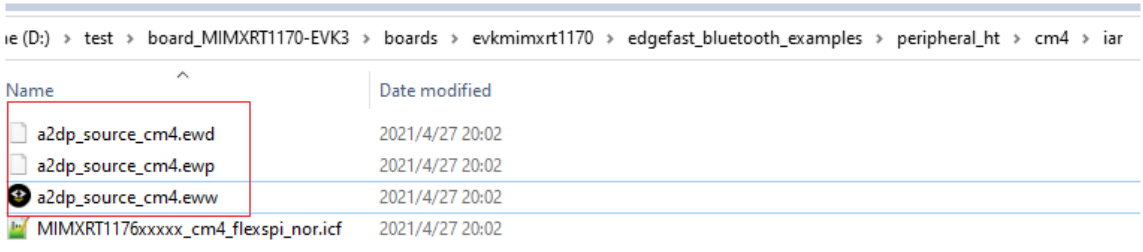
4. Adjust project settings
5. Delete function

In this document, the peripheral_ht example is used to demonstrate how to enable EdgeFast examples on M4 core with IAR and ARMGCC.

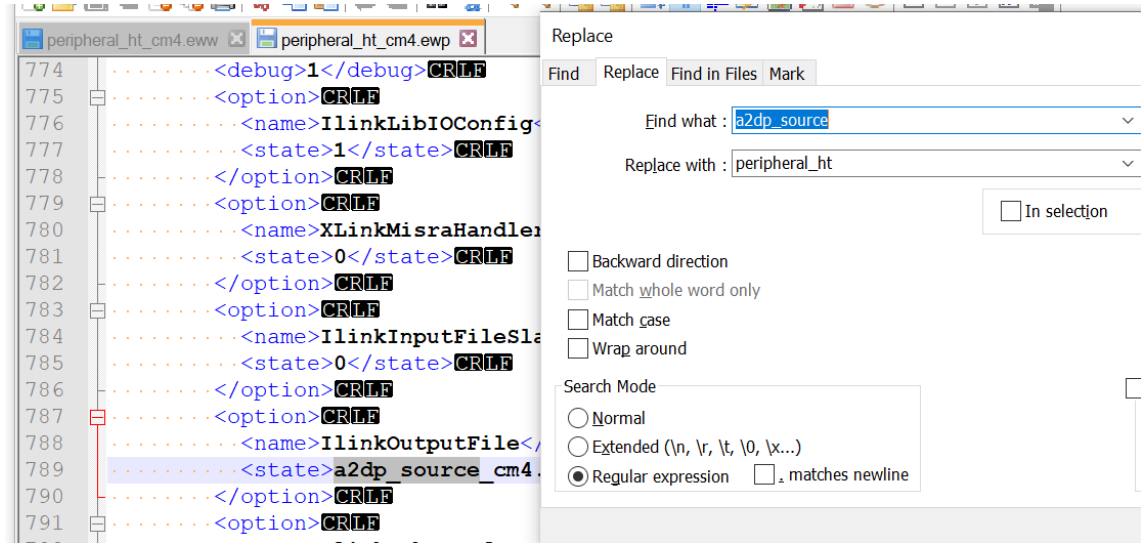
IAR This section describes the steps to create an M4 project with IAR, rearrange source and project files, adjust project settings, and delete function.

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.



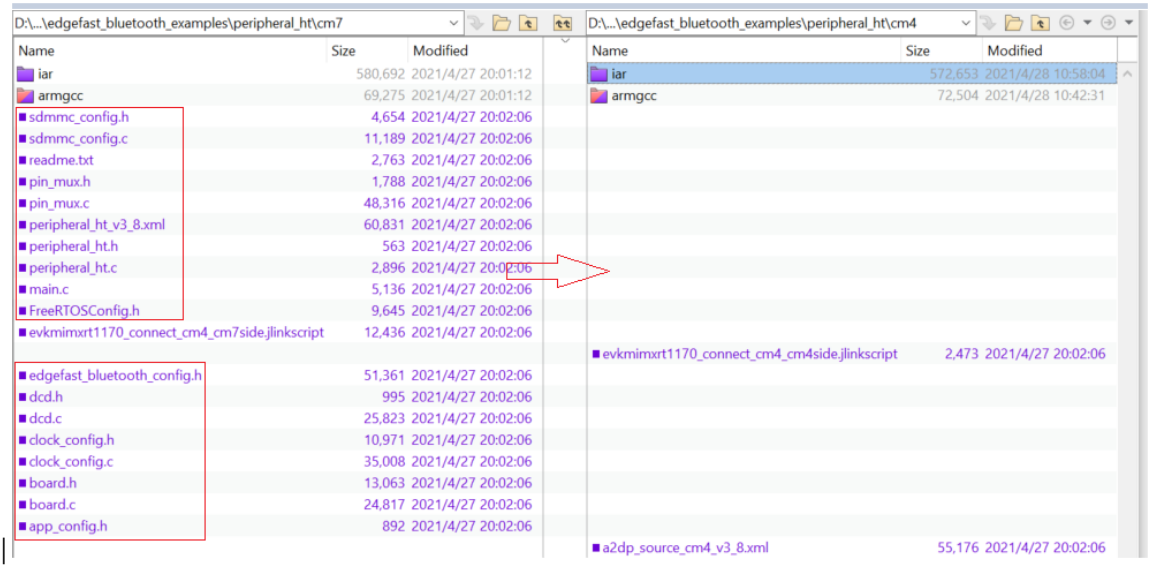
4. Open the files *peripheral_ht_cm4.eww* and *peripheral_ht_cm4.ewp* with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio Code.
5. Search and replace all *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.



Parent topic:*IAR*

Rearrange source files To rearrange source files, perform the following steps:

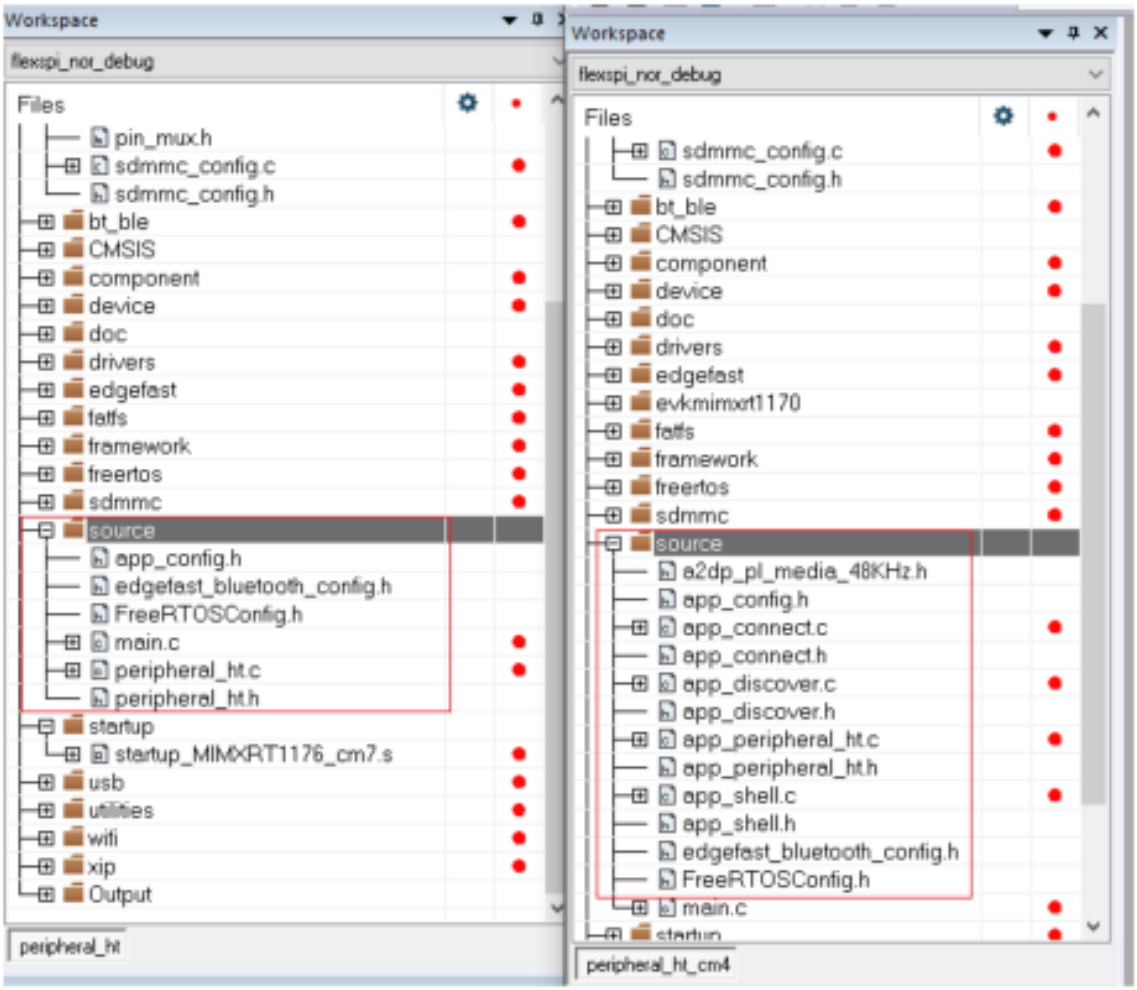
1. Open the folder *cm4* in the directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peri* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h from the folder *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm7*| to the folder <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*.



Parent topic:*IAR*

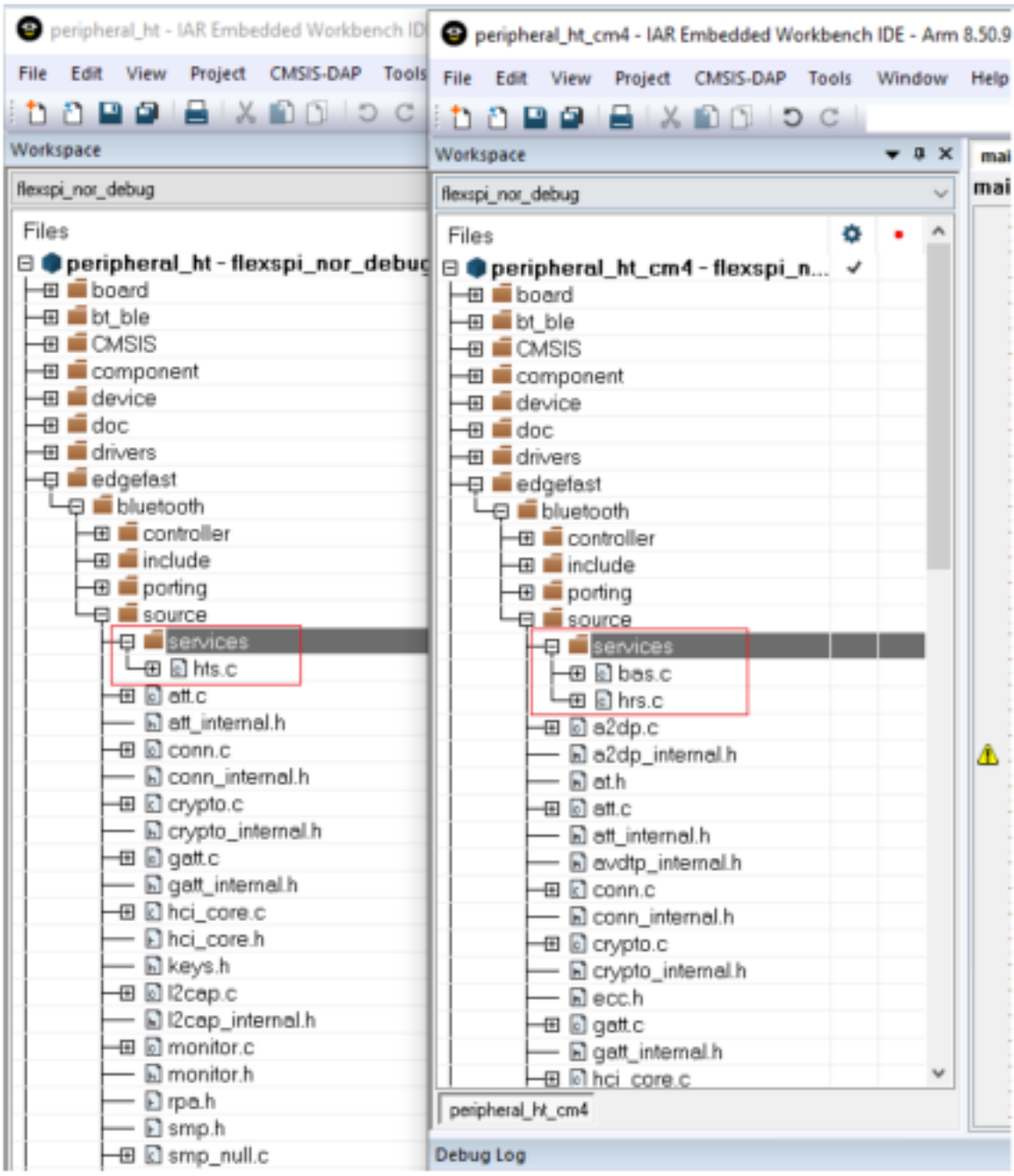
Rearrange project files To rearrange project files, perform the following steps:

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.

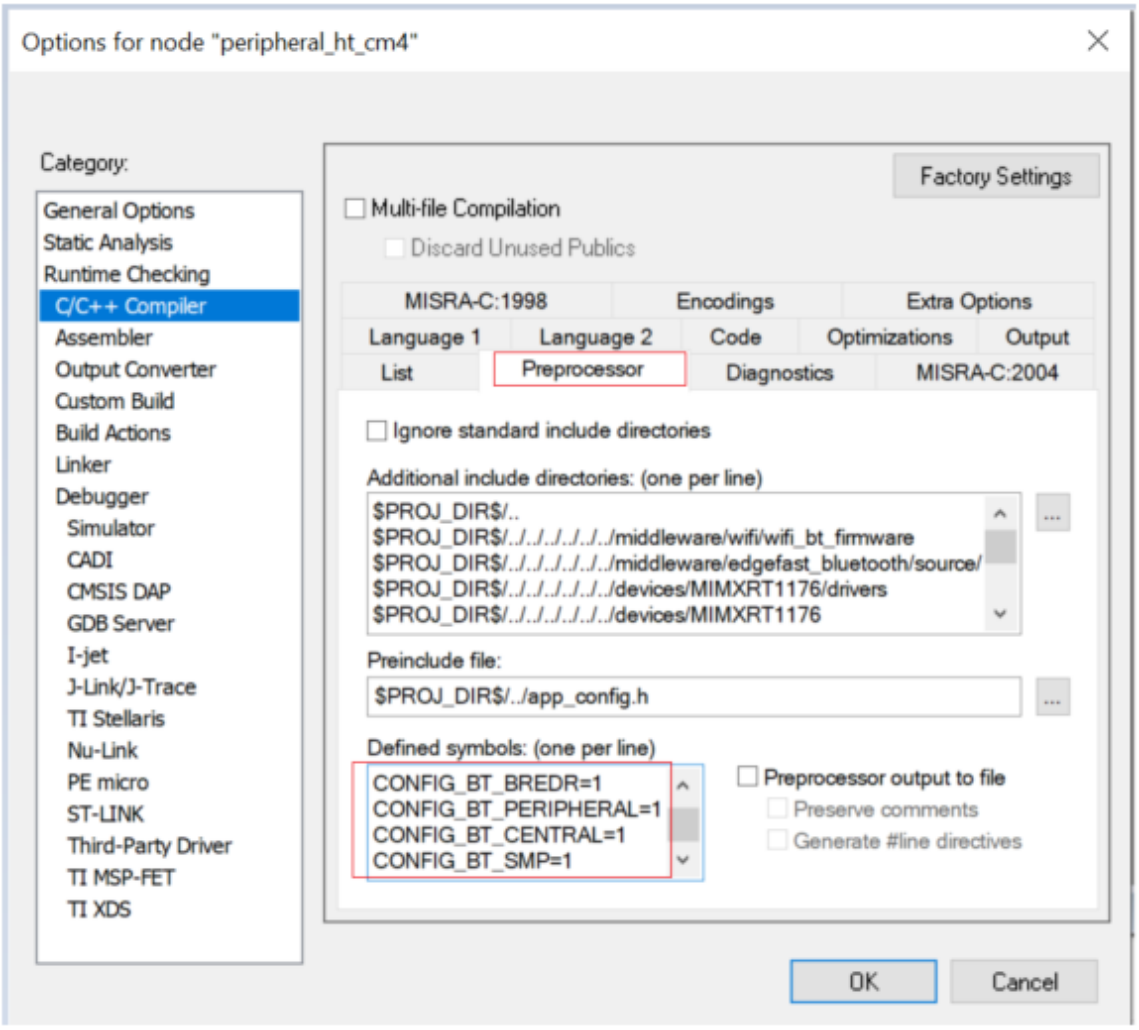
   The peripheral hts profile is in the *services* folder. Add the hts.c file to the *services* group of the *cm4* folder.



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*>

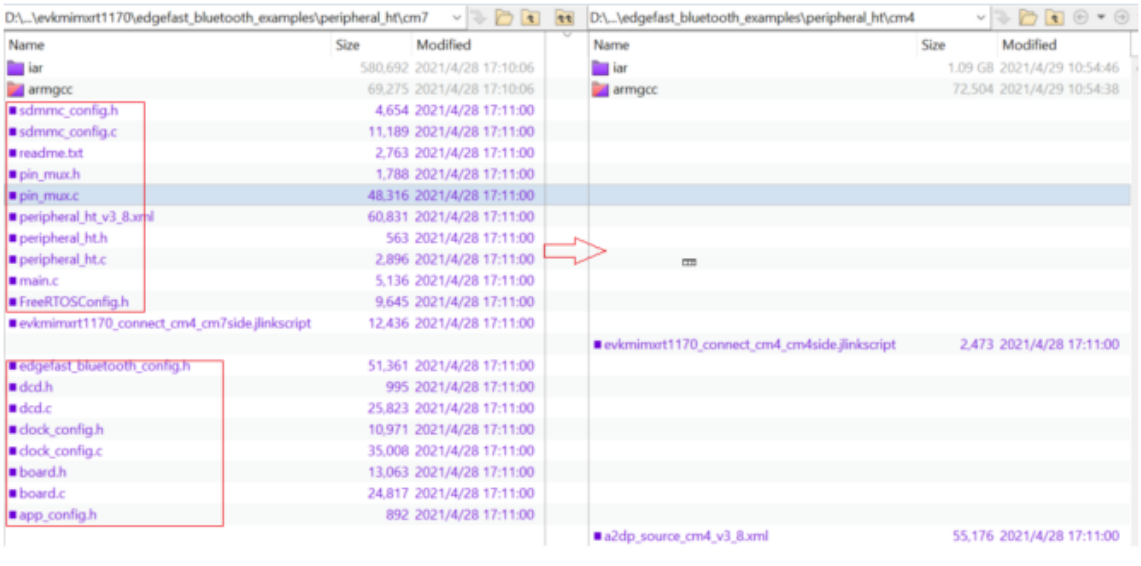
boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source \cm4 into*<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.

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:

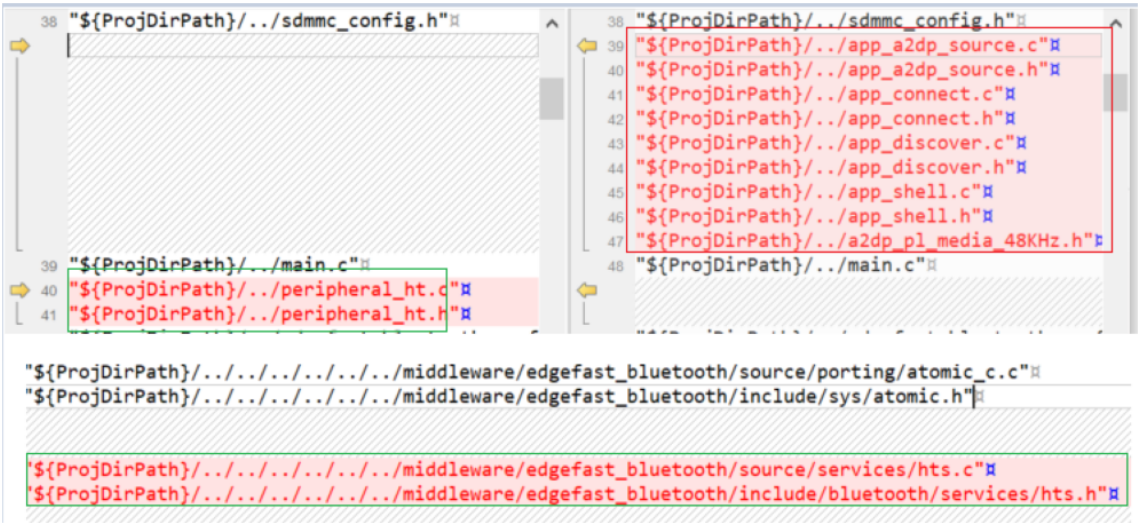
1. Open the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h in the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7* 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:

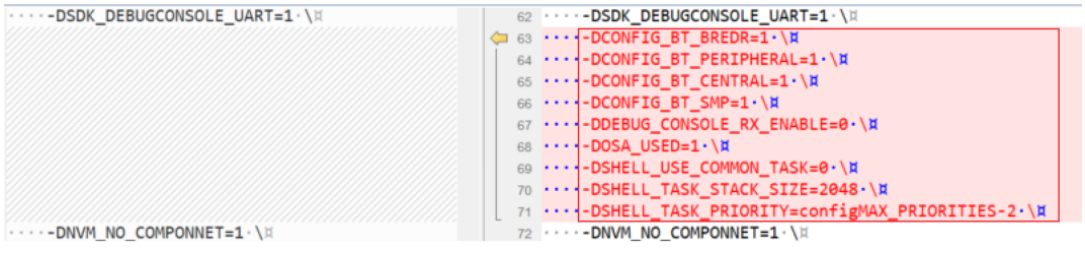
1. Open the *CMakeLists.txt* of the two examples respectively. The two files are in the *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7\armgcc* and *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\armgcc* folders respectively.
2. Search 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.

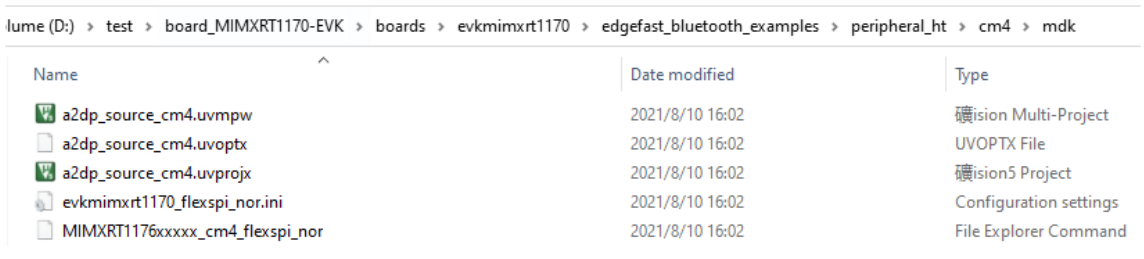
On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:*Arm GCC*

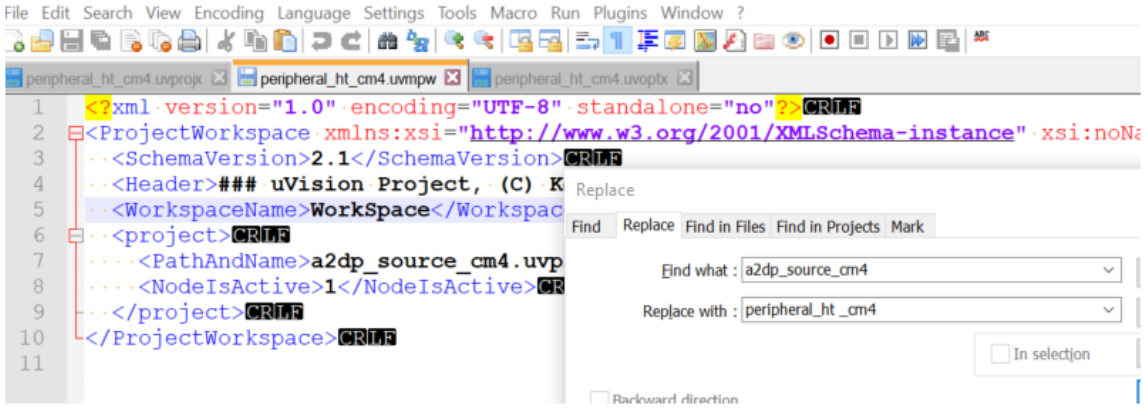
MDK This section describes the steps to create an M4 project with MDK, rearrange source and project files, adjust project settings, and delete function.

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*



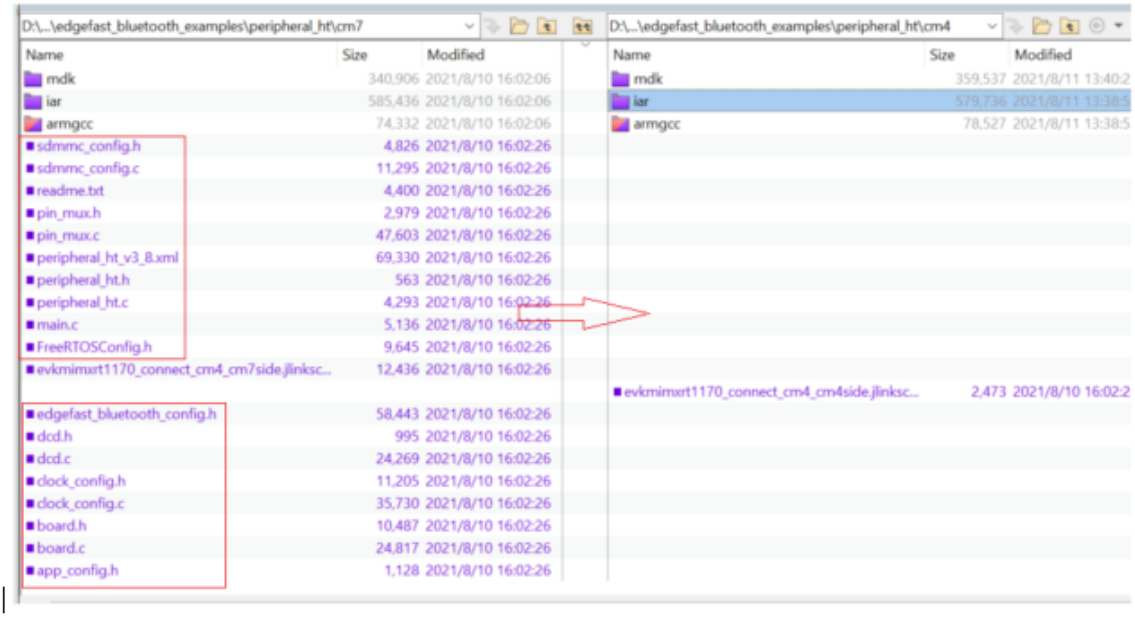
3. Change the filename *a2dp_source_cm4* to *peripheral_ht_cm4* respectively.
4. Open the files *peripheral_ht_cm4.**uvmpw* and *peripheral_ht_cm4.* *uvoptx*, *peripheral_ht_cm4.uvprojx*with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio code.
5. Search and replace *a2dp_source_cm4* 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*.

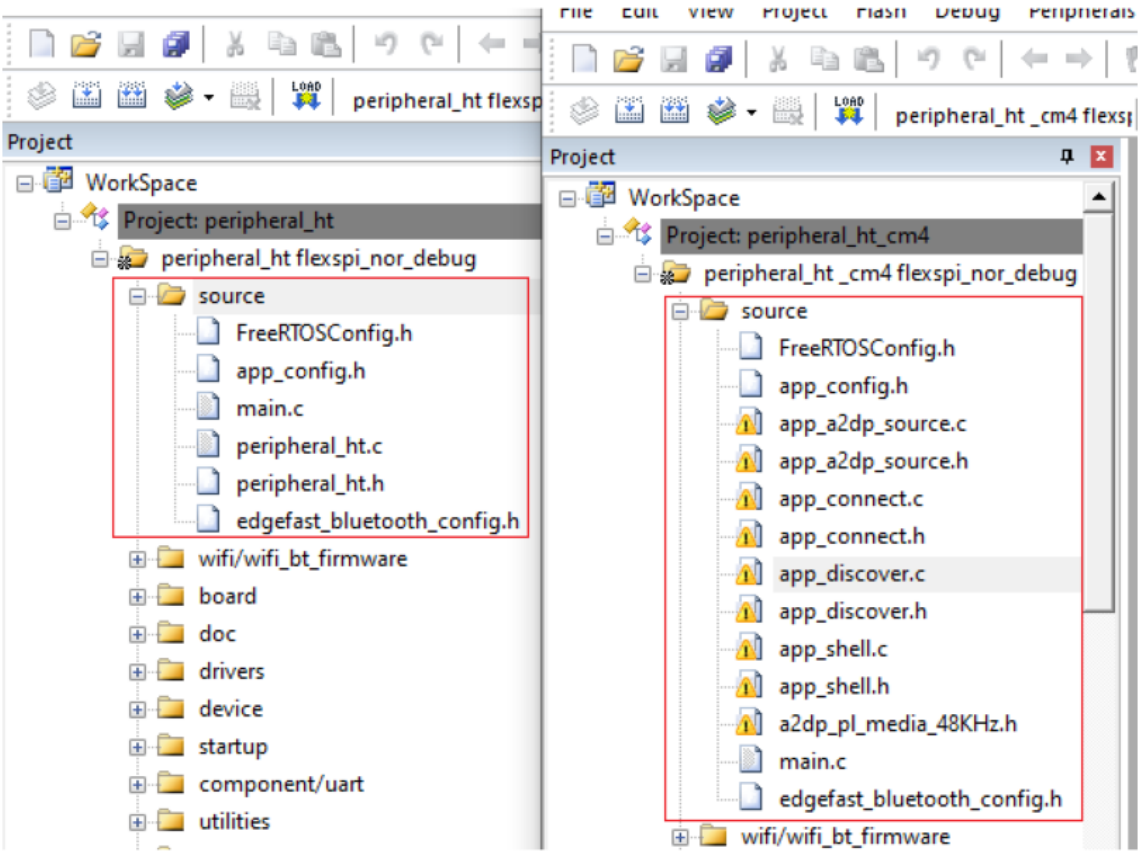


.

Parent topic:*MDK*

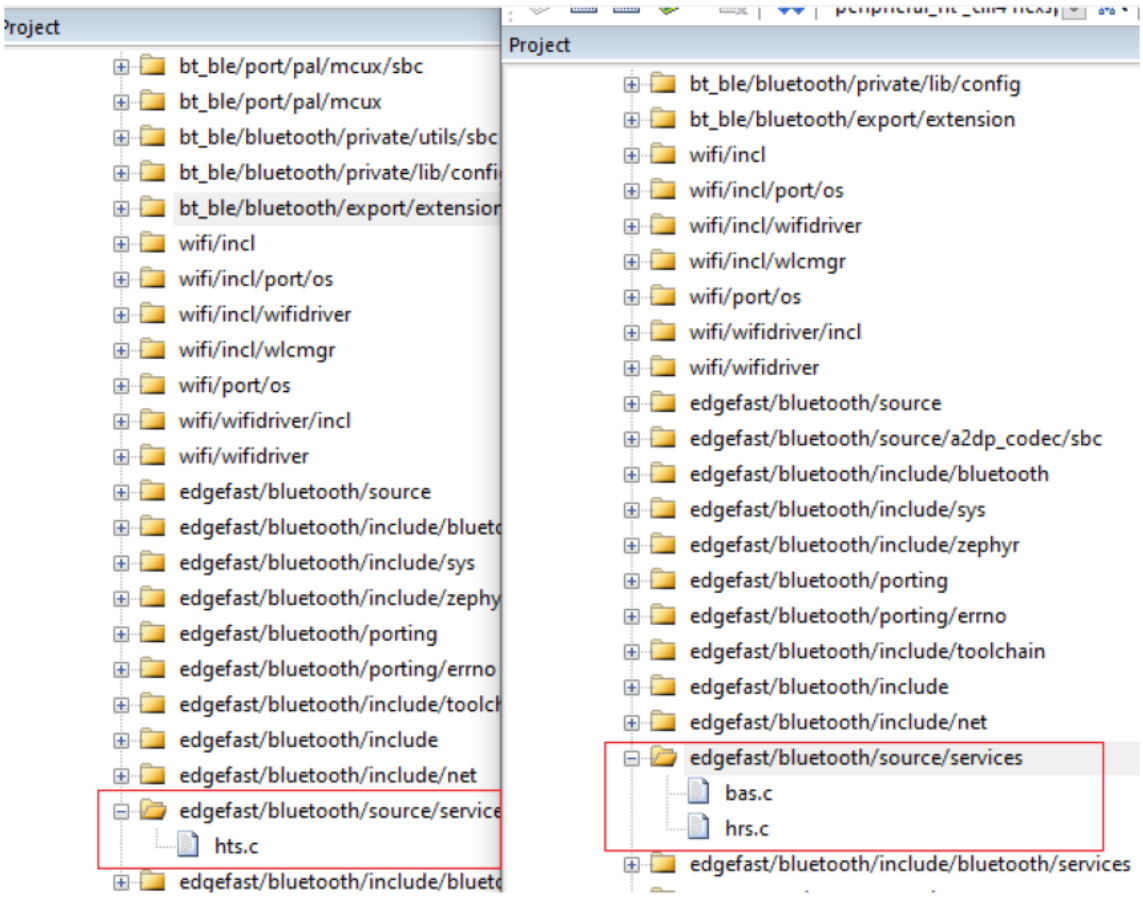
Rearrange project files

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.

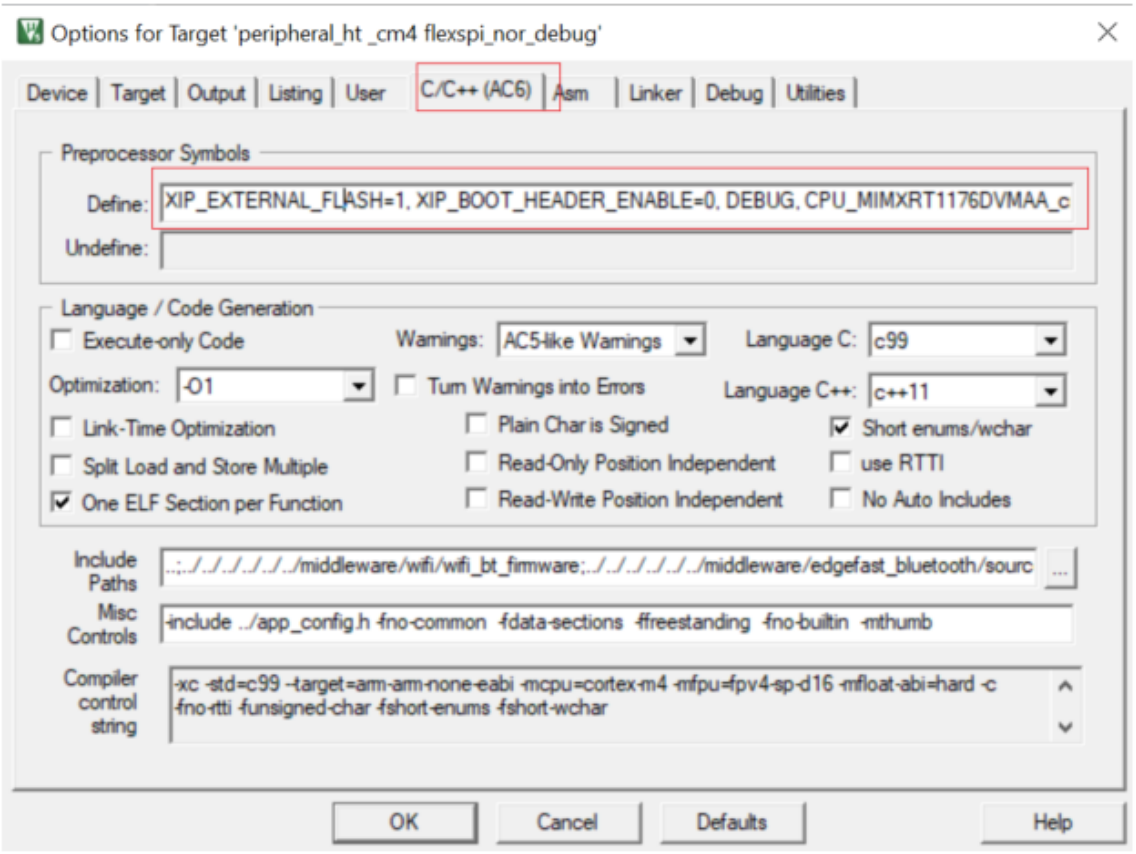


.

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