MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025

Table of contents

1 MCX-N5XX-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 . . . 45
  - 1.3.1 Getting Started with MCUXpresso SDK Repository . . . 45
- 1.4 Release Notes . . . 58
  - 1.4.1 MCUXpresso SDK Release Notes . . . 58
- 1.5 ChangeLog . . . 66
  - 1.5.1 MCUXpresso SDK Changelog . . . 66
- 1.6 Driver API Reference Manual . . . 191
- 1.7 Middleware Documentation . . . 192
  - 1.7.1 Multicore . . . 192
  - 1.7.2 MCU Boot . . . 192
  - 1.7.3 Audio Voice components . . . 192
  - 1.7.4 Maestro Audio Framework for MCU . . . 192
  - 1.7.5 eIQ . . . 192
  - 1.7.6 FreeMASTER . . . 192
  - 1.7.7 AWS IoT . . . 192
  - 1.7.8 NXP Wi-Fi . . . 192
  - 1.7.9 FreeRTOS . . . 192
  - 1.7.10 Wireless EdgeFast Bluetooth PAL . . . 192
  - 1.7.11 lwIP . . . 193
  - 1.7.12 File systemFatfs . . . 193

2 MCXN947 **195**
- 2.1 CACHE: CACHE Memory Controller . . . 195
- 2.2 CACHE: LPCAC CACHE Memory Controller . . . 199
- 2.3 CDOG . . . 199
- 2.4 Clock Driver . . . 203
- 2.5 CRC: Cyclic Redundancy Check Driver . . . 261
- 2.6 CTIMER: Standard counter/timers . . . 263
- 2.7 DAC: Digital-to-Analog Converter Driver . . . 273
- 2.8 DAC14: 14-bit Digital-to-Analog Converter Driver . . . 278
- 2.9 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 284
- 2.10 eDMA core Driver . . . 315
- 2.11 eDMA soc Driver . . . 322
- 2.12 Efuse_driver . . . 322
- 2.13 EIM: error injection module . . . 324
- 2.14 ERM: error recording module . . . 324
- 2.15 EVTG: Event Generator Driver . . . 326
- 2.16 EVTG Driver . . . 331
- 2.17 EWM: External Watchdog Monitor Driver . . . 332
- 2.18 FGPIO Driver . . . 335
- 2.19 Flash_driver . . . 335
- 2.20 Flash_ffr_driver . . . 345

2.21 FlexCAN: Flex Controller Area Network Driver . . . 358
2.22 FlexCAN Driver . . . 358
2.23 FlexCAN eDMA Driver . . . 405
2.24 FlexIO: FlexIO Driver . . . 408
2.25 FlexIO Driver . . . 408
2.26 FlexIO eDMA I2S Driver . . . 425
2.27 FlexIO eDMA SPI Driver . . . 429
2.28 FlexIO eDMA UART Driver . . . 432
2.29 FlexIO I2C Master Driver . . . 435
2.30 FlexIO I2S Driver . . . 444
2.31 FlexIO SPI Driver . . . 454
2.32 FlexIO UART Driver . . . 467
2.33 FLEXSPI: Flexible Serial Peripheral Interface Driver . . . 478
2.34 FLEXSPI eDMA Driver . . . 495
2.35 Flexspi_nor_flash_driver . . . 498
2.36 FREQME: Frequency Measurement . . . 516
2.37 GDET . . . 516
2.38 Glitch Detect . . . 518
2.39 GPIO: General-Purpose Input/Output Driver . . . 518
2.40 GPIO Driver . . . 521
2.41 I3C: I3C Driver . . . 527
2.42 I3C Common Driver . . . 529
2.43 I3C Master Driver . . . 531
2.44 I3C Master DMA Driver . . . 557
2.45 I3C Slave Driver . . . 560
2.46 I3C Slave DMA Driver . . . 573
2.47 INPUTMUX: Input Multiplexing Driver . . . 576
2.48 INTM: Interrupt Monitor Driver . . . 714
2.49 Inline Prince Encryption Decryption . . . 717
2.50 IRTC: IRTC Driver . . . 722
2.51 Intrusion and Tamper Response Controller . . . 731
2.52 ITRC . . . 731
2.53 Common Driver . . . 734
2.54 LPADC: 12-bit SAR Analog-to-Digital Converter Driver . . . 746
2.55 Lpc_freqme . . . 767
2.56 LPCMP: Low Power Analog Comparator Driver . . . 772
2.57 LPFLEXCOMM: LPFLEXCOMM Driver . . . 781
2.58 LPFLEXCOMM Driver . . . 781
2.59 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 783
2.60 LPI2C Master Driver . . . 783
2.61 LPI2C Master DMA Driver . . . 797
2.62 LPI2C Slave Driver . . . 800
2.63 LPSPI: Low Power Serial Peripheral Interface . . . 810
2.64 LPSPI Peripheral driver . . . 810
2.65 LPSPI eDMA Driver . . . 831
2.66 LPTMR: Low-Power Timer . . . 837
2.67 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 842
2.68 LPUART Driver . . . 842
2.69 LPUART eDMA Driver . . . 863
2.70 Mailbox . . . 866
2.71 MCX_CMC: Core Mode Controller Driver . . . 866
2.72 ENET: Ethernet Driver . . . 878
2.73 Mcx_enet . . . 878
2.74 MCX_SPC: System Power Control driver . . . 907
2.75 MCX_VBAT: Smart Power Switch . . . 947
2.76 Memory_interface . . . 967
2.77 MRT: Multi-Rate Timer . . . 975
2.78 Nboot . . . 979

2.79 Nbot_hal . . . 986
2.80 OPAMP: Operational Amplifier . . . 991
2.81 OSTIMER: OS Event Timer Driver . . . 996
2.82 OTP: One-Time Programmable memory and API . . . 999
2.83 PDM: Microphone Interface . . . 1002
2.84 PDM Driver . . . 1002
2.85 PDM EDMA Driver . . . 1014
2.86 PINT: Pin Interrupt and Pattern Match Driver . . . 1018
2.87 PLU: Programmable Logic Unit . . . 1026
2.88 PORT: Port Control and Interrupts . . . 1035
2.89 POWERQUAD: PowerQuad hardware accelerator . . . 1044
2.90 PRINCE: PRINCE bus crypto engine . . . 1074
2.91 PUF: Physical Unclonable Function . . . 1081
2.92 Puf_v3_driver . . . 1083
2.93 PWM: Pulse Width Modulator . . . 1089
2.94 QDC: Quadrature Encoder/Decoder . . . 1114
2.95 Reset Driver . . . 1124
2.96 RNG: Random Number Generator . . . 1130
2.97 RTC: Real Time Clock . . . 1131
2.98 Runbootloader . . . 1136
2.99 SAI: Serial Audio Interface . . . 1136
2.100 SAI Driver . . . 1136
2.101 SAI EDMA Driver . . . 1163
2.102 Sbloader . . . 1169
2.103 SCTimer: SCTimer/PWM (SCT) . . . 1180
2.104 SEMA42: Hardware Semaphores Driver . . . 1197
2.105 SINC: SINC Filter . . . 1200
2.106 Smart Card . . . 1231
2.107 Smart Card EMVSIM Driver . . . 1239
2.108 Smart Card PHY Driver . . . 1242
2.109 Smart Card PHY EMVSIM Driver . . . 1243
2.110 Smart Card PHY TDA8035 Driver . . . 1243
2.111 Smart Card PHY USIM W . . . 1243
2.112 Smart Card USIM Driver . . . 1243
2.113 SMARTDMA: SMART DMA Driver . . . 1246
2.114 SYSPM: System Performance Monitor . . . 1248
2.115 Digital Tamper . . . 1251
2.116 TDET . . . 1251
2.117 TRDC: Trusted Resource Domain Controller . . . 1265
2.118 Trdc_core . . . 1287
2.119 Trdc_soc . . . 1301
2.120 Tsi_v6_driver . . . 1301
2.121 USDHC: Ultra Secured Digital Host Controller Driver . . . 1328
2.122 UTICK: MictoTick Timer Driver . . . 1358
2.123 VREF: Voltage Reference Driver . . . 1360
2.124 WUU: Wakeup Unit driver . . . 1362
2.125 WWDT: Windowed Watchdog Timer Driver . . . 1367

3 Middleware **1371**
3.1 Boot . . . 1371
3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource . . . 1371
3.1.2 MCUboot . . . 1372
3.2 eIQ . . . 1373
3.2.1 eIQ . . . 1373
3.3 Motor Control . . . 1401
3.3.1 FreeMASTER . . . 1402
3.4 Multimedia . . . 1439
3.4.1 Audio Voice . . . 1439

3.5 Wireless . . . 1492
3.5.1 NXP Wireless Framework and Stacks . . . 1492
3.5.2 EdgeFast Bluetooth . . . 1492

4 RTOS **1567**
4.1 FreeRTOS . . . 1567
4.1.1 FreeRTOS kernel . . . 1567
4.1.2 FreeRTOS drivers . . . 1567
4.1.3 backoffalgorithm . . . 1567
4.1.4 corehttp . . . 1567
4.1.5 corejson . . . 1567
4.1.6 coremqtt . . . 1568
4.1.7 corepkcs11 . . . 1568
4.1.8 freertos-plus-tcp . . . 1568

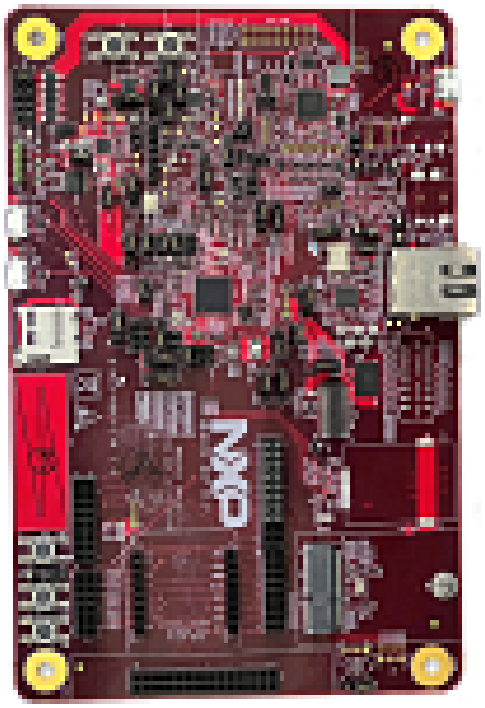
This documentation contains information specific to the mcxn5xxevk board.

Chapter 1

MCX-N5XX-EVK

1.1 Overview

The NXP N5xx-EVK is a development board for the N5xx 150 MHz Arm Dual Cortex-M33 TrustZone microcontroller, which is for Industrial and Consumer IoT Applications.



MCU device and part on board is shown below:

- Device: MCXN547
- PartNumber: MCXN547VDF

1.2 Getting Started with MCUXpresso SDK Package

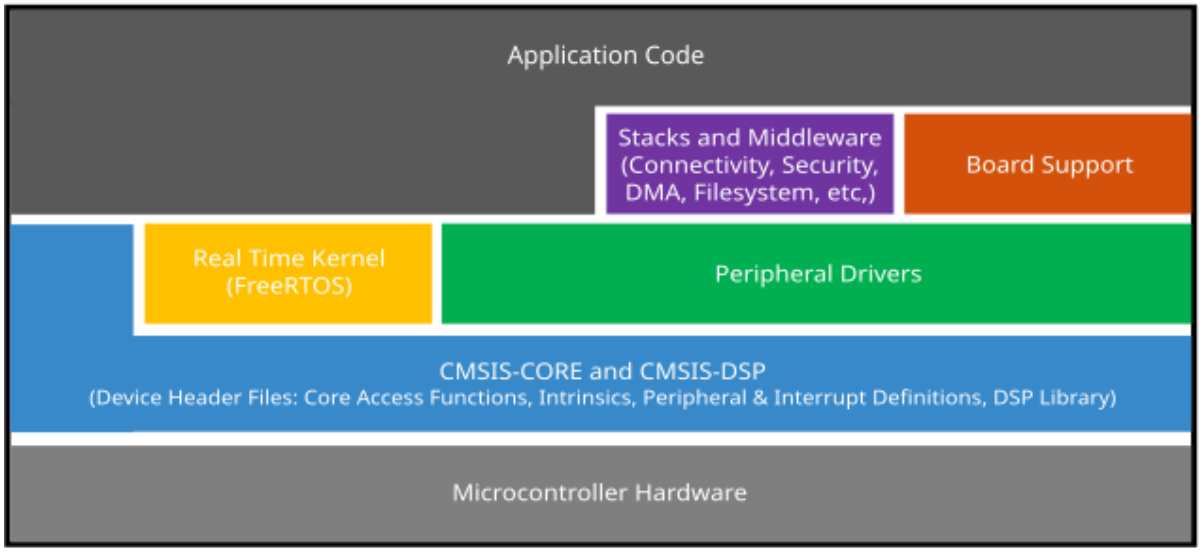
1.2.1 Getting Started with Package

Overview

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

For supported toolchain versions, see *MCUXpresso SDK Release Notes for MCX-N9XX-EVK* (document MCUXSDKMCXN9XXRN).

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



MCUXpresso SDK board support package folders

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

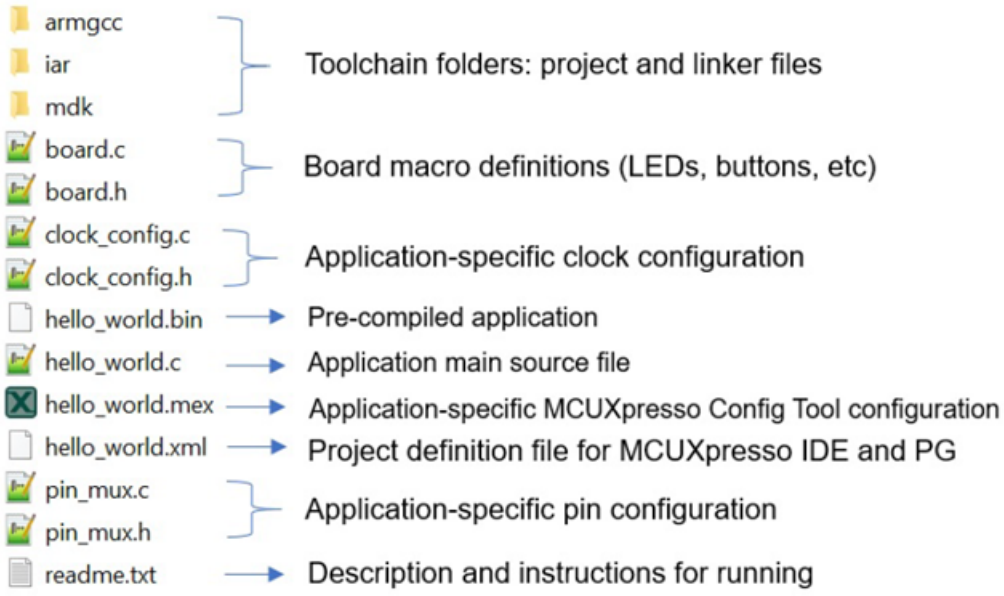
- cmsis_driver_examples: Simple applications intended to show how to use CMSIS drivers.
- demo_apps: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: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

- usb_examples: Applications that use the USB host/device/OTG stack.

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

Each <board_name> folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the hello_world example (part of the demo_apps folder), the same general rules apply to any type of example in the <board_name> folder.

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



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

Parent topic:*MCUXpresso SDK board support package folders*

Locating example application source files When opening an example application in any of the supported IDEs, a variety of 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 device's CMSIS header file, MCUXpresso SDK feature file and a few other files
- devices/<device_name>/cmsis_drivers: All the CMSIS drivers for your specific MCU
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU
- devices/<device_name>/<tool_name>: Toolchain-specific startup code, including vector table definitions
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications

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

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

Parent topic:*MCUXpresso SDK board support package folders*

Run a demo application using MCUXpresso IDE

Note: Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

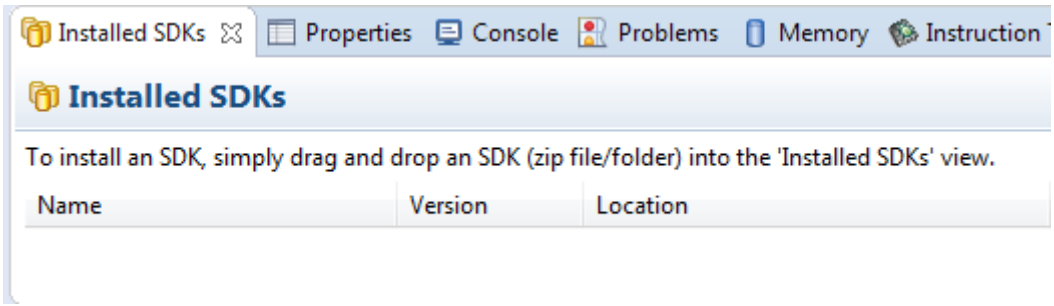
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello_world demo application targeted for the MCX-N9XX-EVK hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be located outside of the MCUXpresso SDK tree.

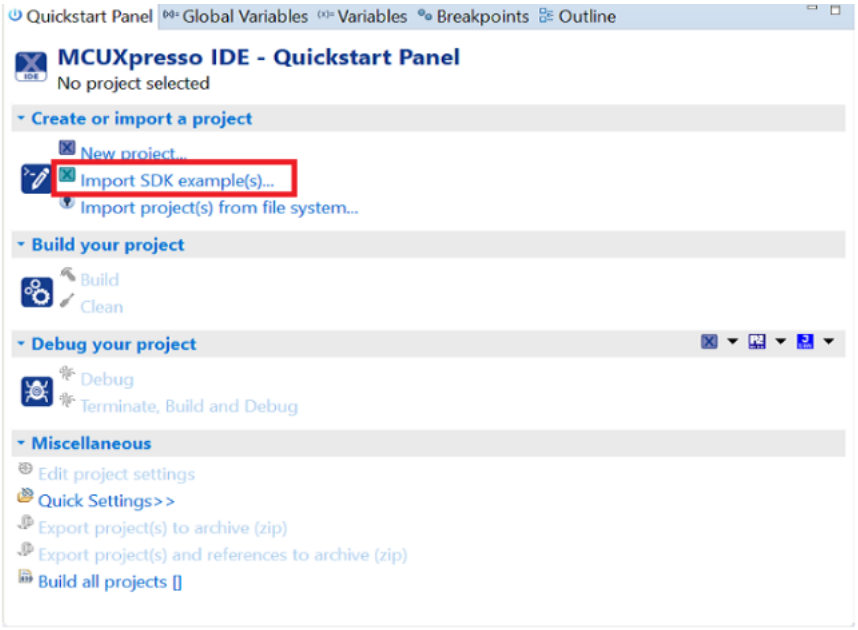
Parent topic:*Run a demo application using MCUXpresso IDE*

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

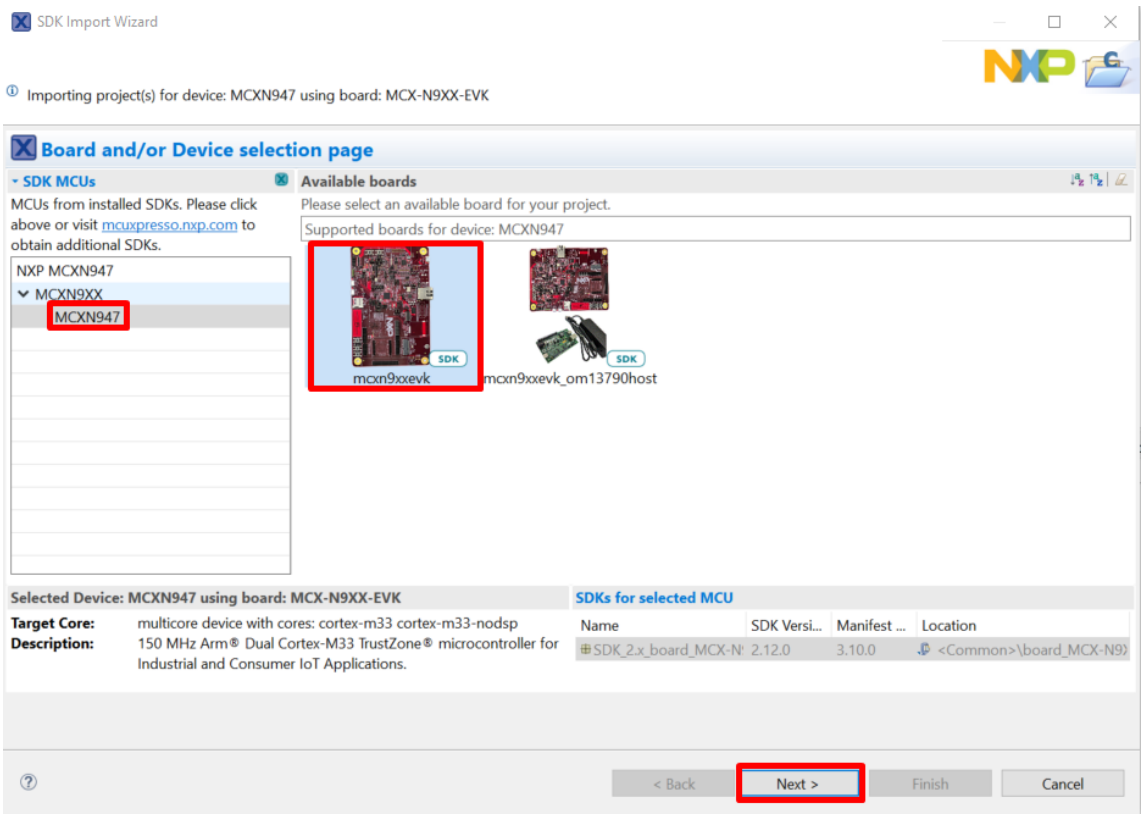
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



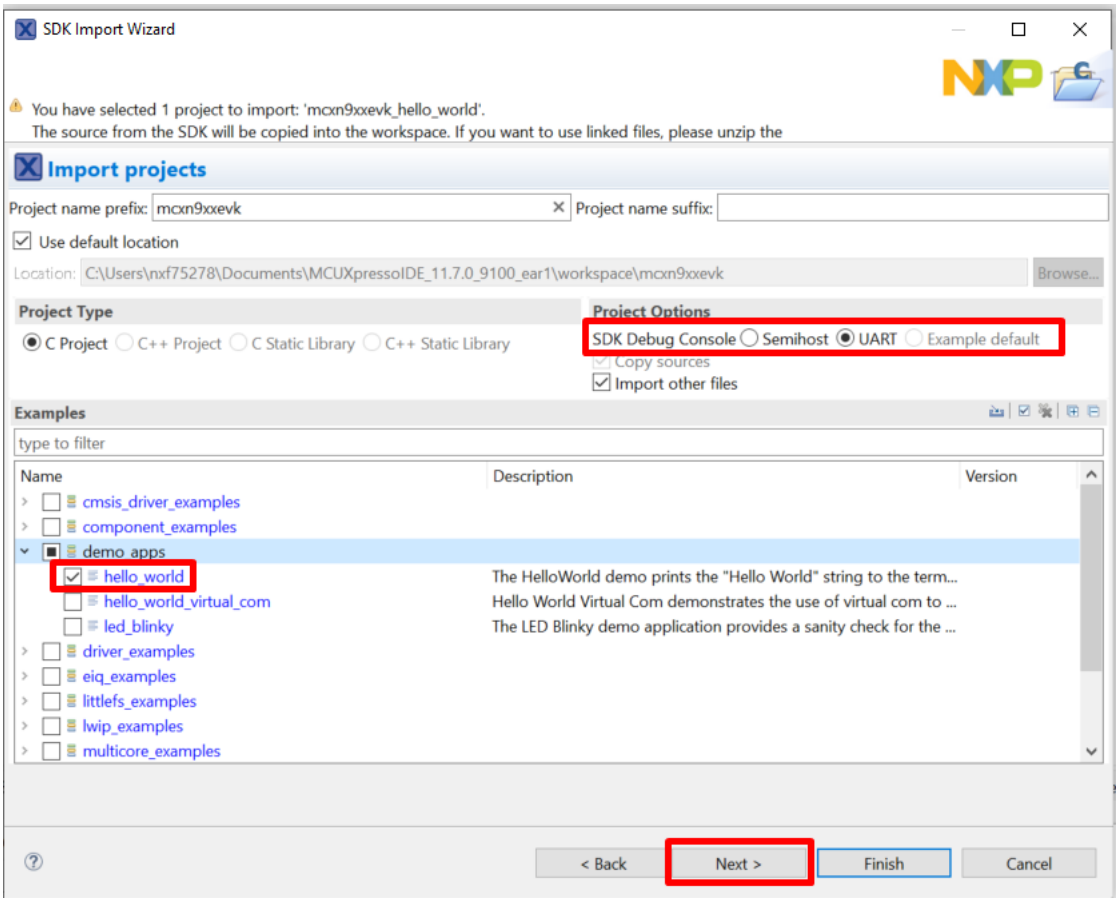
2. On the **Quickstart Panel**, click **Import SDK example(s)...**.



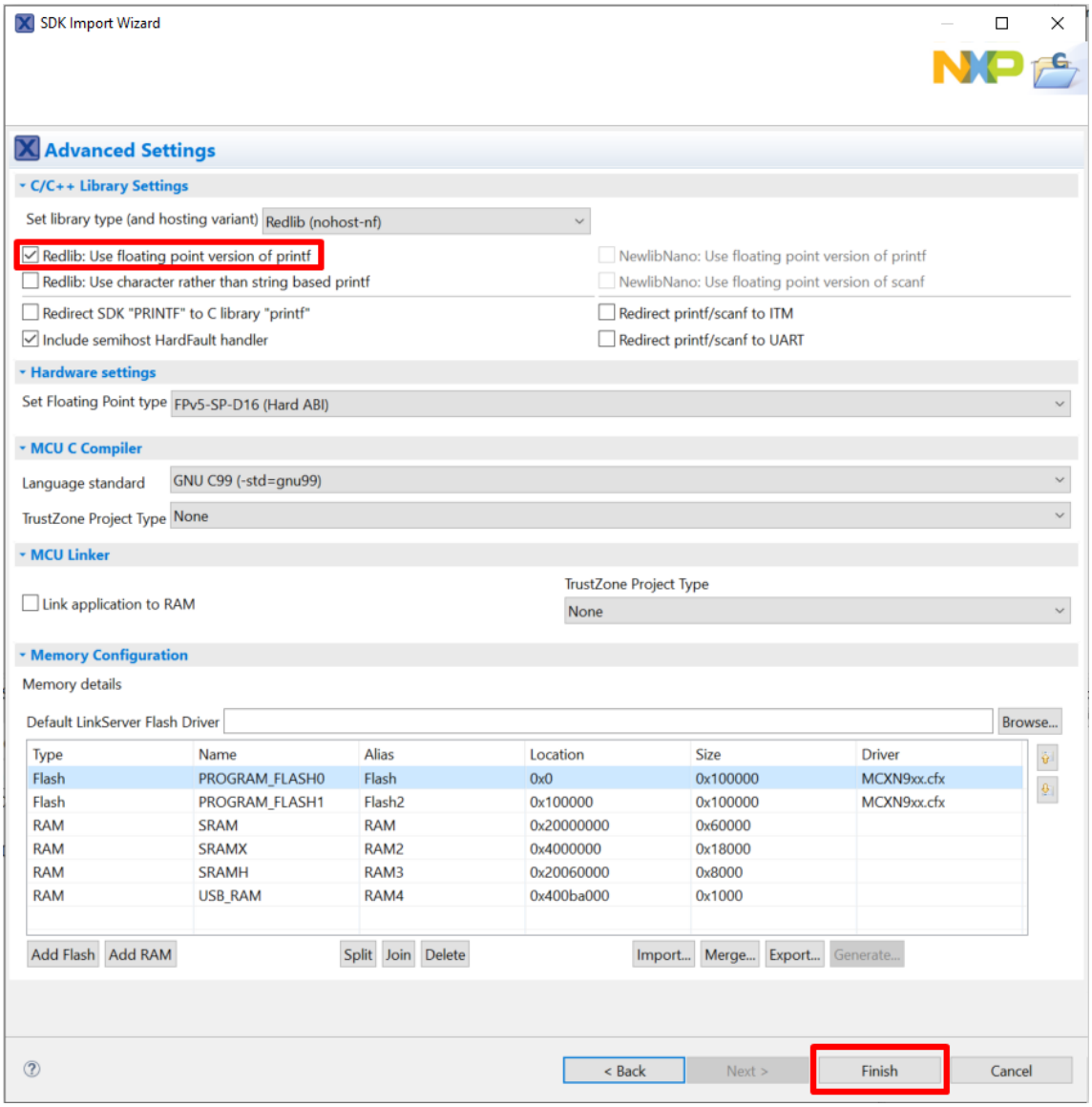
3. In the window that appears, expand the **MCXN9XX** folder and select **MCXN947** . Then, select **mcxn9xxevk** and click **Next**.



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



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



Parent topic:*Run a demo application using MCUXpresso IDE*

Run an example application For more information on debug probe support in the MCUXpresso IDE, see community.nxp.com.

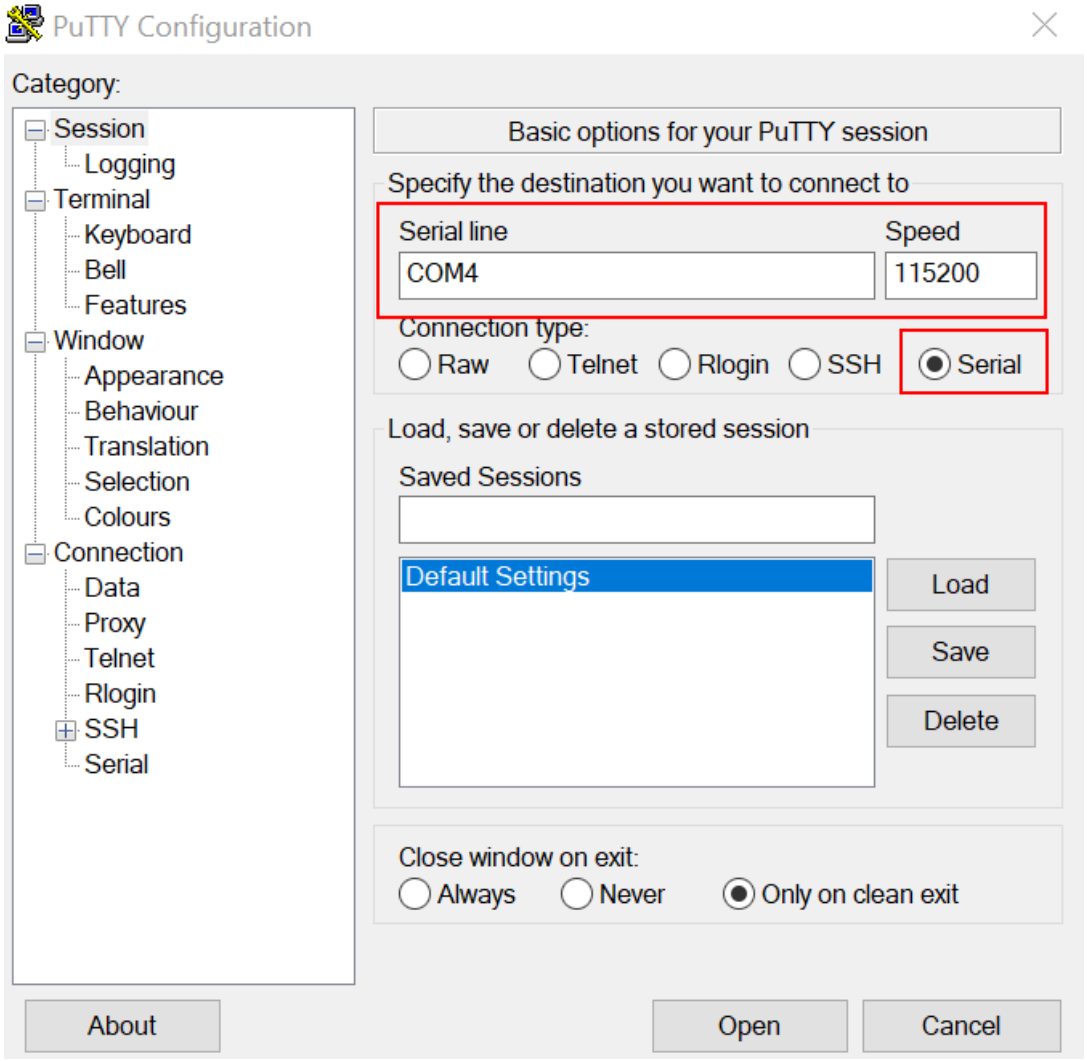
To download and run the application, perform the following steps:

1. See *Table 1* to determine the debug interface that comes loaded on your specific hardware platform. For LPCXpresso boards, install the DFU jumper for the debug probe, then connect the debug probe USB connector.
   - For boards with a P&E Micro interface, see PE micro to download and install the P&E Micro Hardware Interface Drivers package.
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see *How to determine com port*.

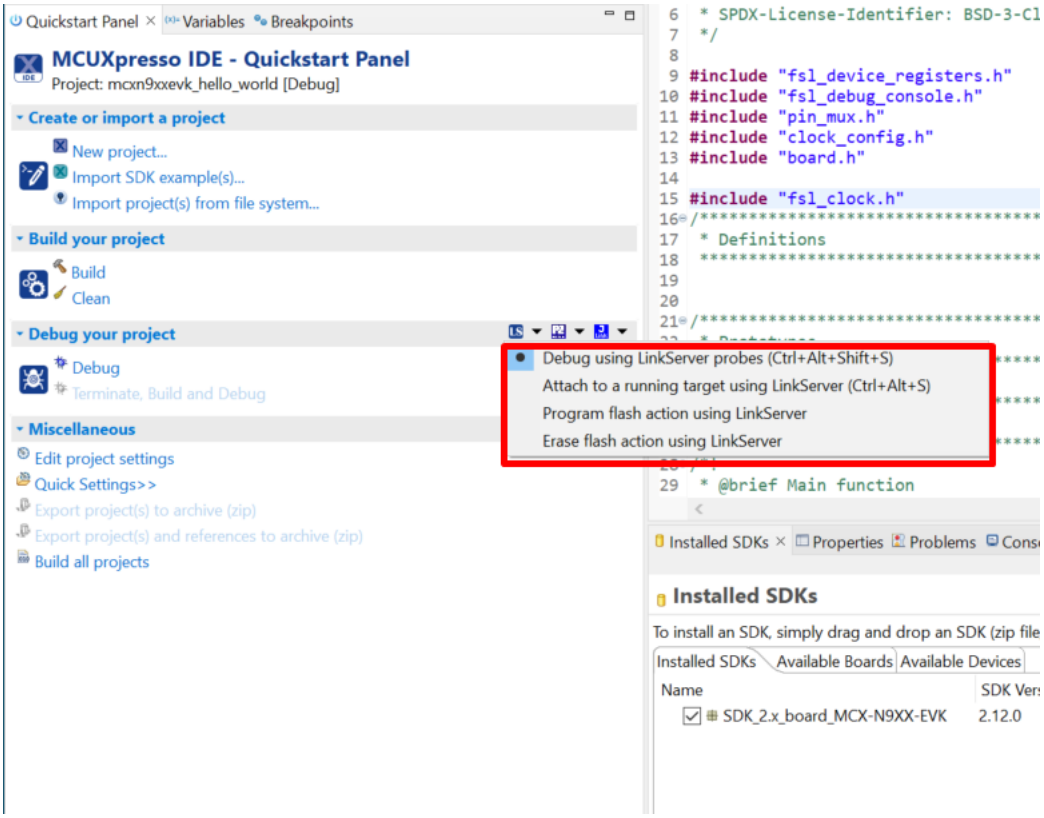
   Configure the terminal with these settings:

   1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)

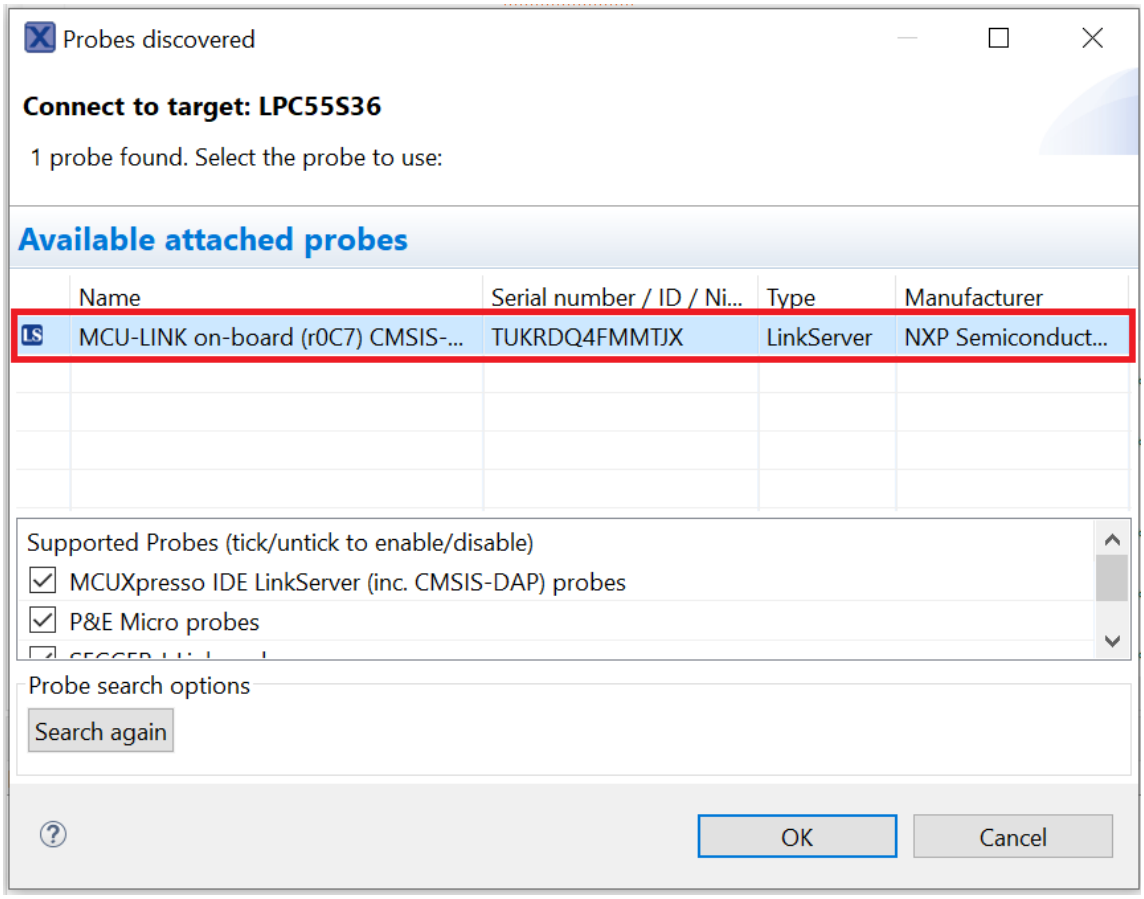
2. No parity
3. 8 data bits
4. 1 stop bit



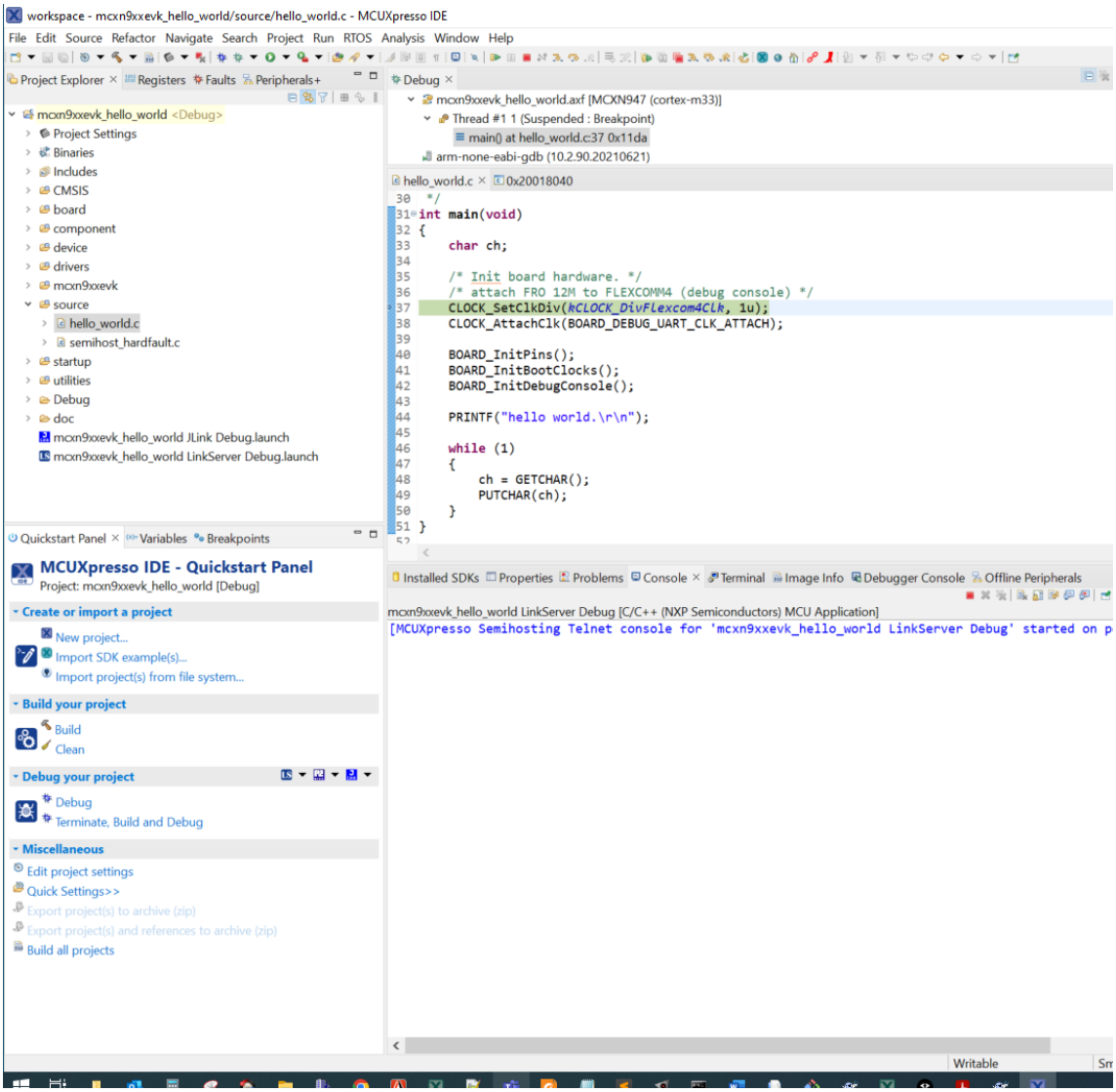
4. On the **Quickstart Panel**, click on **Debug mcxn9xxevk_hello_world [Debug]** to launch the debug session.



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



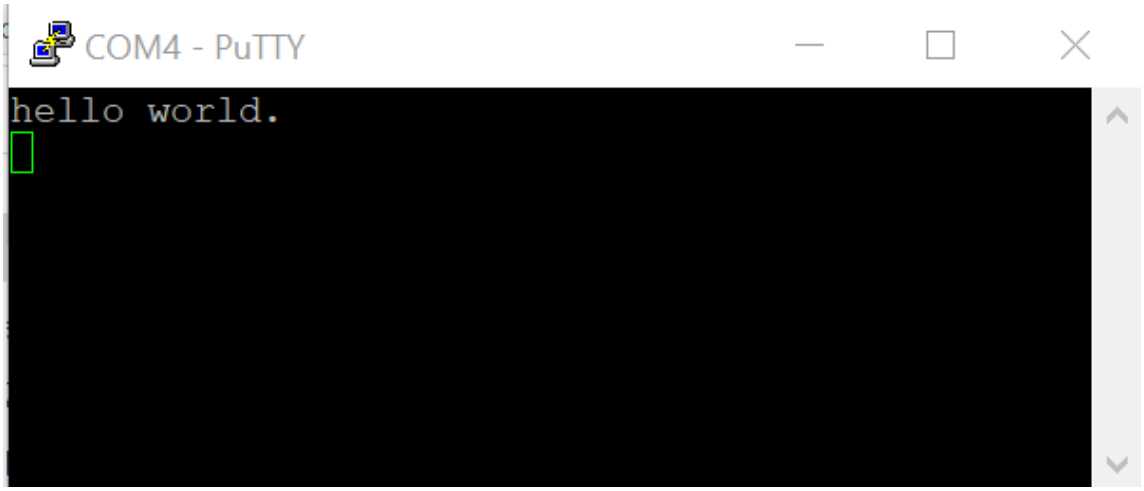
6. The application is downloaded to the target and automatically runs to main().



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



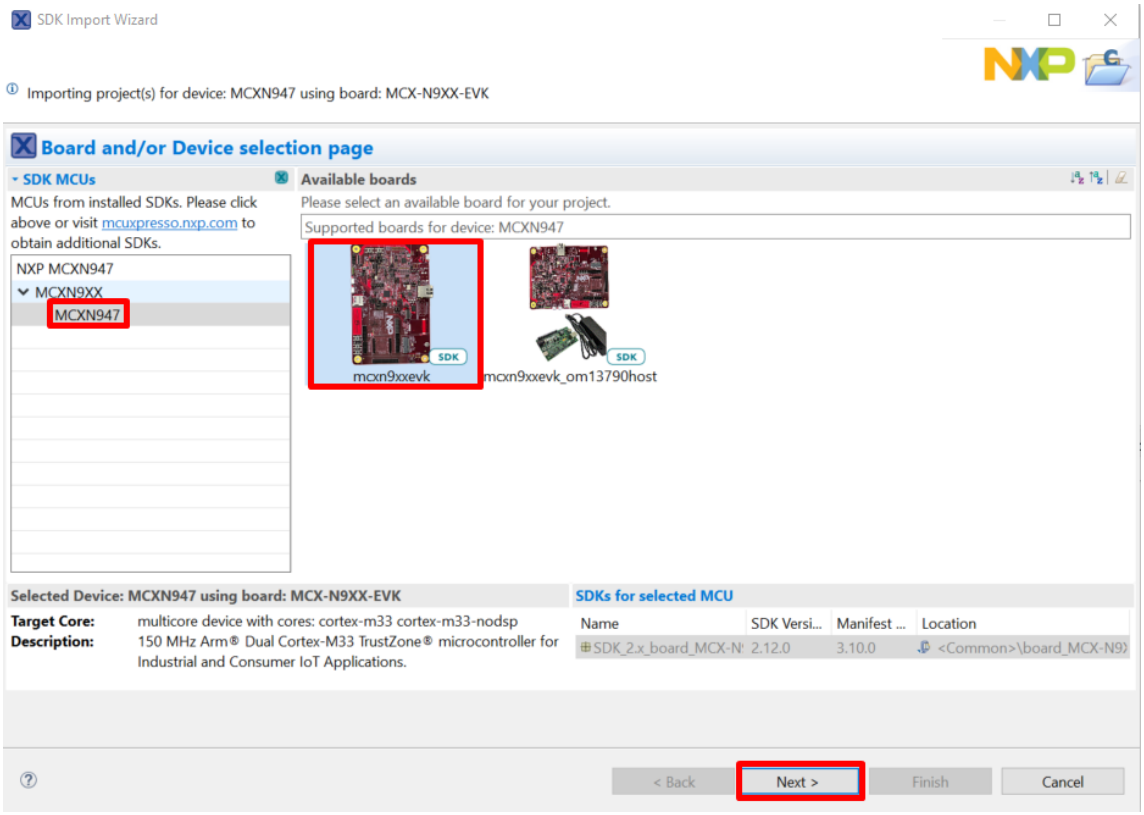
8. The hello_world application is now running and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



Parent topic:*Run a demo application using MCUXpresso IDE*

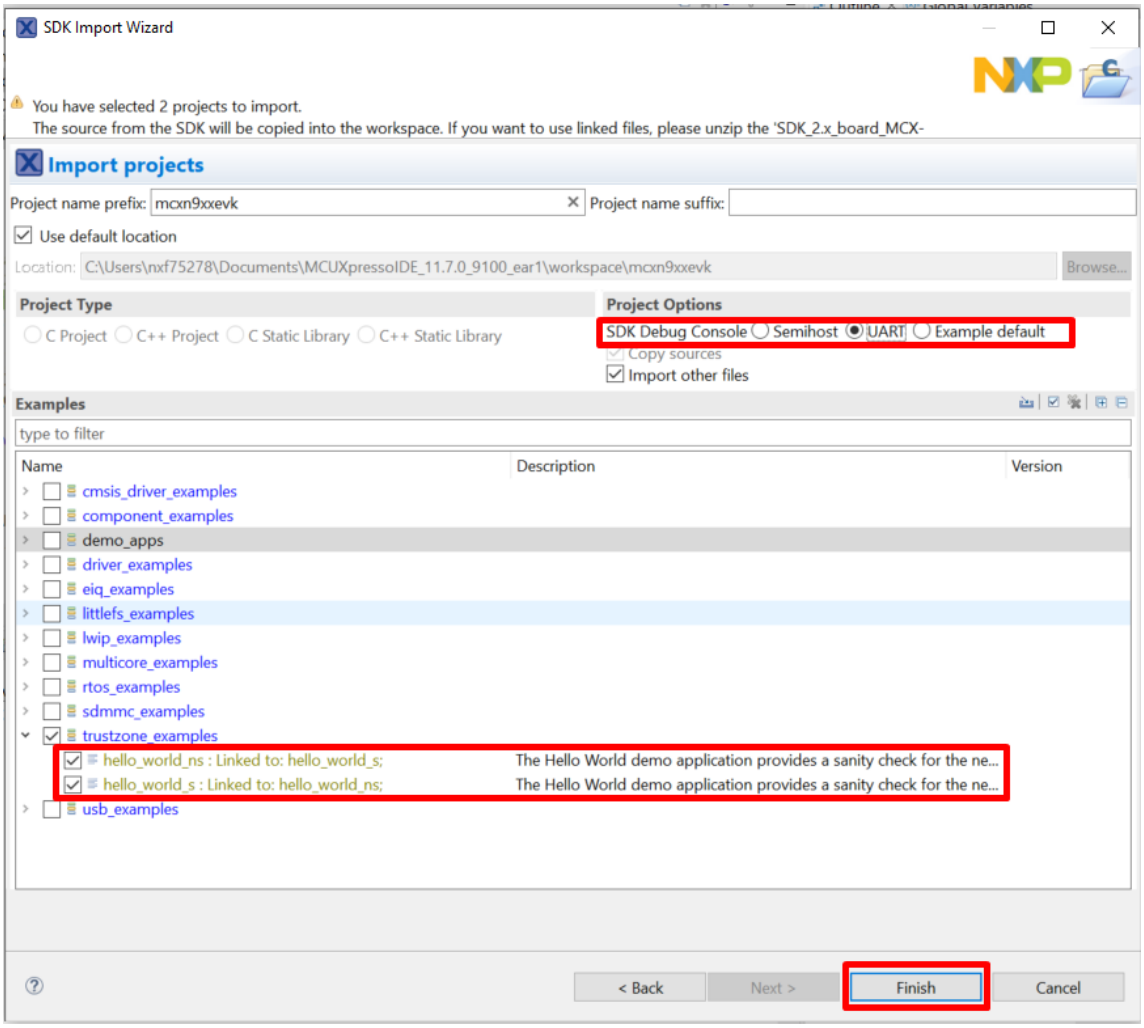
Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The trustzone version of the hello_world example application targeted for the MCX-N9XX-EVK hardware platform is used as an example, though these steps can be applied to any TrustZone example application in the MCUXpresso SDK.

1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for MCX-N9XX-EVK is installed and available in the **Installed SDKs** view, click Import SDK example(s)... on the **Quickstart** Panel. In the window that appears, expand the **MCXN9XX** folder and select **MCXN947**. Then, select **mcxn9xxevk** and click **Next**.

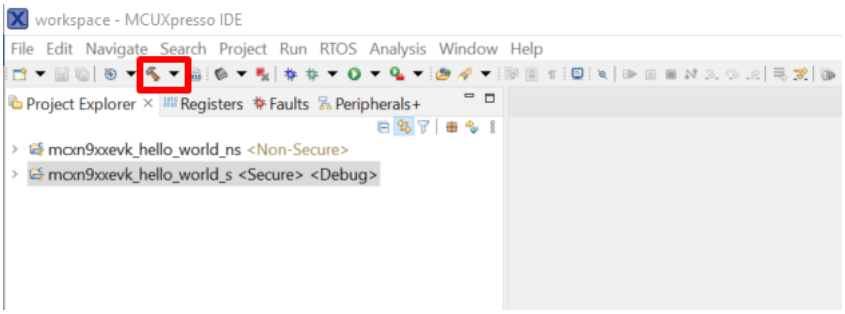


2. Expand the **trustzone_examples/** folder and select **hello_world_s**. Because TrustZone examples are linked together, the non-secure project is automatically imported with the se-

cure project, and there is no need to select it explicitly. Then select **UART** as **SDK Debug Console**. Then, click **Finish**.



3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the **mcxn9xxevk_hello_world_s** project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, Debug or Release, by clicking the downward facing arrow next to the hammer icon, as shown in *Figure 3*. For this example, select the Debug target.

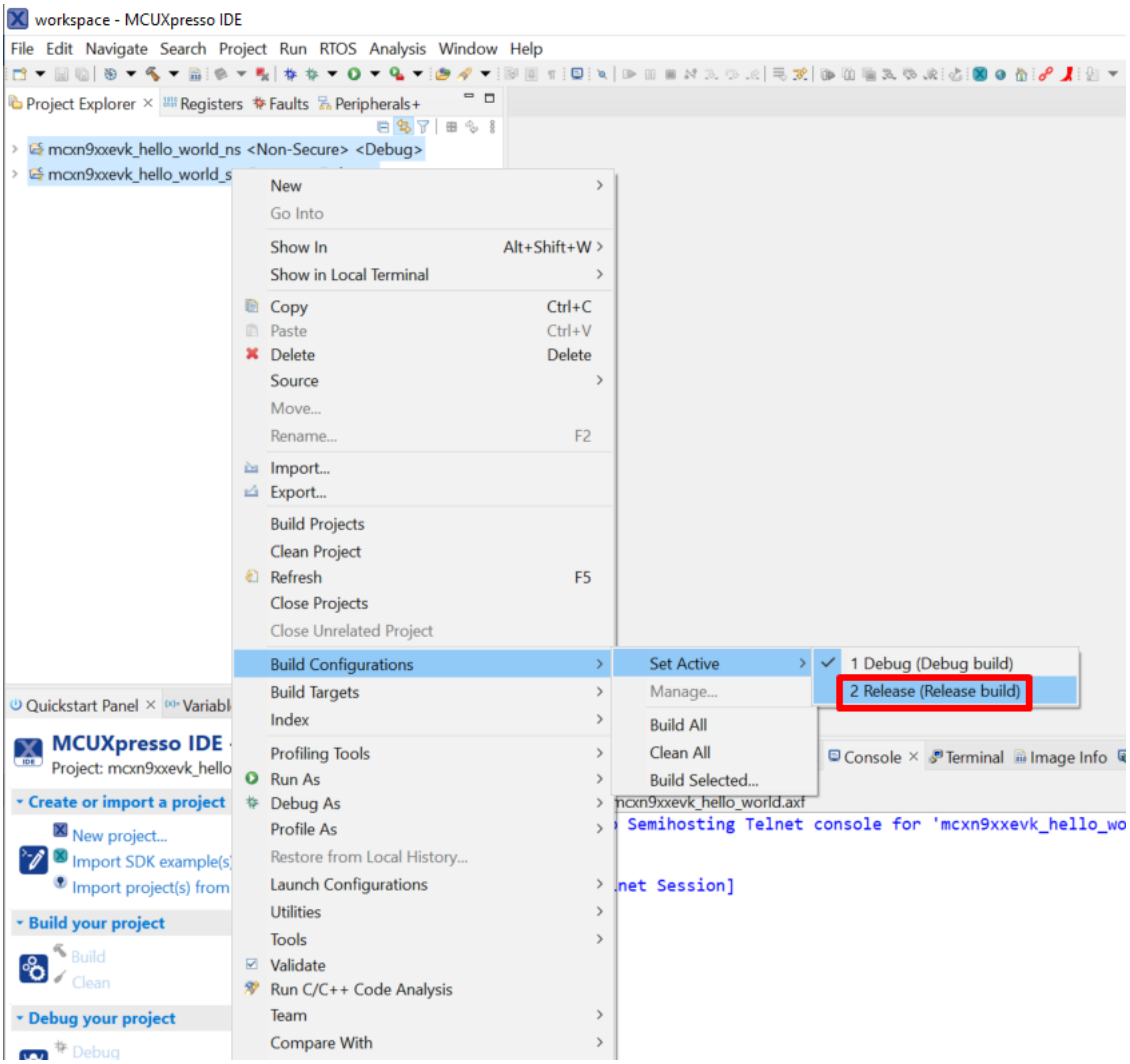


The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker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

Note:

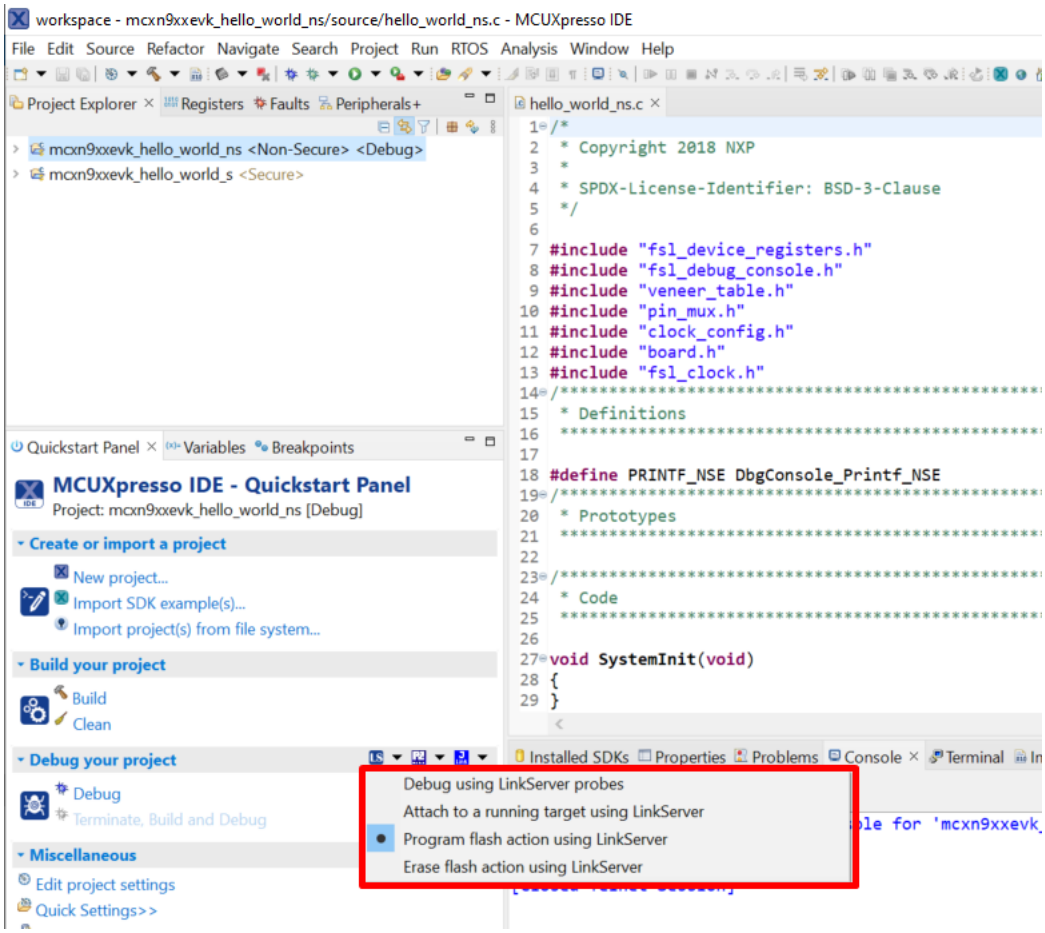
When the Release build is requested, it is necessary to change the build configuration of

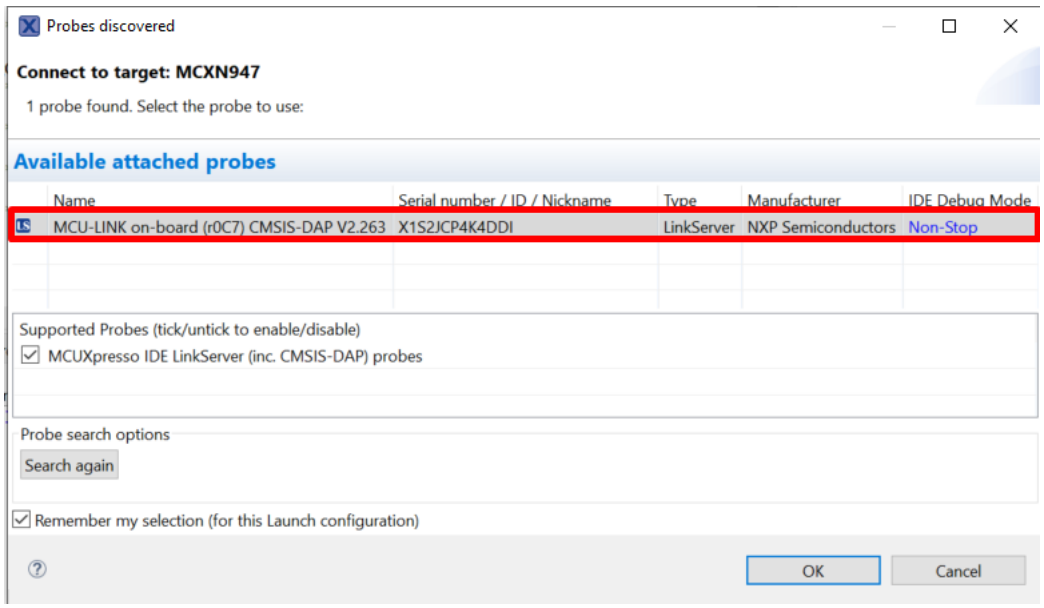
both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click to select the second project. Right click in the Project Explorer view to display the context-sensitive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



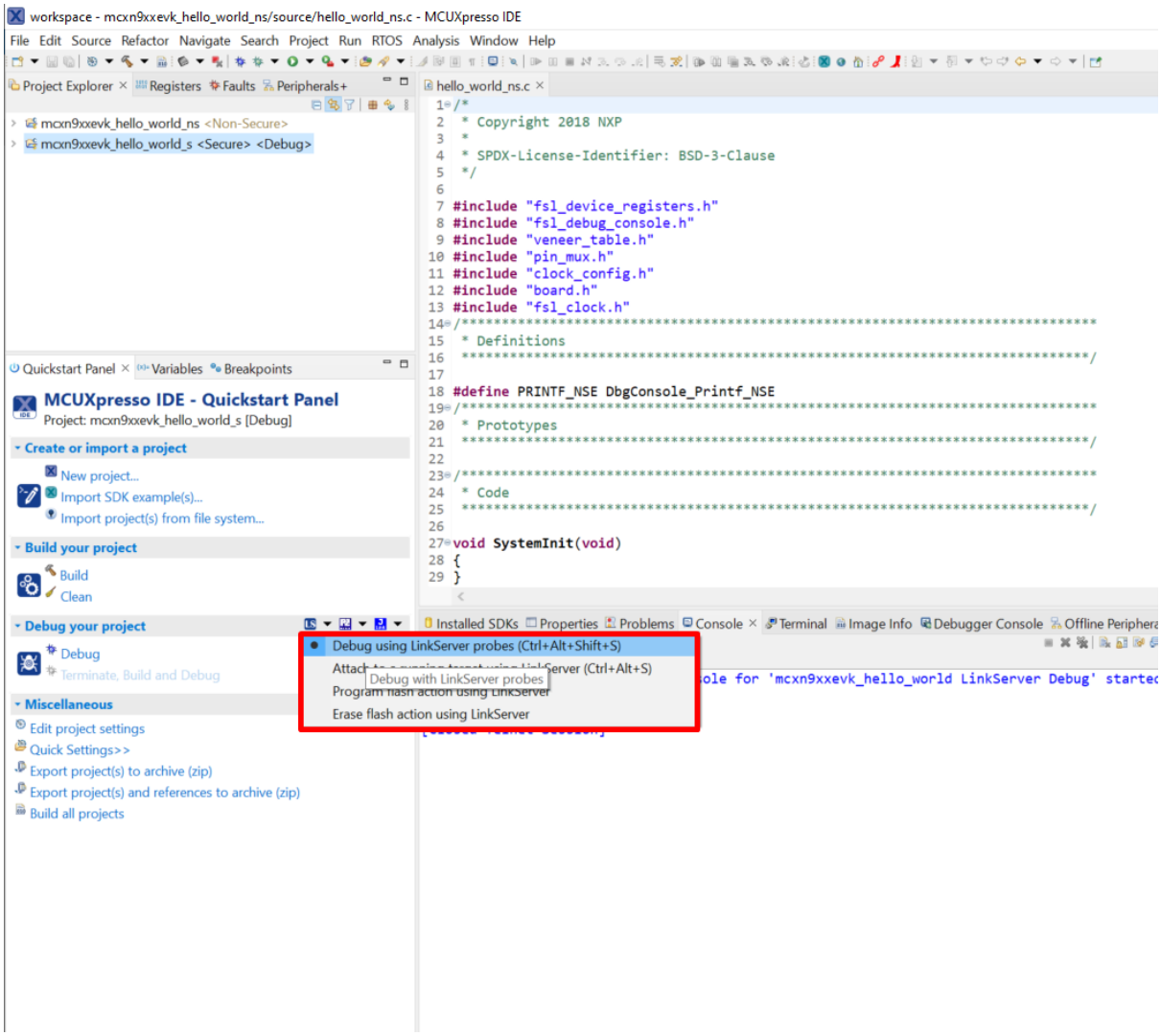
Parent topic:*Run a demo application using MCUXpresso IDE*

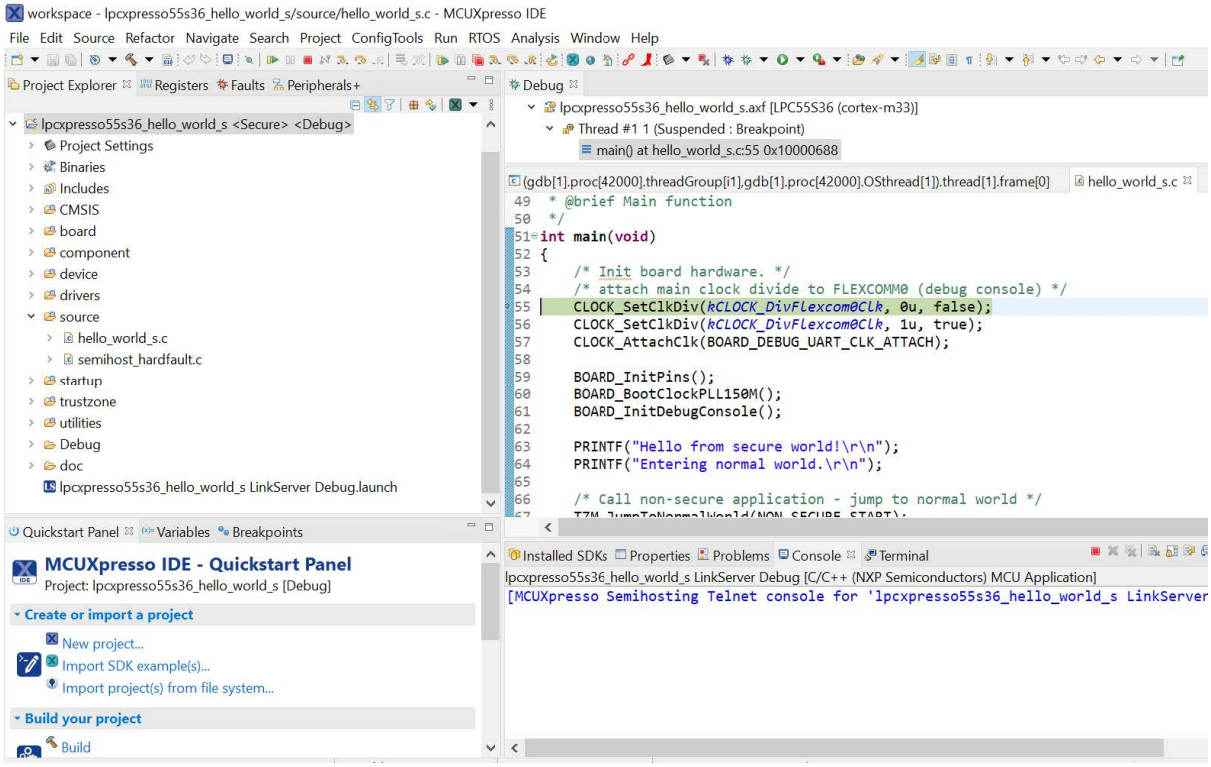
Run a TrustZone example application To download and run the application perform all steps as described in Run an example application. These steps are common for single core, dual-core, and TrustZone applications, ensuring both sides of the TrustZone application are properly loaded and started secure application. However, there is one additional dialogue that is specific to TrustZone examples. See *Figure 1* and *Figure 2* as reference.



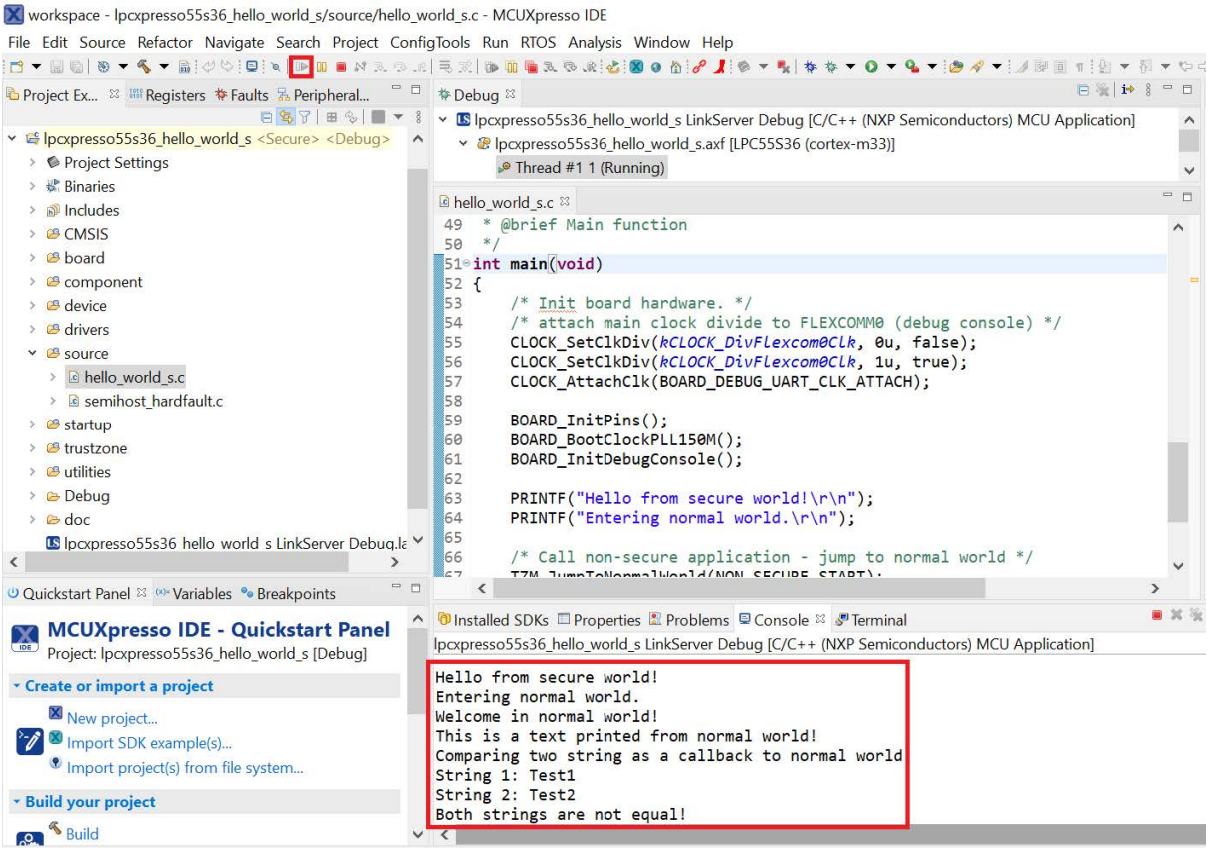


After loading the non-secure application, press **RESET** on board to release the device connect. Then, highlight the mcxn9xxevk_trustzone_examples_hello_world_s project (TrustZone master project) in the Project Explorer. In the Quickstart Panel, click **mcxn9xxevk_trustzone_examples_hello_world_s [Debug]** to launch the second debug session.


workspace - mcxn9xxevk_hello_world_ns/source/hello_world_ns.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project Run RTOS Analysis Window Help
Project Explorer
Registers
Faults
Peripherals+
mcxn9xxevk_hello_world_ns <Non-Secure>
mcxn9xxevk_hello_world_s <Secure> <Debug>
hello_world_ns.c
Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: mcxn9xxevk_hello_world_s [Debug]
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...
Build your project
Build
Clean
Debug your project
Debug
Terminate, Build and Debug
Miscellaneous
Edit project settings
Quick Settings>>
Export project(s) to archive (zip)
Export project(s) and references to archive (zip)
Build all projects
Debug using LinkServer probes (Ctrl+Alt+Shift+S)
Attach to a running target using LinkServer (Ctrl+Alt+S)
Debug with LinkServer probes
Program flash action using LinkServer
Erase flash action using LinkServer
Installed SDKs
Properties
Problems
Console
Terminal
Image Info
Debugger Console
Offline Peripher
ole for 'mcxn9xxevk_hello_world LinkServer Debug' started


workspace - lpcxpresso55s36_hello_world_s/source/hello_world_s.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run RTOS Analysis Window Help
Project Explorer
Registers
Faults
Peripherals+
lpcxpresso55s36_hello_world_s <Secure> <Debug>
Project Settings
Binaries
Includes
CMSIS
board
component
device
drivers
source
hello_world_s.c
semihost_hardfault.c
startup
trustzone
utilities
Debug
doc
lpcxpresso55s36_hello_world_s LinkServer Debug.launch
Debug
lpcxpresso55s36_hello_world_s.axf [LPC55S36 (cortex-m33)]
Thread #1 1 (Suspended : Breakpoint)
main() at hello_world_s.c:55 0x10000688
hello_world_s.c
* @brief Main function
int main(void)
{
/* Init board hardware. */
/* attach main clock divide to FLEXCOMM0 (debug console) */
CLOCK_SetClkDiv(kCLOCK_DivFlexcom0Clk, 0u, false);
CLOCK_SetClkDiv(kCLOCK_DivFlexcom0Clk, 1u, true);
CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);
BOARD_InitPins();
BOARD_BootClockPLL150M();
BOARD_InitDebugConsole();
PRINTF("Hello from secure world!\r\n");
PRINTF("Entering normal world.\r\n");
/* Call non-secure application - jump to normal world */
Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: lpcxpresso55s36_hello_world_s [Debug]
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...
Build your project
Build
Installed SDKs
Properties
Problems
Console
Terminal
lpcxpresso55s36_hello_world_s LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
[MCUXpresso Semihosting Telnet console for 'lpcxpresso55s36_hello_world_s LinkServer

Start the application by clicking **Resume**. The hello_world TrustZone application then starts running, and the secure application starts the non-secure application during run time.



Parent topic:*Run a demo application using MCUXpresso IDE*

Run a demo application using IAR

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

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

Build an example application

Do the following steps to build the hello_world example application.

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

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

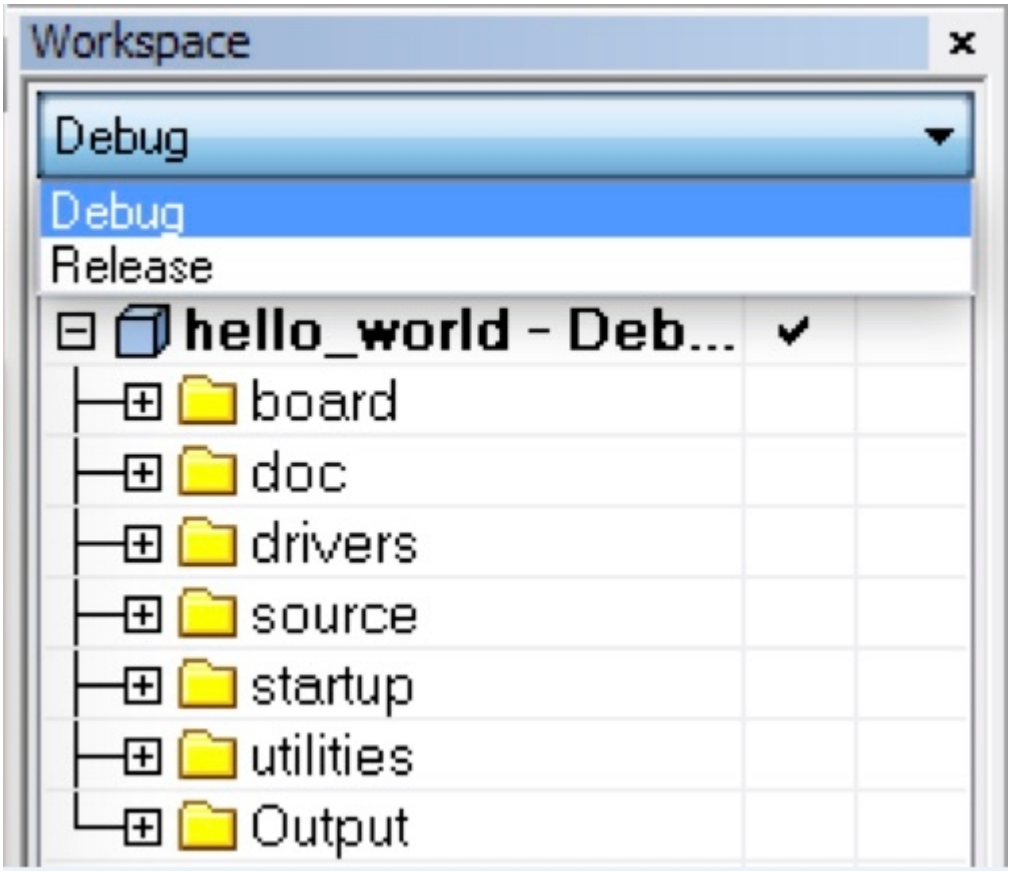
   Using the mcxn9xxevk Freedom hardware platform as an example, the hello_world workspace is located in:

   ```
   <install_dir>/boards/mcxn9xxevk/demo_apps/hello_world/iar/hello_world.eww
   ```

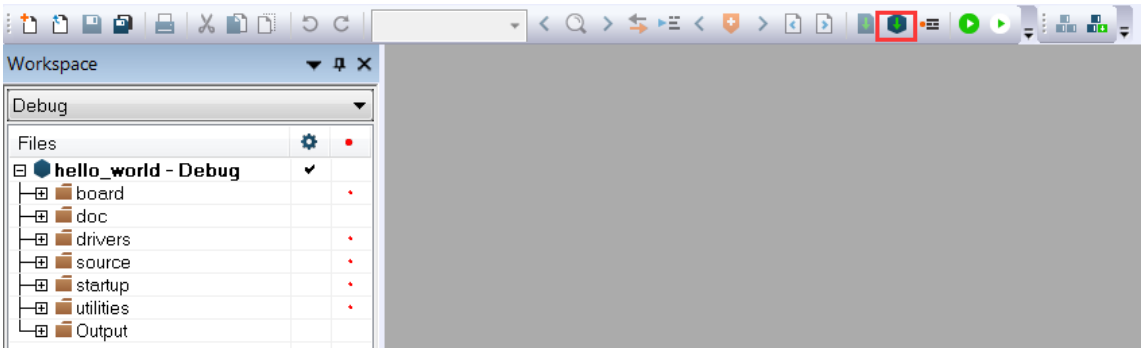
   Other example applications may have additional folders in their path.

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

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



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



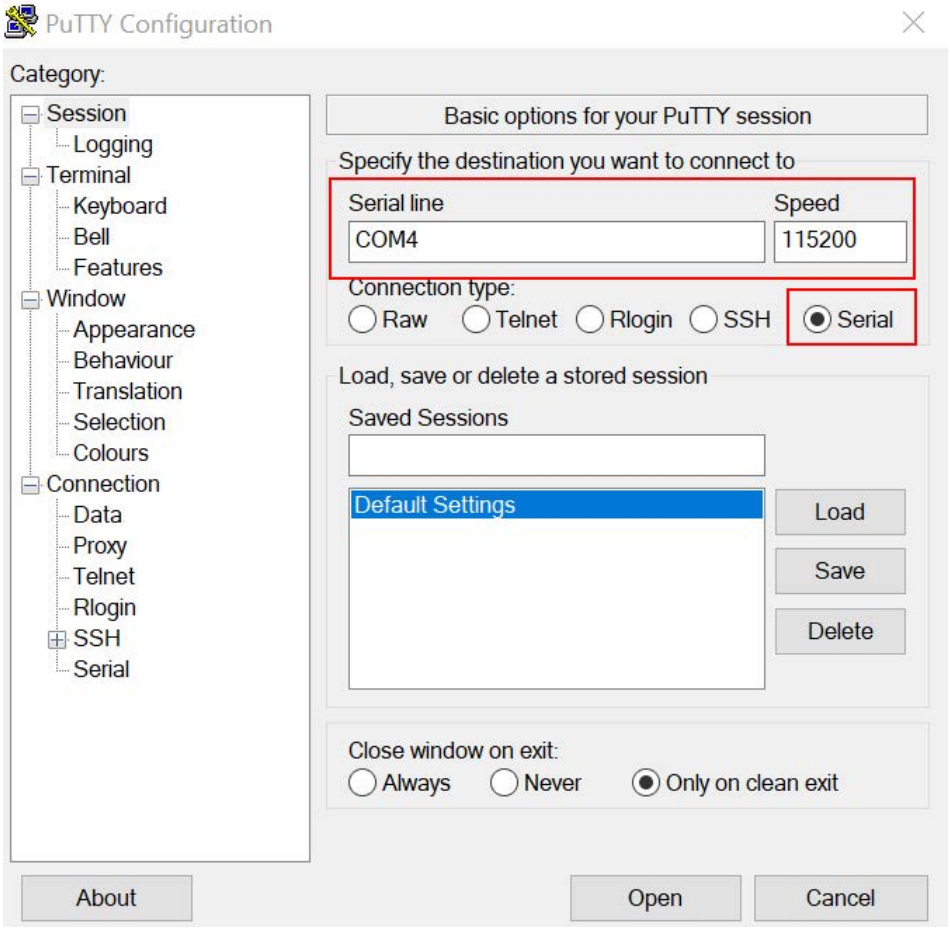
4. The build completes without errors.

Parent topic:*Run a demo application using IAR*

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

1. See *Table 1* to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with CMSIS-DAP/mbed/DAPLink interfaces, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
   - For boards with P&E Micro interfaces, visit www.pemicro.com/support/downloads_find.cfm and download the P&E Micro Hardware Interface Drivers package.
2. Connect the development platform to your PC via USB cable.

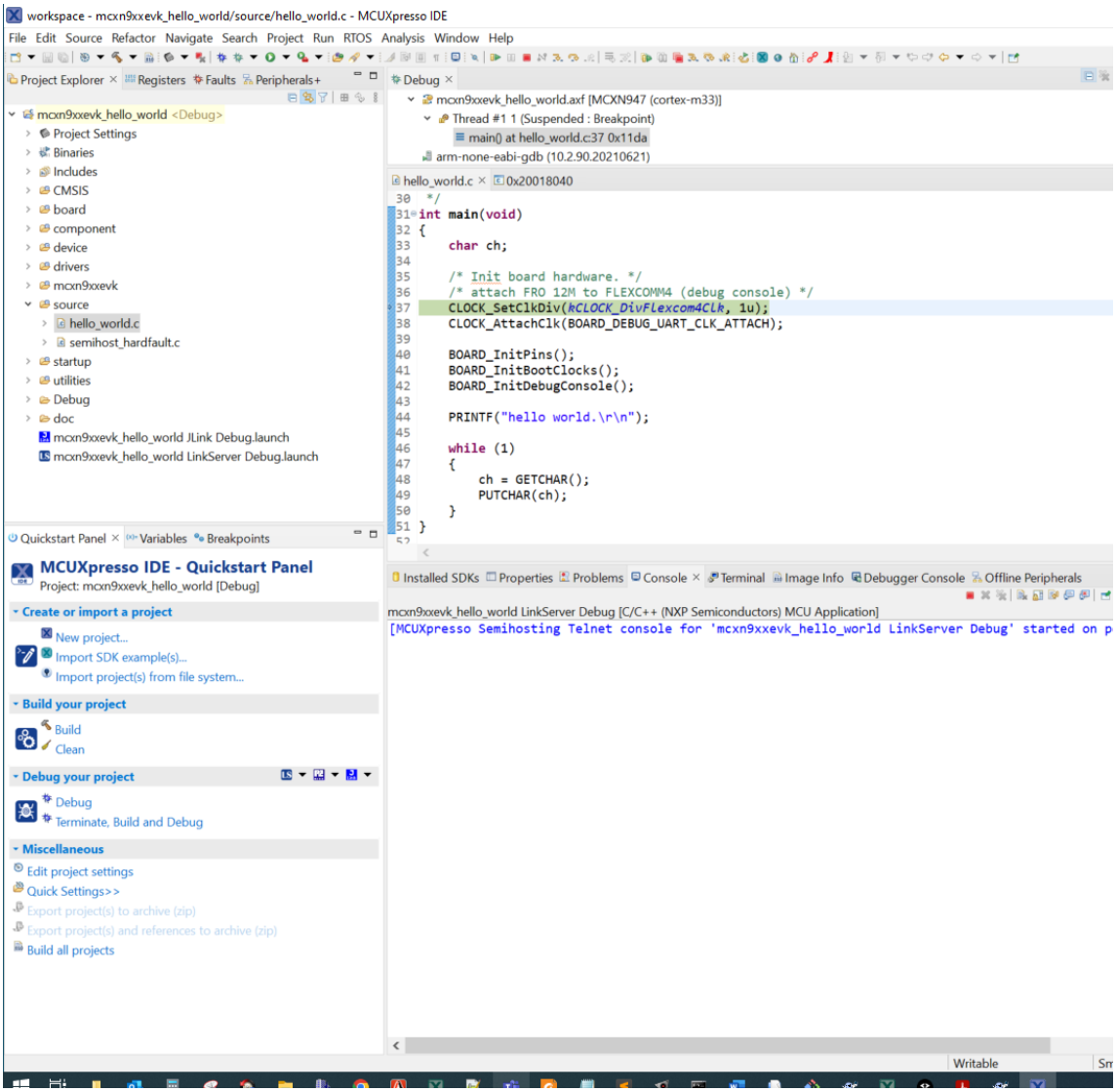
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see *How to determine com port*. Configure the terminal with the MCX-N9XX-EVK settings:
    1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
    2. No parity
    3. 8 data bits
    4. 1 stop bit



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



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



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



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



Parent topic:*Run a demo application using IAR*

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

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

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

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

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_ns/iar/hello_
↪world_ns.eww
```

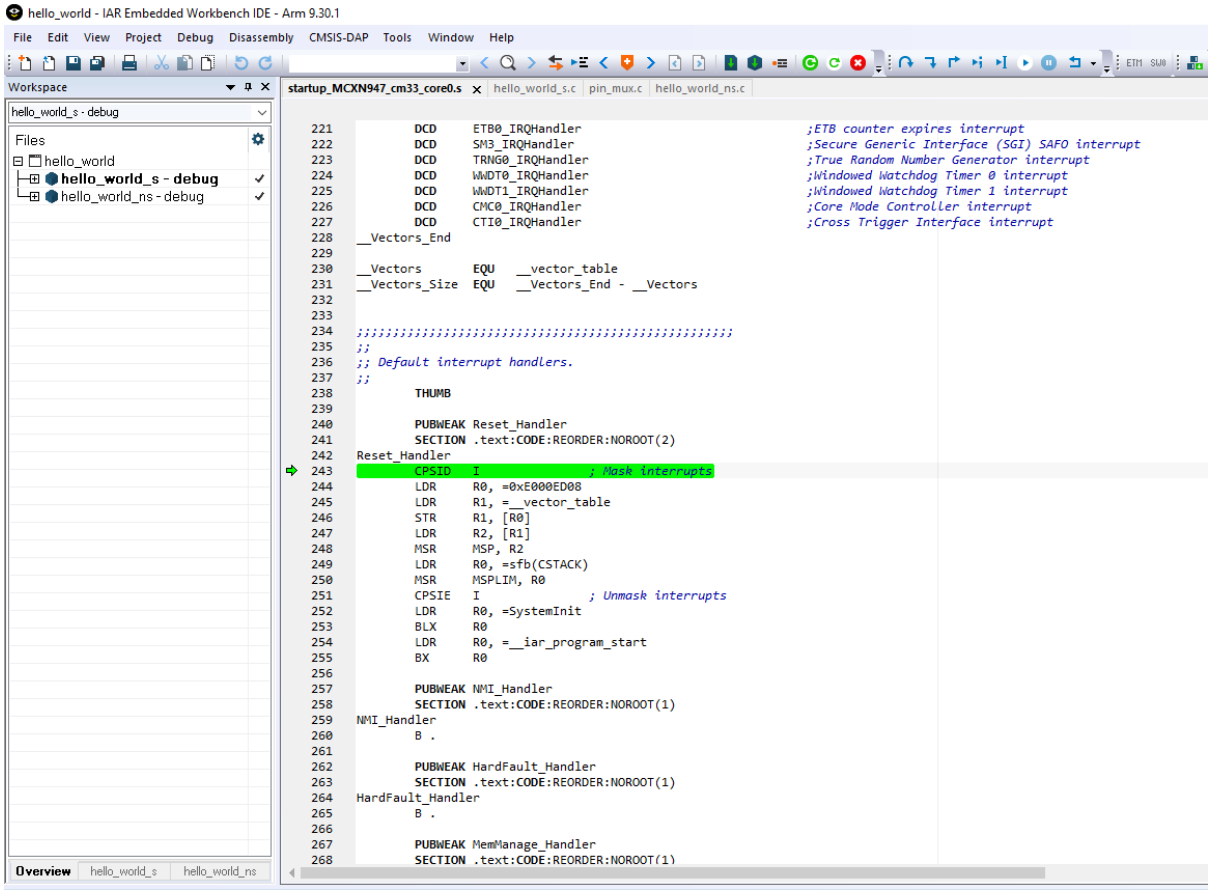
```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_s/iar/hello_
↪world_s.eww
```

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_s/iar/hello_
↪world.eww
```

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

Parent topic:*Run a demo application using IAR*

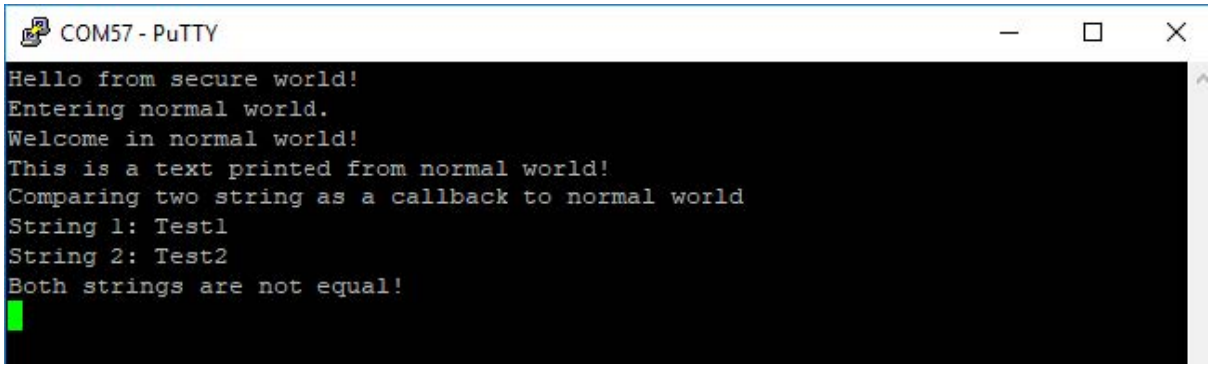
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files, so debugging can be fully managed from the secure project. To download and run the TrustZone application, switch to the secure application project and perform Steps 1 – 4 as described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



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



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



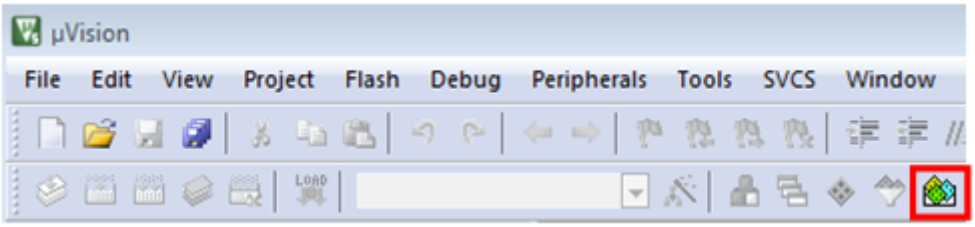
Parent topic:*Run a demo application using IAR*

Run a demo using Keil MDK/µVision

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the MCX-N9XX-EVK hardware platform is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

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

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



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

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/mcxn9xxevk/demo_apps/hello_world/mdk/hello_world.uvmpw
```

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



3. The build completes without errors.

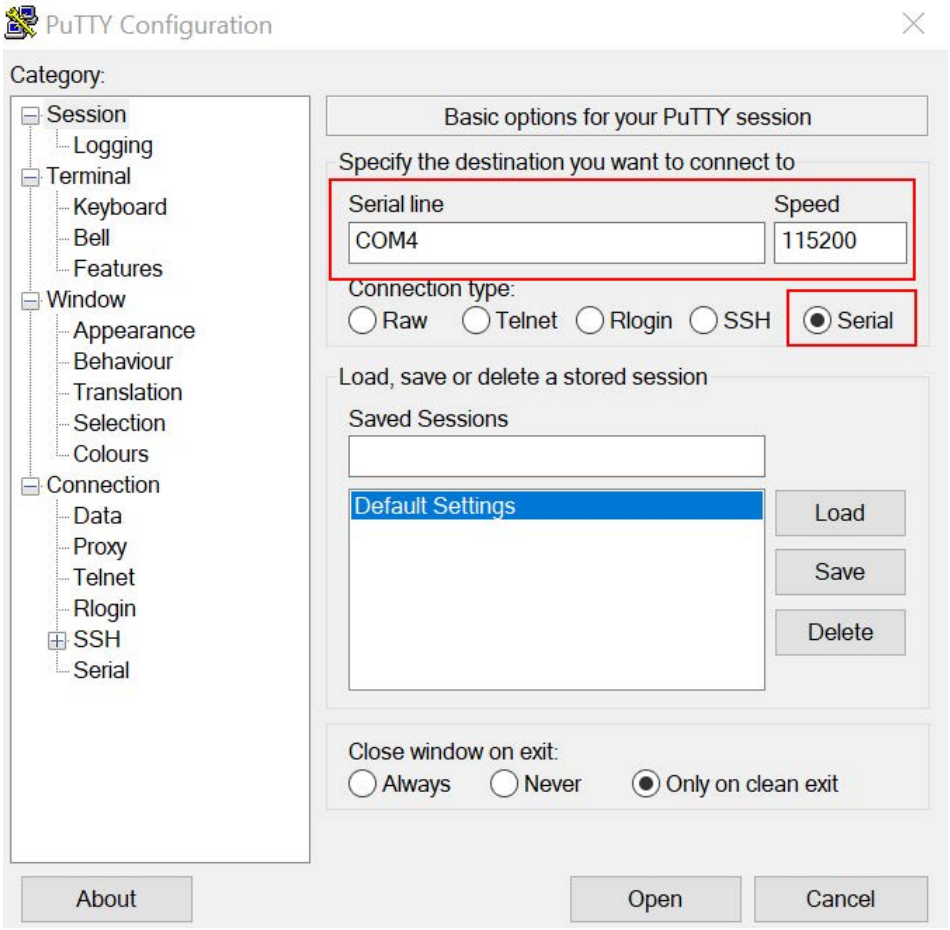
Parent topic:*Run a demo using Keil MDK/µVision*

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

1. See *Table 1* to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with CMSIS-DAP/mbed/DAPLink interfaces, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
   - For boards with P&E Micro interfaces, visit www.pemicro.com/support/downloads_find.cfm and download the P&E Micro Hardware Interface Drivers package.
   - If using J-Link either a standalone debug pod or OpenSDA, install J-Link software (drivers and utilities) from https://www.segger.com/downloads/jlink/.
   - If using J-Link either a standalone debug pod or J-link firmware programmed into the on-board debug probe, install the J-Link software (drivers and utilities) from https://www.segger.com/downloads/jlink/.

- For boards with the OSJTAG interface, install the driver from https://www.keil.com/download/.

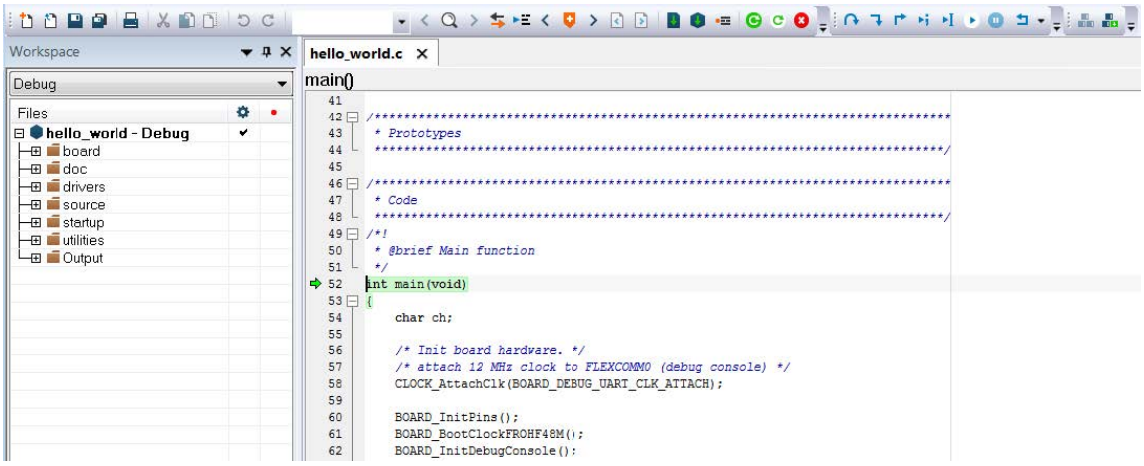
2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see *How to determine com port*. Configure the terminal with the MCX-N9XX-EVK settings:
   1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
   2. No parity
   3. 8 data bits
   4. 1 stop bit



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



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



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



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



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

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

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

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

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World Keil MSDK/µVision

workspaces are located in this folder:

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_ns/mdk/
hello_world_ns.uvmpw
```

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_s/mdk/
hello_world_s.uvmpw
```

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_s/mdk/hello_
↪world.uvmpw
```

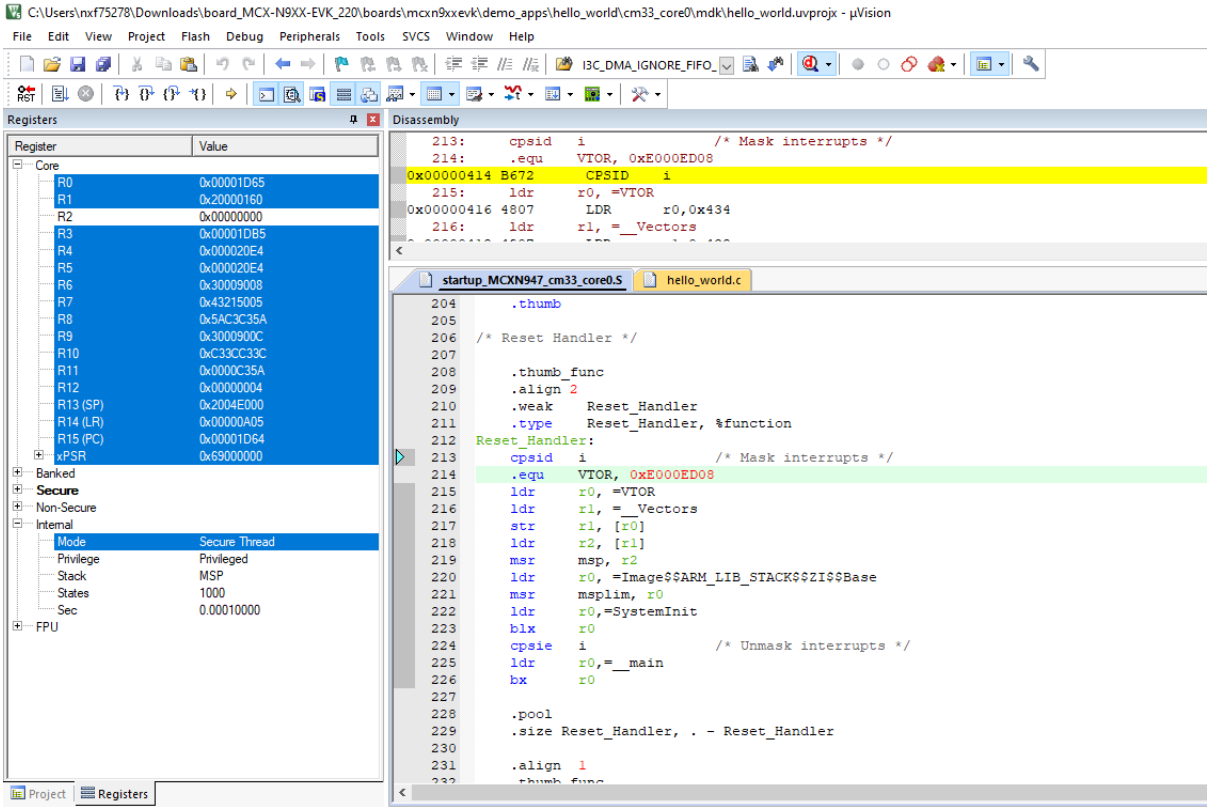
This project hello_world.uvmpw contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another.

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

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

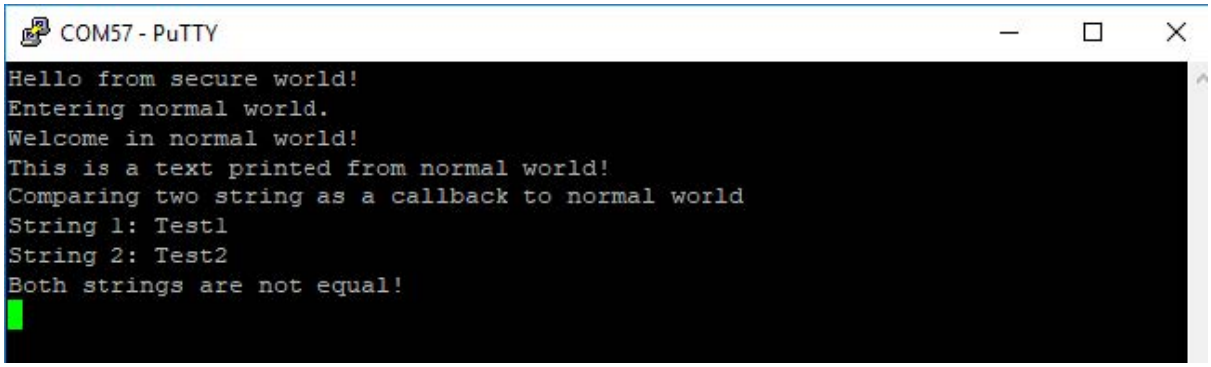
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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



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

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



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

Run a demo using Arm GCC

This section describes the steps to configure the command line Arm GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The hello_world demo application is targeted for the MCX-N9XX-EVK hardware platform which is used as an example.

Note: Arm GCC version 7-2018-q2 is used as an example in this document. The latest GCC version for this package is as described in the *MCUXpresso SDK Release Notes for MCX-N9XX-EVK* (document MCUXSDKMCXN9XXRN).

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

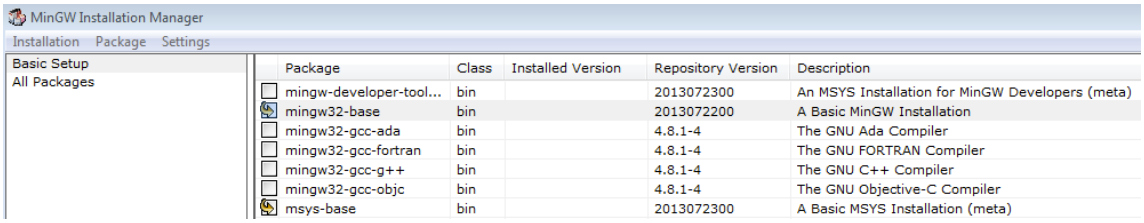
Install GCC Arm Embedded tool chain Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes*.

Parent topic: Set up toolchain

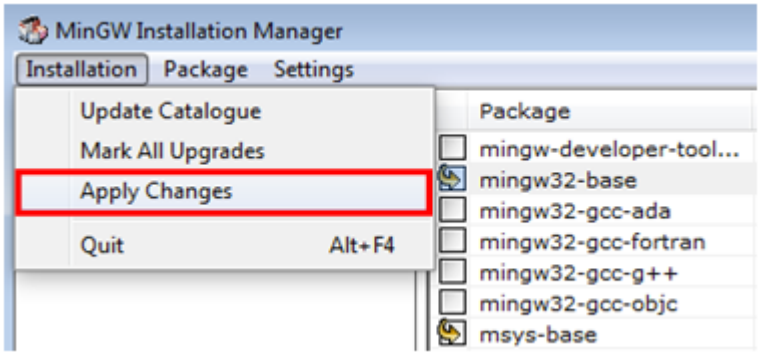
Install MinGW (only required on Windows OS) The Minimalist GNU for Windows (MinGW) development tools provide a set of tools that are not dependent on third-party C-Runtime DLLs (such as Cygwin). The build environment used by the MCUXpresso SDK does not use the MinGW build tools, but does leverage the 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 MinGW.
2. Run the installer. The recommended installation path is C:\MinGW, however, you may install to any location.

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



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

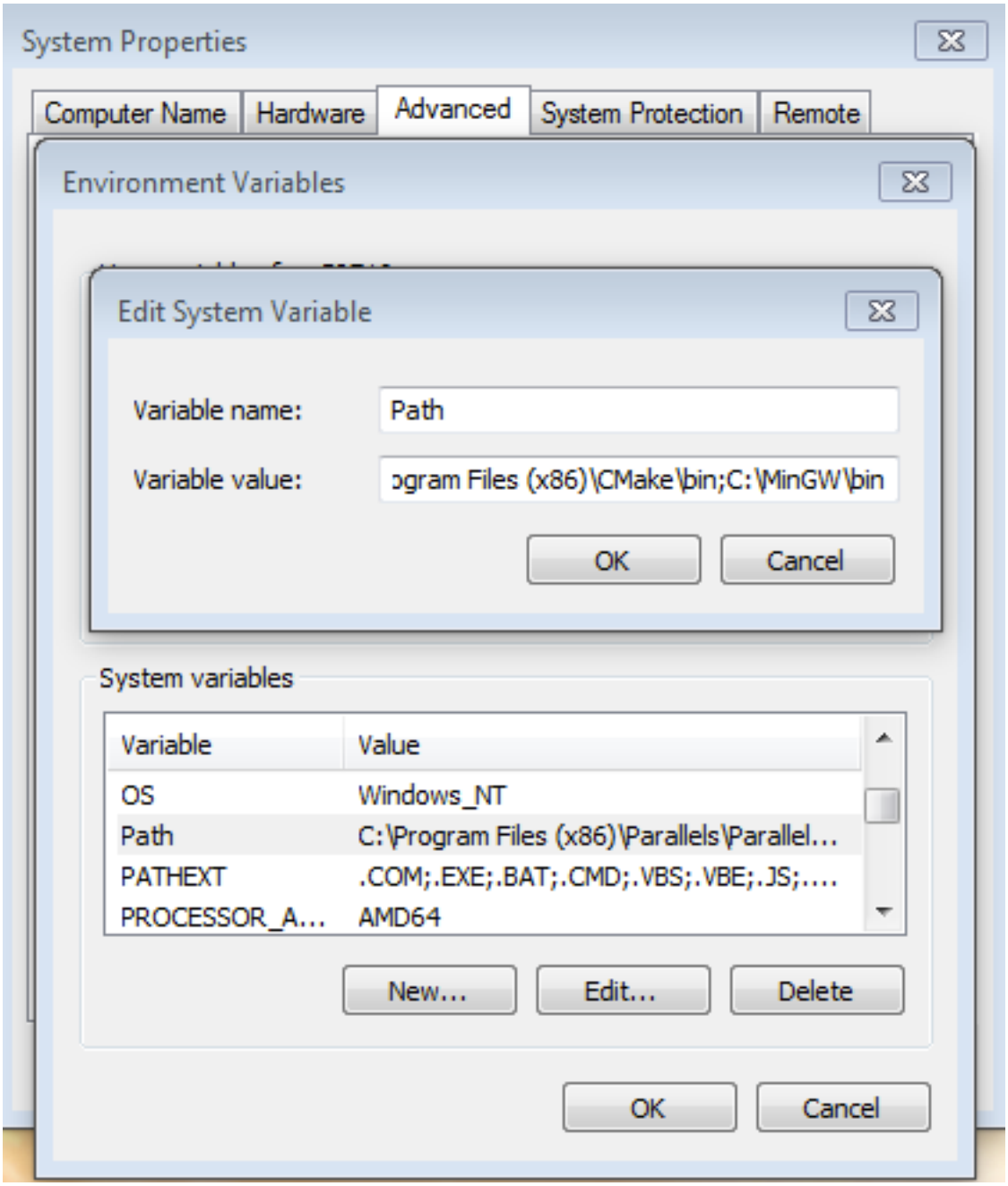


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:

```
<mingw_install_dir>\bin
```

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

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



Parent topic: Set up toolchain

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

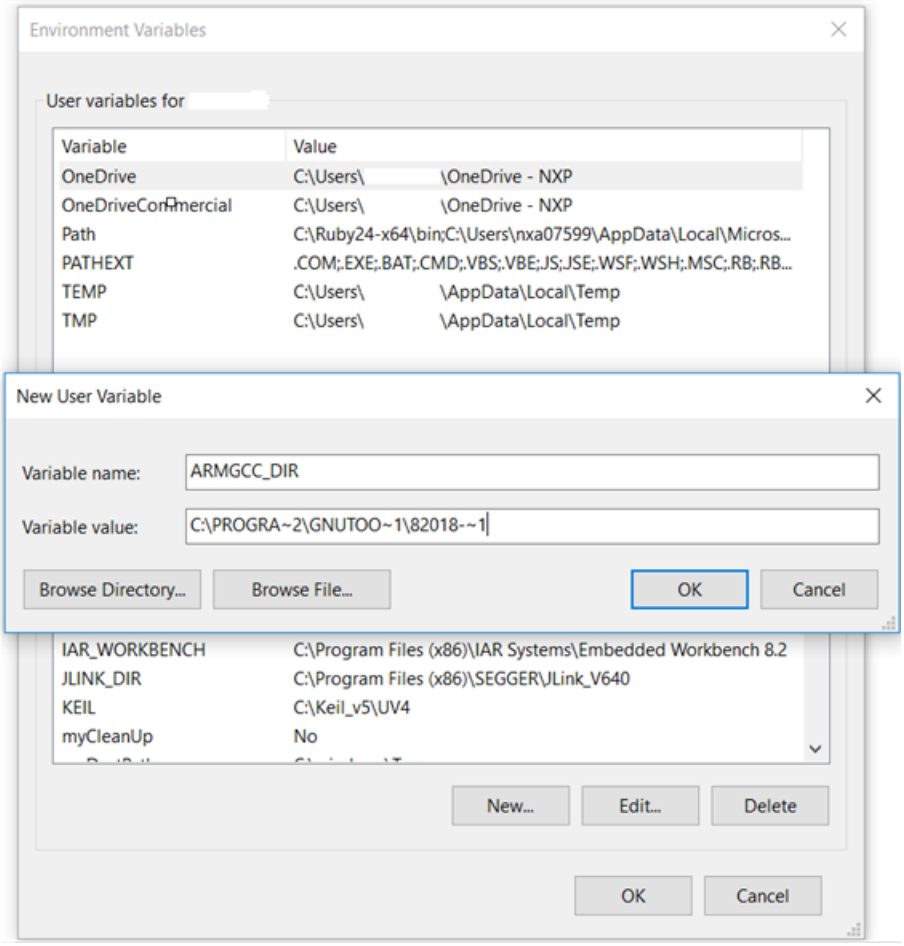
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major
```

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 command `for %I in (.) do echo %~sI` in above path.

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

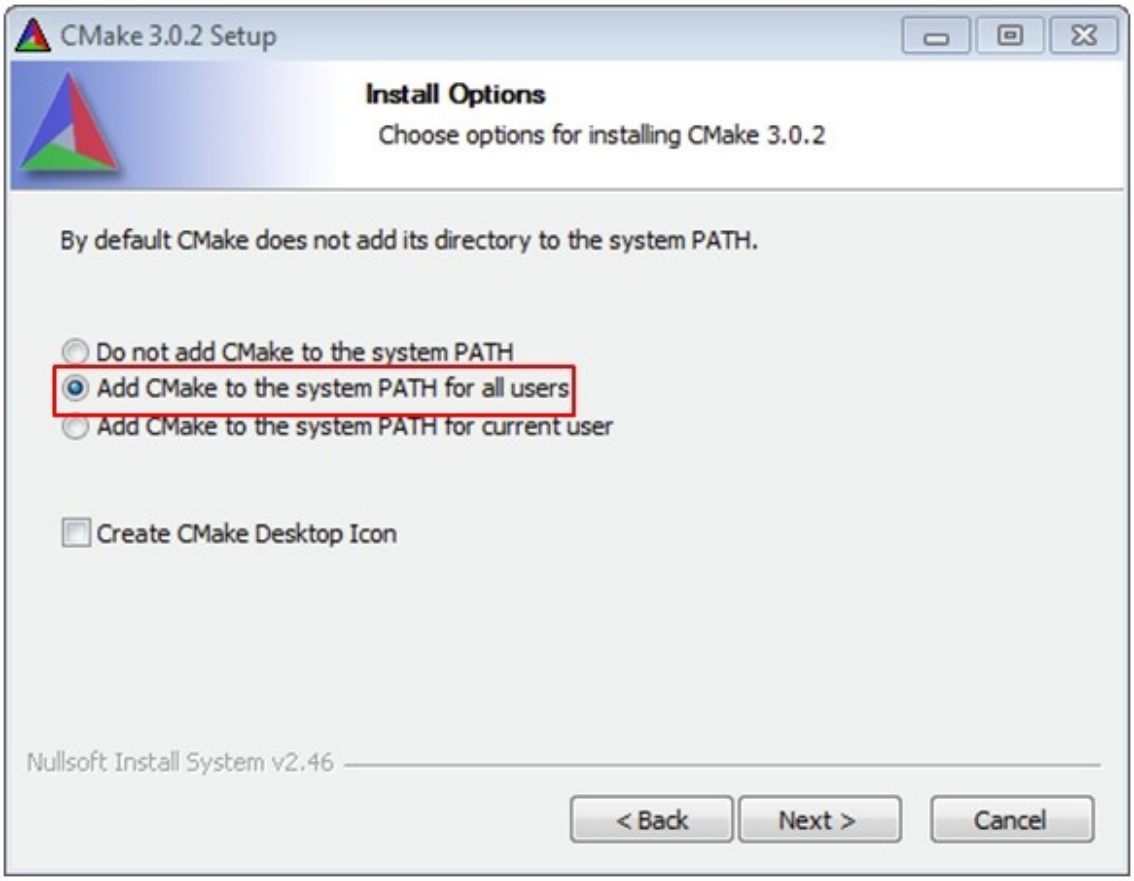
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018-~1
C:\PROGRA~2\GNUTOO~1\82018-~1
```



Parent topic: Set up toolchain

Install CMake

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



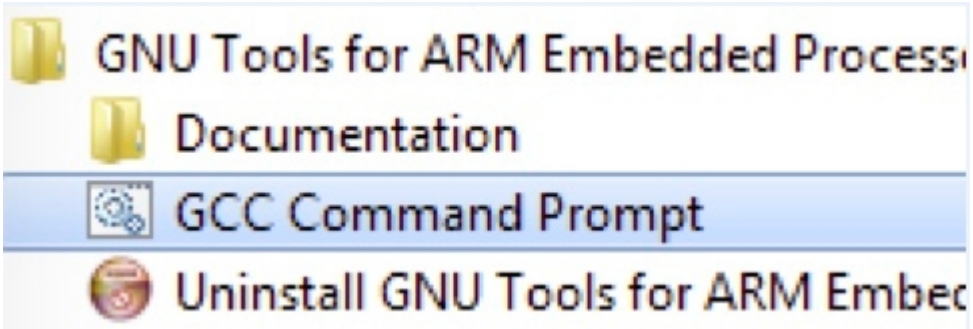
3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.
5. Make sure sh.exe is not in the Environment Variable PATH. This is a limitation of mingw32-make.

Parent topic: Set up toolchain

Parent topic:*Run a demo using Arm GCC*

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

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



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

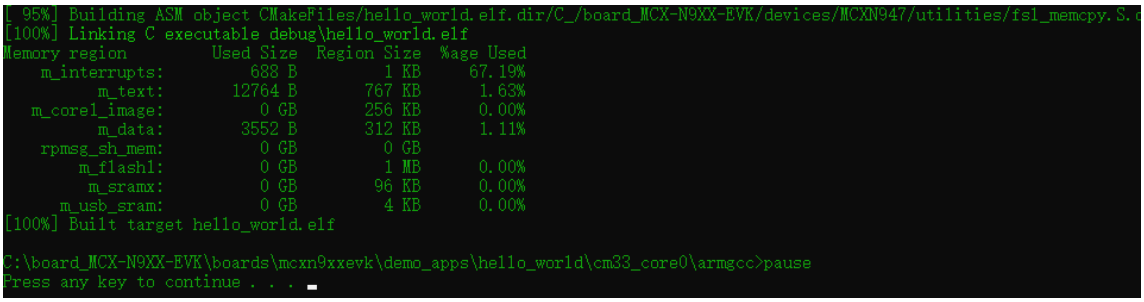
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc
```

For this example, the exact path is:

```
<install_dir>/examples/mcxn9xxevk/demo_apps/hello_world/armgcc
```

Note: To change directories, use the cd command.

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



Parent topic:*Run a demo using Arm GCC*

Run an example application This section describes steps to run a demo application using J-Link GDB Server application.

To complete the set-up check if your board supports OpenSDA, see *Default debug interfaces*.

- If your board supports OpenSDA:
  - The OpenSDA interface on your board is pre-programmed with the J-Link OpenSDA firmware.
  - For instructions on reprogramming the OpenSDA interface.
- If your board does not support OpenSDA:
  - A standalone J-Link pod is required which should be connected to the debug interface of your board.

    Note: Some hardware platforms require hardware modification in order to function correctly with an external debug interface.

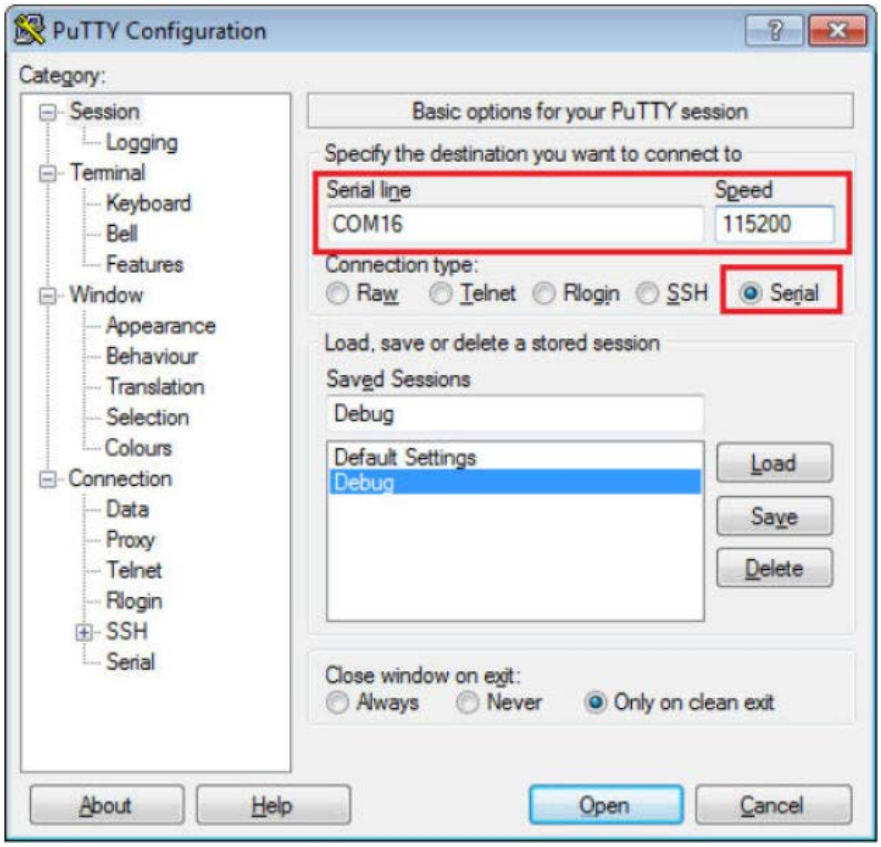
    Note: J-Link GDB Server application is not supported for TFM examples. Use CMSIS DAP instead of J-Link for flashing and debugging TFM examples.
  - After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:
    1. Connect the development platform to your PC via USB cable between the LPC-Link2 USB connector (may be named OSJTAG for some boards) and the PC USB connector. If using a standalone J-Link debug pod, connect it to the SWD/JTAG connector of the board.
    2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see *How to determine com port*.
    3. Configure the terminal with these settings:

       a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)

       b. No parity

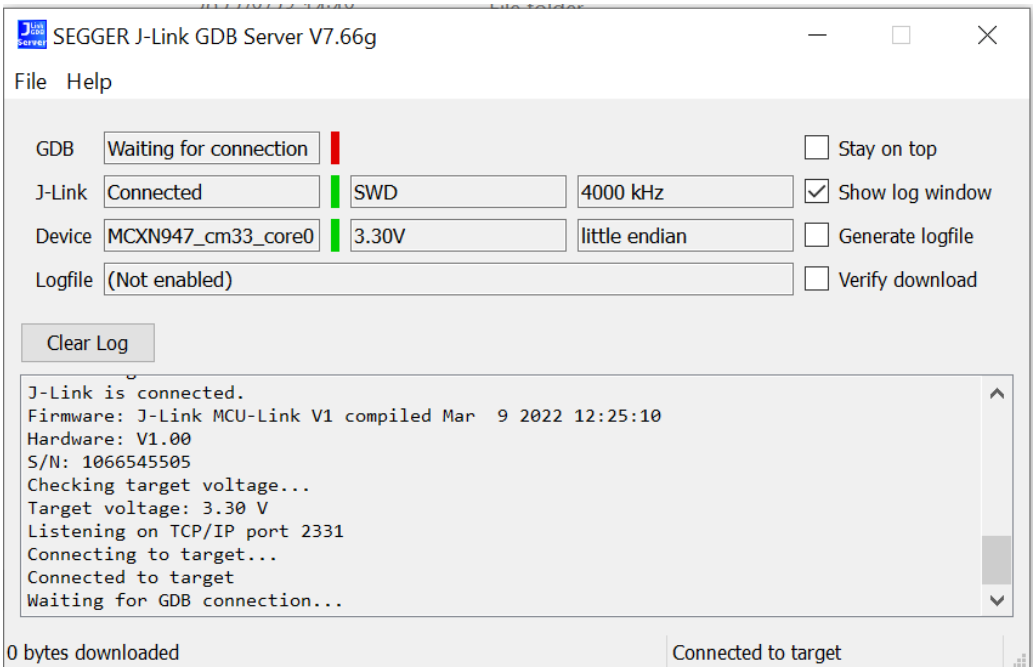
       c. 8 data bits

d. 1 stop bit

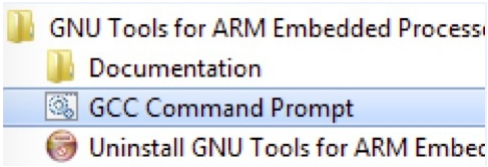


Note: Make sure the board is set to FlexSPI flash boot mode (ISP2: ISP1: ISP0 = ON, OFF, ON) before use GDB debug.

4. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system Start menu and selecting **Programs** -> **SEGGER** -> **J-Link <version> J-Link GDB Server**.
5. Modify the settings as shown below. The target device selection chosen for this example is **MCXN947_cm33_core0**.
6. After it is connected, the screen should look like *Figure 2*.



7. 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**.



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

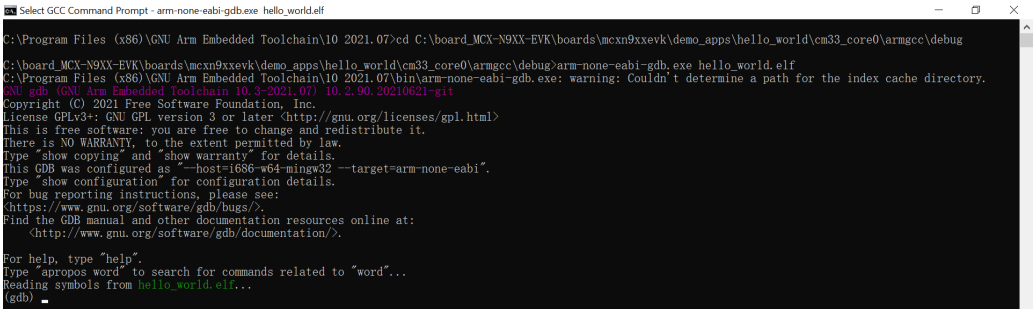
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/
↪debug
```

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

For this example, the path is:

```
<install_dir>/boards/mcxn9xxevk/demo_apps/hello_world/cm4/armgcc/debug
```

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



10. Run these commands:

a. target remote localhost:2331

b. monitor reset

c. monitor halt

d. load

e. monitor reset

11. The application is now downloaded and halted at the watch point. Execute the 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 does not appear, check your terminal settings and connections.



Parent topic:*Run a demo using Arm GCC*

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

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

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

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

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_ns.
↪eww
```

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/hello_world_s/iar/hello_world_s.eww
```

```
<install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/hello_world_s/iar/hello_world.eww
```

Build both applications separately, following steps for single core examples as described in Build an example application. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project, since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because the CMSE library is not ready.

```
[ 25%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_s/boa
rd.c.obj
[ 29%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_s/clo
ck_config.c.obj
[ 33%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/str/fsl_str.c.obj
[ 37%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/debug_console_lite/fsl_debug_console.c.obj
[ 41%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/components/uart/fsl_adapter_lpuart.c.obj[ 45%] Building C object CMakeFiles/h
ello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_common.c.obj[ 50%] [ 58%] [ 58%] Building C object CMakeFiles/hello_world_s.elf.dir/C_
/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_common_arm.c.obj[ 62%] s/fsl_lpflexcomm.c.objdevices/MCXN947/drivers/fsl_lpuart.c.obj[ 66%] Building C object
 CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_clock.c.obj

Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_reset.c.obj


[ 70%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/debug_console_lite/fsl_assert.c.obj
[ 75%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_gpio.c.obj
[ 79%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/system_MCXN947_cm33_core0.c.obj
[ 83%] Building ASM object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/gcc/startup_MCXN947_cm33_core0.S.obj
[ 87%] Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/components/lists/fsl_component_generic_list.c.obj
[ 91%] Building ASM object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/fsl_memcpy.S.obj[ 95%]
Building C object CMakeFiles/hello_world_s.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/fsl_sbrk.c.obj
[100%] Linking C executable debug\hello_world_s.elf
Memory region         Used Size  Region Size  %age Used
    m_interrupts:         688 B         1 KB     67.19%
          m_text:       14564 B     129536 B     11.24%
  m_veneer_table:          32 B       512 B      6.25%
   m_core1_image:           0 GB       256 KB      0.00%
          m_data:        3552 B        32 KB     10.84%
    rpmsg_sh_mem:           0 GB         0 GB
         m_flash1:          0 GB         1 MB      0.00%
          m_sramx:          0 GB        96 KB      0.00%
     m_usb_sram:           0 GB         4 KB      0.00%
[100%] Built target hello_world_s.elf

C:\board_MCX-N9XX-EVK\boards\mcxn9xxevk\trustzone_examples\hello_world\cm33_core0\hello_world_s\armgcc>pause
Press any key to continue . . .
```

```
[ 23%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/str/fsl_str.c.obj
[ 28%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/debug_console_lite/fsl_debug_console.c.obj
[ 33%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/components/uart/fsl_adapter_lpuart.c.obj
[ 38%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_common.c.obj
[ 42%] [ 47%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_common_arm.c.objBuilding C object CMakeFi
les/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_reset.c.obj
[ 52%]
Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_clock.c.obj
[ 57%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_lpuart.c.obj
[ 61%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/drivers/fsl_lpflexcomm.c.obj
[ 66%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/ert.c.obj □j[ 76%] Building ASM object CMakeFiles
/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/gcc/startup_MCXN947_cm33_core0.S.obj[ 80%] Building C object CMakeFiles/hello_world_ns.elf.dir/
C_/board_MCX-N9XX-EVK/devices/MCXN947/system_MCXN947_cm33_core0.c.obj

[ 85%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/components/lists/fsl_component_generic_list.c.obj
[ 90%] Building C object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/fsl_sbrk.c.obj
[ 95%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/C_/board_MCX-N9XX-EVK/devices/MCXN947/utilities/fsl_memcpy.S.obj
[100%] Linking C executable debug\hello_world_ns.elf
Memory region         Used Size  Region Size  %age Used
    m_interrupts:         688 B         1 KB     67.19%
          m_text:        4592 B       639 KB      0.70%
   m_core1_image:           0 GB       256 KB      0.00%
          m_data:        3536 B       280 KB      1.23%
    rpmsg_sh_mem:           0 GB         0 GB
         m_flash1:          0 GB         1 MB      0.00%
          m_sramx:          0 GB        96 KB      0.00%
     m_usb_sram:           0 GB         4 KB      0.00%
[100%] Built target hello_world_ns.elf

C:\board_MCX-N9XX-EVK\boards\mcxn9xxevk\trustzone_examples\hello_world\cm33_core0\hello_world_ns\armgcc>pause
Press any key to continue . . .
```

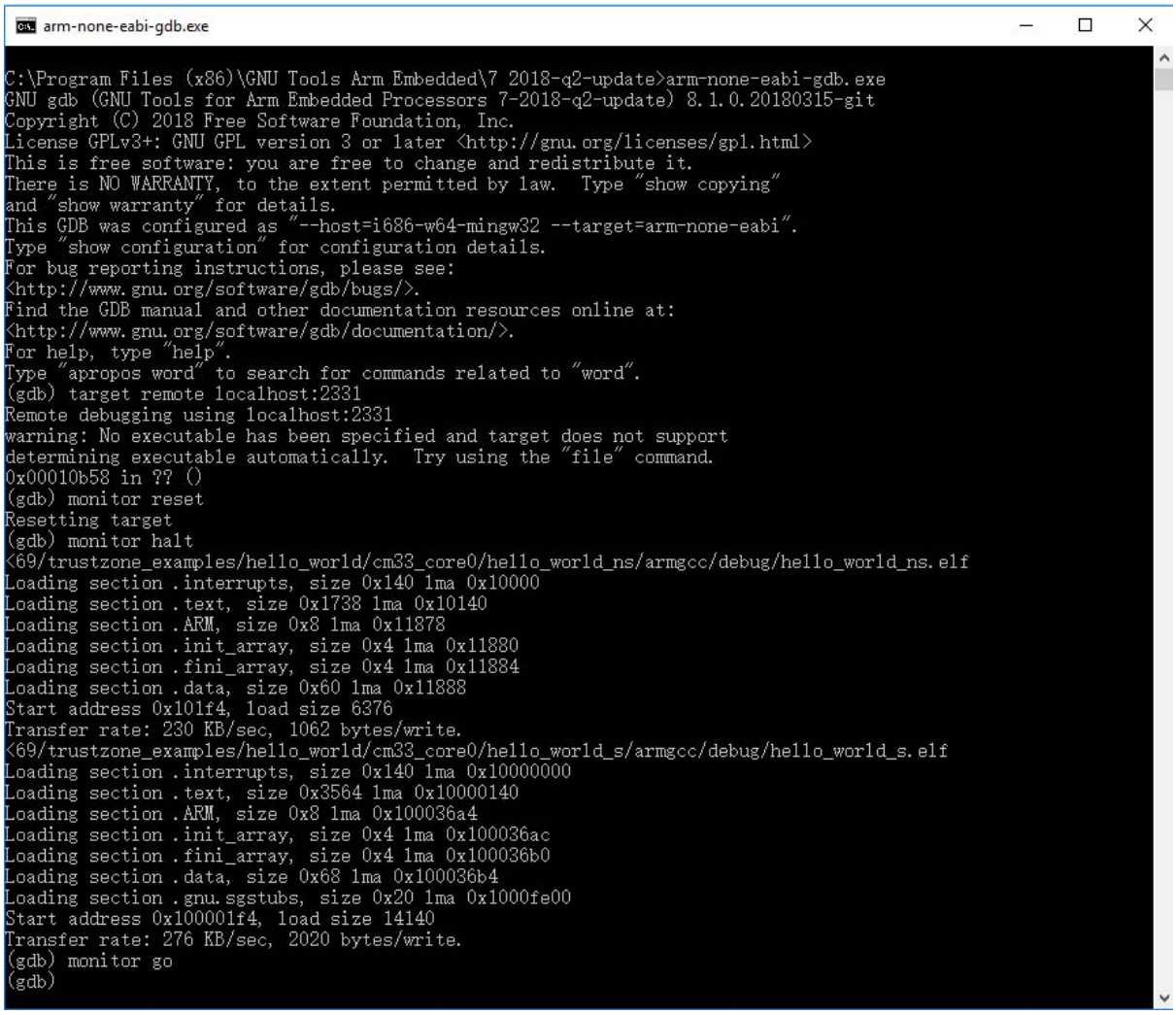
Parent topic:*Run a demo using Arm GCC*

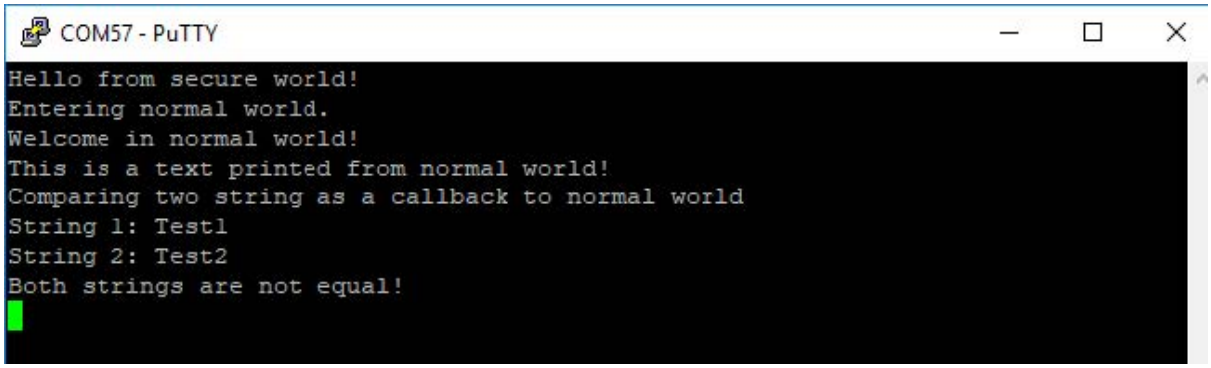
Run a TrustZone example application When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in Run an example application.

To download and run the TrustZone application, perform Steps 1 to 10, as described in Run an example application. These steps are common for both single core and trustzone applications in Arm GCC.

Then, run these commands:

1. arm-none-eabi-gdb.exe
2. target remote localhost:2331
3. monitor reset
4. monitor halt
5. load <install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_ns/armgcc/debug/hello_world_ns.elf
6. load <install_dir>/boards/mcxn9xxevk/trustzone_examples/hello_world/cm33_core0/hello_world_s/armgcc/debug/hello_world_s.elf
7. The application is now downloaded and halted at the watch point. Execute the monitor go command to start the demo application.



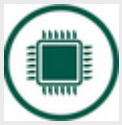


Parent topic:*Run a demo using Arm GCC*

MCUXpresso Config Tools

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

| | **Device Configuration tool** | Configures Device Configuration Data (DCD) contained in the program image that the Boot ROM code interprets to setup various on-chip peripherals prior the program launch. | 

|

MCUXpresso Config Tools can be accessed in the following products:

- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from www.nxp.com/mcuxpresso. Recommended for customers using IAR Embedded Workbench, Keil MDK µVision, or Arm GCC.
- **Online version** available on mcuxpresso.nxp.com. Recommended to do 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.

MCUXpresso IDE New Project Wizard

MCUXpresso IDE features a new project wizard. The wizard provides functionality for the user to create new projects from the installed SDKs (and from pre-installed part support). It offers user the flexibility to select and change multiple builds. The wizard also includes a library and provides source code options. The source code is organized as software components, categorized as drivers, utilities, and middleware.

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in *Figure 1*.



For more details and usage of new project wizard, see the *MCUXpresso_IDE_User_Guide.pdf* in the MCUXpresso IDE installation folder.

How to determine com port

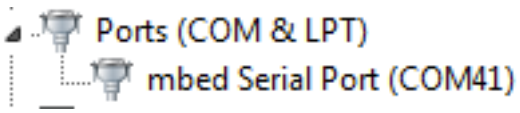
This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform. All NXP boards ship with a factory programmed, on-board debug interface, whether it's based on OpenSDA or the legacy P&E Micro OSJTAG interface. To determine what your specific board ships with, see *Default debug interfaces*.

1. **Linux**: The serial port can be determined by running the following command after the USB Serial is connected to the host:

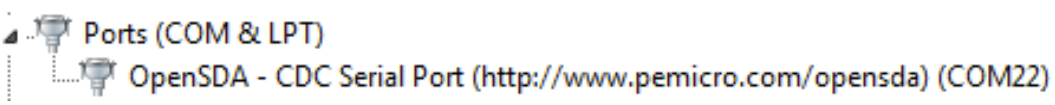
```
$ dmesg | grep "ttyUSB"
                [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
                [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
```

   There are two ports, one is Cortex-A core debug console and the other is for Cortex M4.

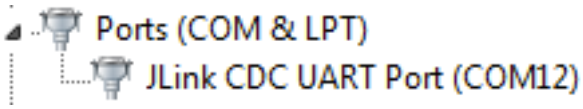
2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click on the **Start** menu and type **Device Manager** in the search bar.
3. In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names will be different for all the NXP boards.
   1. OpenSDA – CMSIS-DAP/mbed/DAPLink interface:

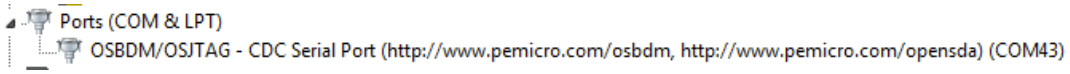
   2. **OpenSDA – P&E Micro**:

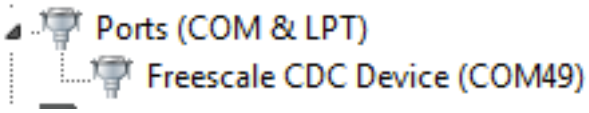
3. **OpenSDA – J-Link**:



4. **P&E Micro OSJTAG**:



5. **MRB-KW01**:



How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to override the default IRQ handler. For example, to override the default PIT_IRQHandler define in startup_DEVICE.s, application code like app.c can be implement like:

```
c
void PIT_IRQHandler(void)
{
    // Your code
}
```

When application file is CPP file, like app.cpp, then extern "C" should be used to ensure the function prototype alignment.

```
cpp
extern "C" {
    void PIT_IRQHandler(void);
}
void PIT_IRQHandler(void)
{
    // Your code
}
```

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.

Note: The *OpenSDA details* column in *Table 1* is not applicable to LPC.

Hardware platform	Default interface	OpenSDA details
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

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details
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
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
LPCXpresso55s36	CMSIS-DAP	N/A
LPCXpresso55s69	CMSIS-DAP	N/A
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0
MCX-N9XX-EVK	CMSIS-DAP	N/A
MCX-N5XX-EVK	CMSIS-DAP	N/A
MCX-N9XX-BRK	CMSIS-DAP	N/A
MIMXRT1170-EVK	CMSIS-DAP	N/A
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

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details
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
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

Updating debugger firmware

Updating MCXN board firmware The MCXN hardware platform comes with a CMSIS-DAP-compatible debug interface (known as LPC-Link2). This firmware in this debug interface may be updated using the host computer utility called LPCScrypt. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

Note: If MCUXpresso IDE is used and the jumper making DFUlink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), LPC-Link2 debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the CMSIS-DAP firmware to the probe before flash memory programming (after clicking **Debug**). Using DFU mode ensures most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the LPCScrypt utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from www.nxp.com/lpcutilities.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScrypt user guide (www.nxp.com/lpcutilities, select **LPCScrypt**, and then the documentation tab).

1. Install the LPCScript utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labelled DFUlink).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the LPCScrypt installation directory (<LPCScrypt install dir>).
    1. To program CMSIS-DAP debug firmware: <LPCScrypt install dir>/scripts/program_CMSIS

2. To program J-Link debug firmware: <LPCScrypt install dir>/scripts/program_JLINK

6. Remove DFU link (remove the jumper installed in *Step 3*).
7. Re-power the board by removing the USB cable and plugging it in again.

Parent topic:*Updating debugger firmware*

1.3 Getting Started with MCUXpresso SDK GitHub

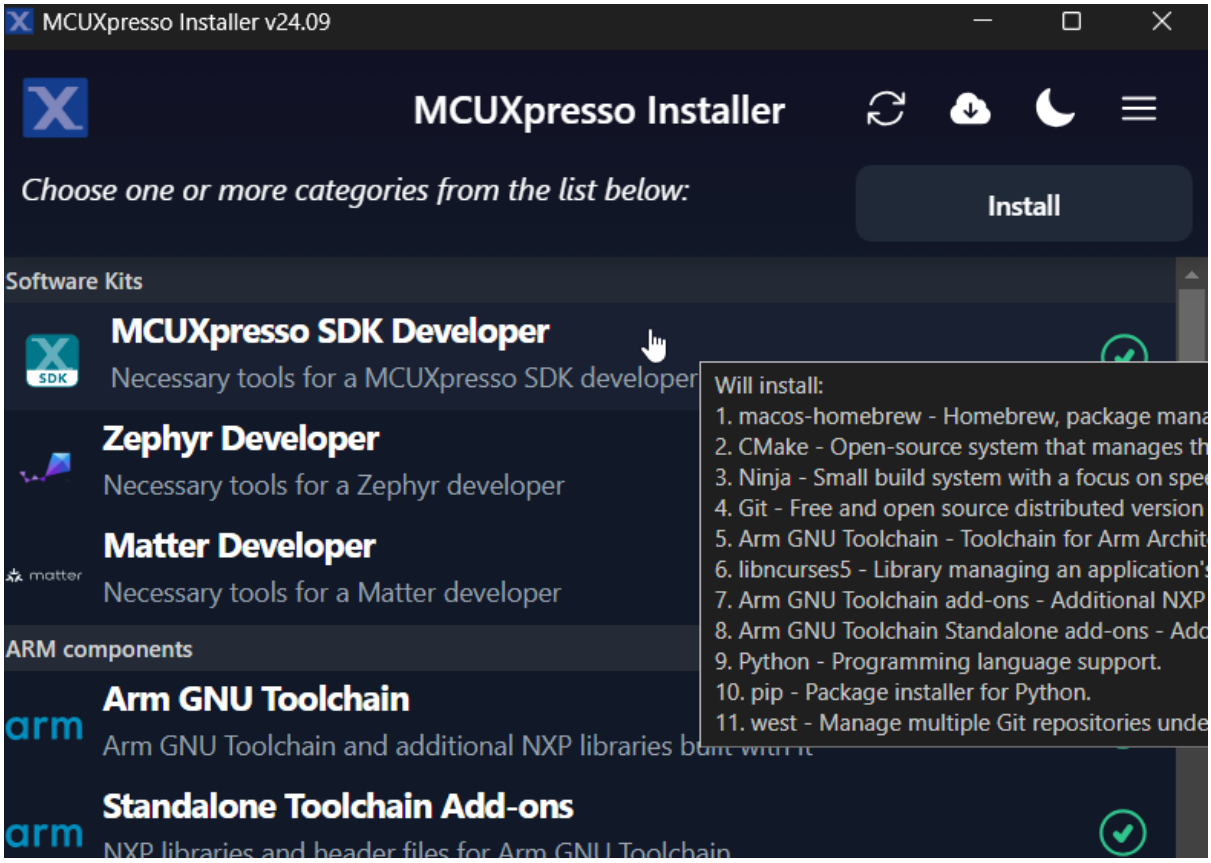
1.3.1 Getting Started with MCUXpresso SDK Repository

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.

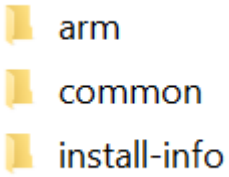
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

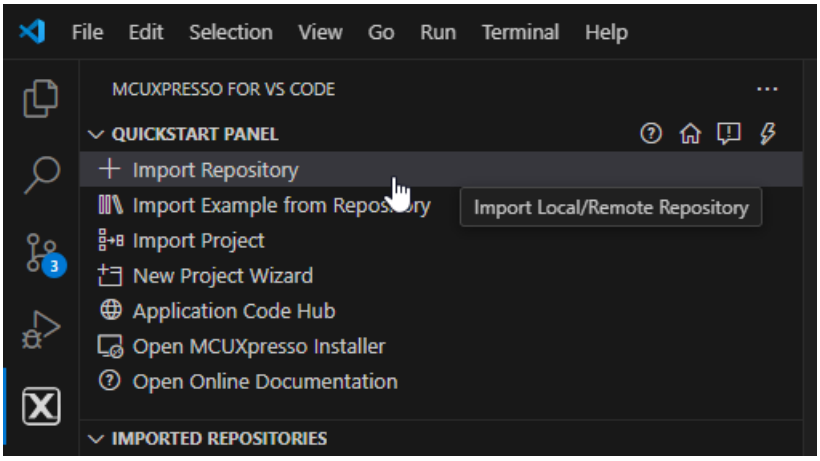
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the hello_world demo application as an example. However, these

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*.

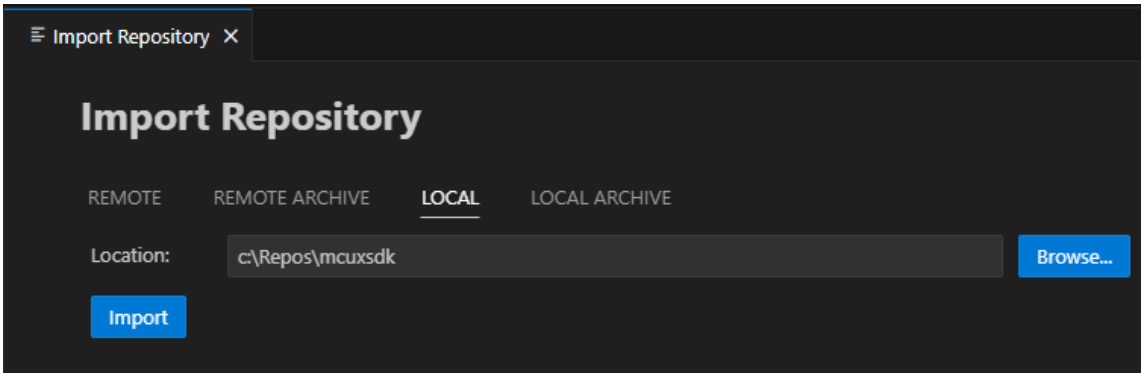
To build an example application:

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

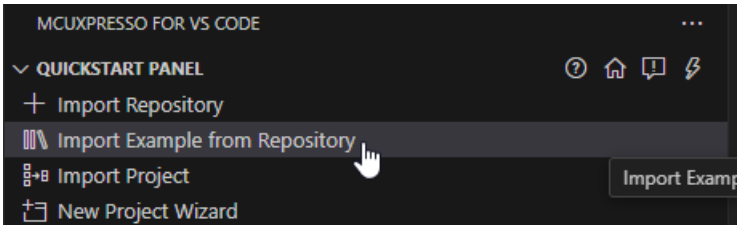


Note: You can import the SDK in several ways. Refer to MCUXpresso for VS Code Wiki for details.

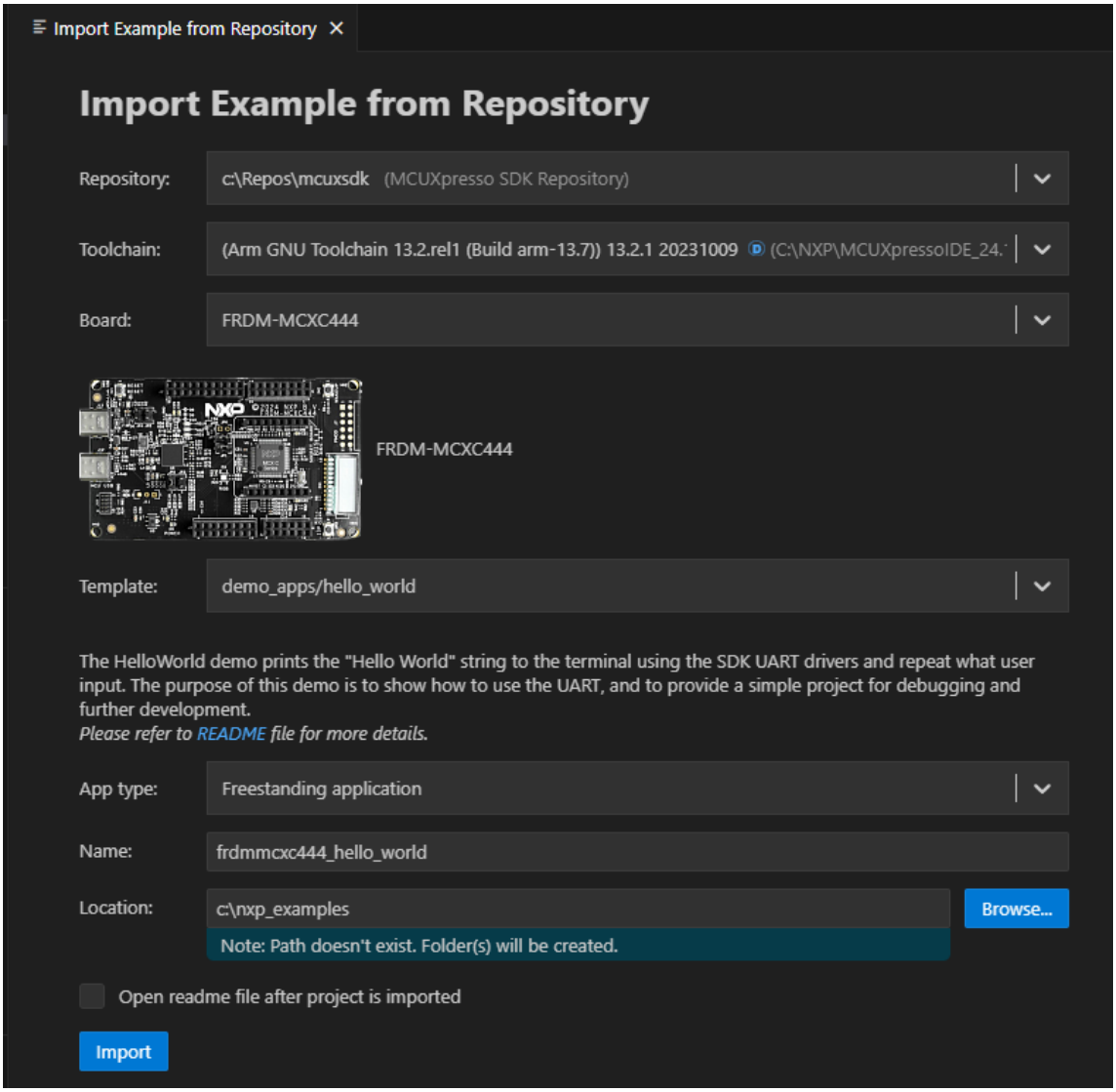
Select **Local** if you've already obtained the SDK as seen in *Get MCUXpresso SDK Repo*. Select your location and click **Import**.



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

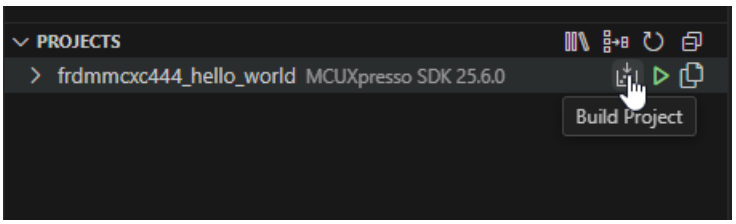


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

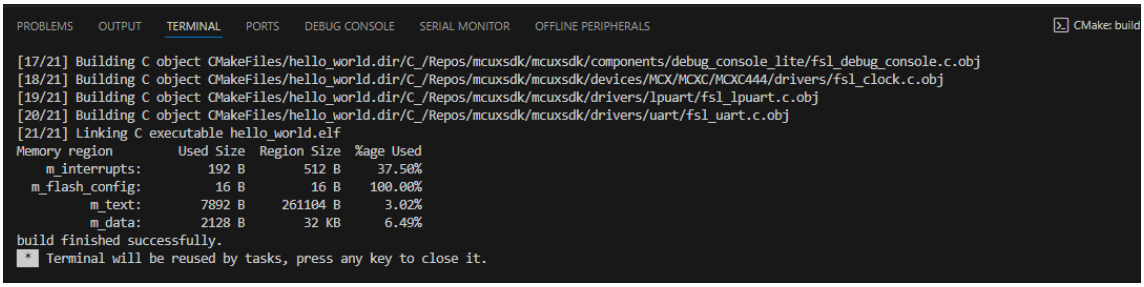


Note: The MCUXpresso SDK projects can be imported as **Repository applications** or **Freestanding applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Freestanding examples can be imported to 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.

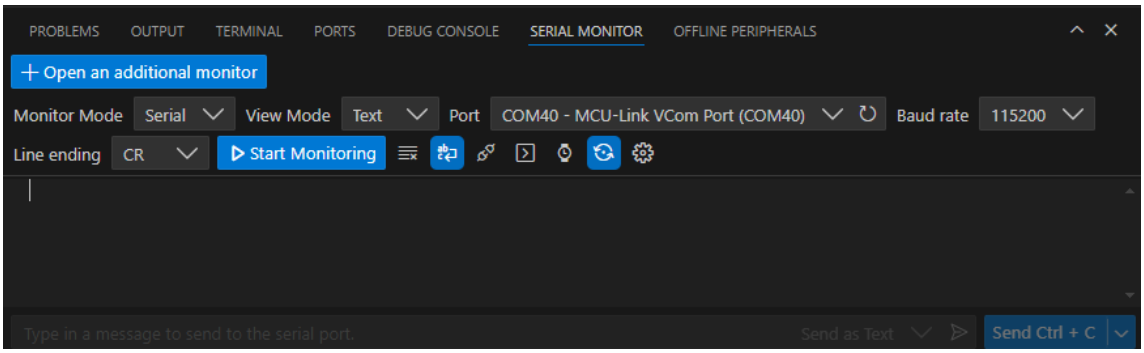


The integrated terminal will open at the bottom and will display the build output.

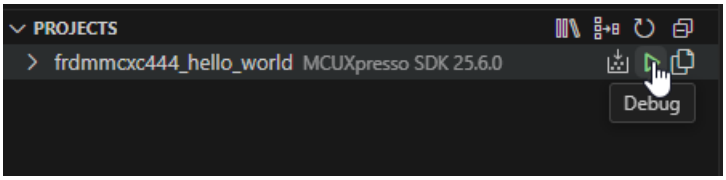


Run an example application **Note:** for full details on MCUXpresso for VS Code debug probe support, see MCUXpresso for VS Code Wiki.

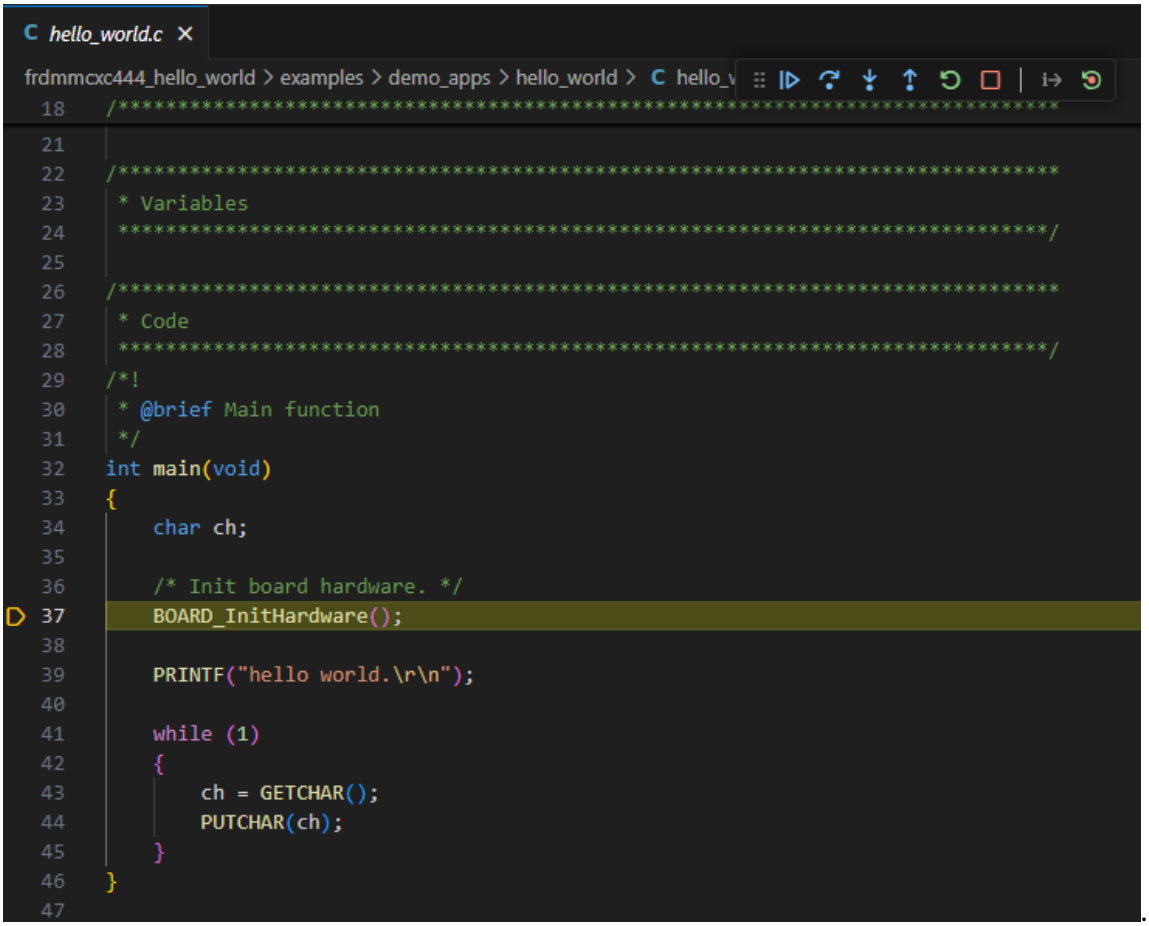
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



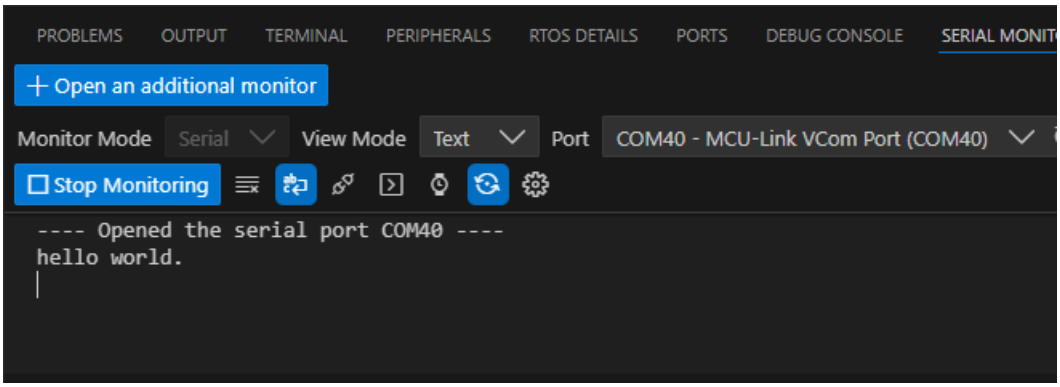
2. Navigate to the **PROJECTS** view and click the play button to initiate a debug session.



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

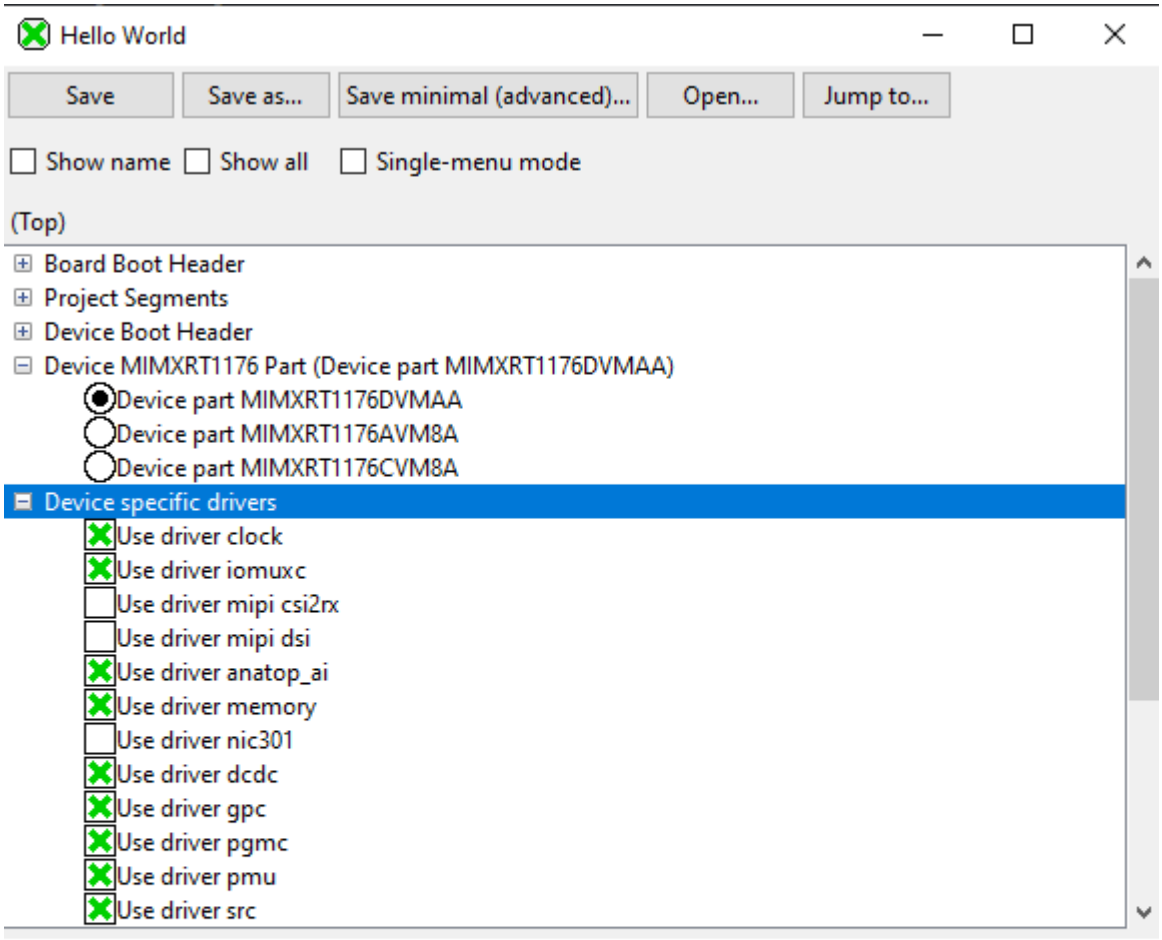
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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:

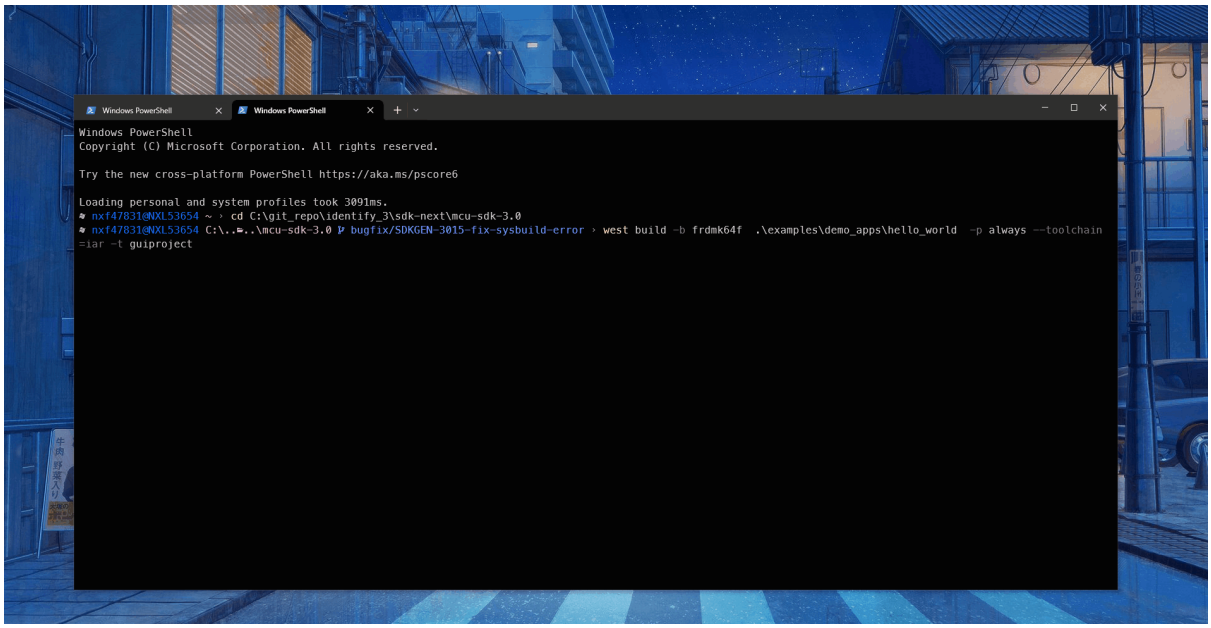
```
west debug -r linkserver
```

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 boards	MCU devices
MCX-N5XX-EVK	MCXN546VAB, MCXN546VDF, MCXN546VKL, MCXN546VNL, MCXN546VPB, MCXN547VAB, **MCXN547VDF**, MCXN547VKL, MCXN547VNL, MCXN547VPB

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.

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

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.

NXP Touch Library NXP Touch Library

TinyCBOR Concise Binary Object Representation (CBOR) Library

TF-M Trusted Firmware - M Library

PSA Test Suite Arm Platform Security Architecture Test Suite

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

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

NXP IoT Agent NXP IoT Agent

Multicore Multicore Software Development Kit

MCU Boot Open source MCU Bootloader.

mbedTLS mbedtls SSL/TLS library v3.x

Voice Seeker (no AEC) VoiceSeeker is a multi-microphone voice control audio front-end signal processing solution. VoiceSeeker is not featuring acoustic echo cancellation (AEC).

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

Audio Voice components Audio Voice components for MCU

Maestro Audio Framework for MCU Maestro Audio Framework library for MCU

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

LVGL LVGL Open Source Graphics Library

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.

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

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

NXP ELS PKC ELS PKC crypto library

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.

Only FreeRTOS is tested for RTOS support

This release only supports the FreeRTOS kernel and a bare-metal non-preemptive task scheduler.

Bluetooth LE

Most sensor applications have pairing and bonding disabled to allow a faster interaction with mobile applications. These two security features can be enabled in the app_preinclude.h header file.

Bluetooth LE controller:

- The maximum Advertising data length is limited to 800 bytes.
- Missing chained packets during scanning.
- Need to add a dummy nbu request to wake-up NBU core during asynchronous wakeup to avoid low power race condition issue (ie: for NBU ramlog dump from shared memory).

Periodic Advertising with Responses (PAwR):

- Periodic Advertising with Response (PAwR) is not supported with the configuration “Subevent Interval = Number of Response Slots x Response Slot Spacing with Response Slot Spacing = 0x2”.
- Periodic Advertising placement with connection events is unoptimized.

- The feature is not functional with the Free-Running Oscillator (FRO32K); it requires a 32 KHz Crystal Oscillator with accuracy less than 50 ppm.
- Periodic Advertising Response scheduling is unoptimized, which could result in missing a few responses.
- The PAwR event skipping feature is not correctly supported.

KW45/MCXW71: No specific issues.

KW47/MCXW72: Channel Sounding (CS): Limitations:

- RTT with Sounding Sequence is not supported.
- LE 2M 2BT PHY is not supported.
- Maximum 6 Channel Sounding procedures are supported in parallel.
- Scheduling of activities may be non optimal when multiple Channel Sounding procedures are running in parallel.
- Phase measurement bias is within certification range (<1.7x2πns) with KW47 EVK board. However, if different PCB or antenna matching is used, some bias may appear due to increased delay.
- Channel Sounding is not supported with internal FRO32. Known issues:
- When CS_SYNC=2Mbps, sensitivity drops at –85dBm (vs –95dBm for 1Mbps).
- NADM may report false positive attacks when using localization board at 2Mbps.
- When CS Subevents are configured very close from each other (<700us), some Subevents may be aborted with reason 0x3.
- When CS offset is configured too close from ACL anchor point, the anchor point may not be served (TX on central or RX on peripheral will not happen). Ideally, CS Offset should be configured greater than 1ms.
- Wireless_ranging stability issue in test mode 2Mbps with RTT random sequence payload 128 bits.
- RTT bias compensation:
  - For parts not properly configured at production (IFR blank), RTT bias may not be compensated properly. Consequently, an inaccuracy of +/-2m may be observed.
  - Temperature variation: RTT variation of +/-2m is observed based on temperature.

Zigbee

- OTA interruption is not resuming correctly (for example, when using a reset in the middle of the transfer).
- Zigbee ZED RX OFF example application on FreeRTOS fails sometime.
- Minor fixes and stability improvements for connectivity_test example application.

Flash ROMAPI

Note that:

- If using ROM API for internal flash or SPI NOR operation, reserve RAM location 0x200030A0 - 0x200032CF (0x300030A0 - 0x300032CF).
- If using kb API, reserve 0x20002000 - 0x200032FF (0x30002000 - 0x300032FF).

Other limitations

- The following Connectivity Framework configurations are Experimental and not recommended for mass production:
  - Power down on application power domain.
- A hardfault can be encountered when using fsl_component_mem_manager_light.c memory allocator and shutting down some unused RAM banks in low power. It is due to a wrong reinitialization of ECC RAM banks. To be sure not to reproduce the issue, gPlatformShutdownEccRamInLowPower should be set to 0.
- GenFSK Connectivity_test application is not operational with Low Power enabled.
- Serial manager is only supported on UART (not I2C nor SPI).
- The --no-warn-rwx-segments cannot been recognized on legacy MCUXpresso IDE versions.

  The --no-warn-rwx-segments option in MCUXpresso projects should be manually removed from the project settings if someone needs to use legacy (< 11.8.0) MCUXpresso IDE versions
- If the FRO32K is configured as the clock source of the CM33 Core then the debug session will block in both IAR, MCUX CMSIS-DAP while debugging. Use a lower debug wire speed, for example 1 MHz instead of the default one.

  In IAR, the option is in **Runtime Checking** -> **Debugger** -> **CMSIS DAP** -> **Interface** -> **Interface speed**.

  In MCUXpresso IDE, the option is in **LinkServer Debugger** -> **Advanced Settings** -> **Wire-speed (Hz)**.
- Low power reference design applications are not supported for the armgcc toolchain from zip archives. Please use MCUXpresso IDE or IAR toolchains for development using these applications.

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.

Example mbedtls_benchmark may hang on some targets on devices with ELS acceleration

Some targets of ELS accelerated devices may experience runtime issues when run with the default configuration of the mbedtls_benchmark application.

Examples: mbedtls_benchmark

Affected toolchains: All

The iped example does not complete successfully

The iped example fails at multiple points.

Examples: iped

Affected toolchains: All

TF-M secure and EL2GO examples incorrect path in “Download extra image” with iar and mdk IDEs with Kex package

TF-M secure and EL2GO examples are missing the target path for ns binary in “extra download image” with iar and mdk IDEs

Examples: tfm_demo_s, tfm_psatest_s, tfm_regression_s, tfm_secureboot_s, el2go_agent_s, el2go_blob_test_s, el2go_import_blob_s, el2go_mqtt_demo_s **Affected toolchains**: mdk, iar **Affected platforms**: mcxn5xxevk, frdmmcxn947, mcxn9xxevk, rdrw612bga, frdmrw612 **Workaround**: There are two ways 1.) Flash secure and non secure bins via Jlink or SPSDK after the build with IDE and providing with correct paths of secure and non-secure binaries. or 2.) Add {target} debug/release in path of “Download extra image” for iar and for MDK in xxx_flashdownload.ini file.

1.5 ChangeLog

1.5.1 MCUXpresso SDK Changelog

Board Support Files

board

[25.06.00]

- Initial version

clock_config

[25.06.00]

- Initial version

pin_mux

[25.06.00]

- Initial version

CACHE64

[2.0.12]

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

[2.0.11]

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

[2.0.10]

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

[2.0.9]

- Improvements
  - Removed assert(false) in CACHE64_GetInstanceByAddr.

[2.0.8]

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

[2.0.7]

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

[2.0.6]

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

[2.0.5]

- Improvement
  - Made use of FSL_FEATURE_CACHE64_CTRL_HAS_NO_WRITE_BUF feature

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

CACHE LPCAC

[2.2.1]

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

[2.2.0]

- Improvements
  - Used new add macros FSL_FEATURE_SYSCON_HAS_LPCAC_CTRL_PARITY_MISS_EN_BIT and FSL_FEATURE_SYSCON_HAS_LPCAC_CTRL_PARITY_FAULT_EN_BIT to support some platforms which PARITY_MISS_EN and PARITY_FAULT_EN bit may be reserved.

[2.1.1]

- Bug Fixes
  - Fixed an issue of L1CACHE_InvalidateCodeCache() function, to clean cache the LPCAC_CTRL[CLR_LPCAC] should be set not clear.

[2.1.0]

- Improvements
  - Supported more features, such as write buffer contron, write buffer limit and so on.

[2.0.0]

- Initial version.

CDOG

[2.1.3]

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

[2.1.2]

- Support multiple IRQs
- Fix default CONTROL values

[2.1.1]

- Remove bit CONTROL[CONTROL_CTRL].

[2.1.0]

- Rename CWT to CDOG.

[2.0.2]

- Fix MISRA-2012 issues.

[2.0.1]

- Fix doxygen issues.

[2.0.0]

- Initial version.

CLOCK

[2.0.1]

- Improvements
  - Added kCLOCK_None for the peripherals do not have clock gate.

[1.0.1]

- Improvements
  - Added Clock ip name array for SINC.

[1.0.0]

- initial version.

MCX_CMC

[2.5.0]

- New features
  - Update MCX_CMC driver to be compatible with new platforms.

[2.4.0]

- New features
  - Update MCX_CMC driver to be compatible with new platforms.

[2.3.0]

- Improvements
  - Added macros to support some device have BSR_SCR bit field.

[2.2.3]

- Improvements
  - Clear SCB SCR[SLEEPDEEP] bitfield after wakeup.

[2.2.2]

- Improvements
  - Fixed the violation of MISRA C-2012 rules.

[2.2.1]

- Improvements
  - Updated _cmc_system_reset_interrupt_enable, _cmc_system_reset_interrupt_flag and _cmc_system_reset_sources to support new added bit field.
- Bug Fixes
  - Fixed issue in CMC_PowerOffSRAMAllMode() and CMC_PowerOffSRAMLowPowerOnly() which overwrite reserved bit fields.

[2.2.0]

- Improvements
  - Added feature macro "FSL_FEATURE_MCX_CMC_HAS_NO_FLASHCR_WAKE" to support some devices where FLASHCR[WAKE] is reserved.

[2.1.0]

- Improvements
  - Added macros to support some devices(such as MCXA family) that only support one power domain.

[2.0.0]

- Initial version.

COMMON

[2.6.1]

- Improvements
  - Support Cortex M23.

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

- Improvements
  - Enable irqs that mount under irqsteer interrupt extender.

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.3]

- New Features
  - Added NETC into status group.

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

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

[2.3.0]

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

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Other Change
  - Added MECC status group definition.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.4]

- New Features
  - Added OTFAD into status group.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

CRC

[2.0.4]

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

[2.0.3]

- Bug fix:
  - Fix MISRA issues.

[2.0.2]

- Bug fix:
  - Fix MISRA issues.

[2.0.1]

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

[2.0.0]

- Initial version.

CTIMER

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

DAC

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EDMA

[2.10.7]

- 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.

EDMA_SOC

[1.0.0]

- Initial version.

EIM

[2.0.4]

- Improvements
  - Change eim_memory_channel_t to uint32_t.
  - Support channel number 16-23.

[2.0.3]

- Bug Fixes
  - Fixed s_eimResets variable declaration issue.

[2.0.2]

- Improvements
  - Reset the peripheral during initialization, otherwise some platforms may not work.
- Bug Fixes
  - Fixed incorrect register access in EIM_GetDataBitMask().

[2.0.1]

- Improvements
  - Update driver to support fewer channel.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3.

[2.0.0]

- Initial version.

MCX_ENET

[2.1.4]

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

[2.1.3]

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

[2.1.2]

- New features
  - Added hardware checksum acceleration support.

[2.1.1]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.

ERM

[2.0.4]

- Improvements
  - Change erm_memory_channel_t to uint32_t.
  - Support channel number 16-23.

[2.0.3]

- Bug Fixes
  - Fixed s_ermResets variable declaration issue.

[2.0.2]

- Improvements
  - Reset the peripheral during initialization, otherwise some platforms may not work.
- Bug Fixes
  - Only keep bit 0-3 of ERM_GetInterruptStatus() as it may get other channel results in higher bits.

[2.0.1]

- Improvements
  - Update driver to support fewer channel.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3.

[2.0.0]

- Initial version.

EVTG

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.7,15.7,10.1,10.4,10.8,10.3,16.1,16.3 .

[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_I2C

[2.6.2]

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

[2.6.1]

- Bug Fixes
  - Fixed coverity issues

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

- Bug Fixes
  - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
  - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
  - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
  - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
  - Fixed the Dozen mode configuration error in FLEXIO_I2C_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.

  - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking API called FLEXIO_I2C_MasterTransferCreateHandle, which lead to the s_flexioHandle/s_flexioIsr/s_flexioType variable being written. Then, if calling FLEXIO_I2C_MasterTransferBlocking API multiple times, the s_flexioHandle/s_flexioIsr/s_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S_EDMA

[2.1.9]

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

[2.1.8]

- Improvements
  - Applied EDMA ERRATA 51327.

FLEXIO_MCU_LCD

[2.3.0]

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

[2.2.0]

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

[2.1.0]

- New Features
  - Supported transmit only data without command.

[2.0.8]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

FLEXIO_MCU_LCD_EDMA

[2.0.6]

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

[2.0.5]

- New Features
  - Supported transmit only data without command.

[2.0.4]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

FLEXIO_MCU_LCD_SMARTDMA

[2.0.6]

- Other Changes
  - Add more MCXA devices support.

[2.0.5]

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

[2.0.4]

- New Features
  - Supported the platforms which use FlexIO SHIFTER DMA to trigger SmartDMA, such as MCXN235, MCXN236.

[2.0.3]

- New Features
  - Supported transmit only data without command.

[2.0.2]

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

[2.0.1]

- Other Changes
  - Update driver implementation due to SMARTDMA driver update.

[2.0.0]

- Initial version.

FLEXIO_SPI

[2.4.3]

- Improvements
  - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed coverity issues

[2.4.0]

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

[2.3.5]

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

[2.3.4]

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

[2.3.3]

- Bugfixes
  - Fixed cs-continuous mode.

[2.3.2]

- Improvements
  - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

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

[2.3.0]

- New Features
  - Supported FLEXIO_SPI slave transfer with continuous master CS signal and CPHA=0.
  - Supported FLEXIO_SPI master transfer with continuous CS signal.
  - Support 32 bit transfer width.
- Bug Fixes
  - Fixed wrong timer compare configuration for dma/edma transfer.
  - Fixed wrong byte order of rx data if transfer width is 16 bit, since the we use shifter buffer bit swapped/byte swapped register to read in received data, so the high byte should be read from the high bits of the register when MSB.

[2.2.1]

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

[2.2.0]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
  - Fixed bug where FLEXIO SPI transfer data is in 16 bit per frame mode with eDMA.
  - Fixed bug when FLEXIO SPI works in eDMA and interrupt mode with 16-bit per frame and Lsbfirst.
  - Fixed the Dozen mode configuration error in FLEXIO_SPI_MasterInit/FLEXIO_SPI_SlaveInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
- Improvements
  - Added #ifndef/#endif to allow users to change the default TX value at compile time.

[2.1.0]

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

FLEXIO_UART

[2.6.4]

- Improvements
  - Make UART_RETRY_TIMES configurable by CONFIG_UART_RETRY_TIMES.

[2.6.3]

- Bug Fixes
  - Fixed coverity issues

[2.6.2]

- Bug Fixes
  - Fixed coverity issues

[2.6.1]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

- Improvements
  - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_UART_EDMA

[2.3.1]

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

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0

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.

FREQME

[2.1.2]

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

[2.1.1]

- Fixed MISRA issues.

[2.1.0]

- Updated register name.

[2.0.0]

- Initial version.

GDET

[2.1.0]

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

[2.0.1]

- Fix MISRA in GDET_ReconfigureVoltageMode().

[2.0.0]

- Initial version.

GPIO

[2.8.2]

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

[2.8.1]

- Bug Fixes
  - Fixed CERT INT31-C issues.

[2.8.0]

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

[2.8.0]

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

[2.7.3]

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

[2.7.2]

- New Features
  - Support devices without PORT module.

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
  - Removed deprecated APIs.

[2.3.0]

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

[2.2.1]

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

[2.1.1]

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

[2.1.0]

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

I3C

[2.14.3]

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

[2.14.2]

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

[2.14.1]

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

[2.14.0]

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

[2.13.1]

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

[2.13.0]

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

[2.12.0]

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

[2.11.1]

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

[2.11.0]

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

[2.10.8]

- Improvements
  - Support more instances.

[2.10.7]

- Improvements
  - Fixed the potential compile warning.

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- New features
  - Added I3C extended IBI data support.

[2.9.0]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.4]

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

[2.5.3]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH and FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

I3C_EDMA

[2.2.12]

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

[2.2.11]

- Improvements
  - Fixed Coverity CERT-C violations.

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Bug Fixes
  - Fixed MISRA issue rule 11.6.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
  - Deleted legacy IBI data request code.

[2.1.0]

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

[2.0.1]

- Bug Fixes
  - Fixed MISRA issue rule 9.1.

[2.0.0]

- Initial version.

INPUTMUX

[2.0.10]

- Bug Fixes
  - Fixed CERT-C violations.

[2.0.9]

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

[2.0.8]

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

[2.0.7]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Support channel mux setting in INPUTMUX_EnableSignal().

[2.0.0]

- Initial version.

INTM

[2.1.0]

- Replace macro FSL_FEATURE_INTM_MONITOR_COUNT to INTM_MON_COUNT.

[2.0.1]

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

[2.0.0]

- Initial version.

IPED

[2.2.0]

- Renamed CSS to ELS.

[2.1.1]

- Fix build error due to renamed symbols.

[2.1.0]

- Add IPED_Config() (including FFR CMPA write) and IPED_Reconfig() features.

[2.0.0]

- Initial version.

IRTC

[2.3.3]

- Bug Fixes
  - Fix CERT INT31-C issue.

[2.3.2]

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

[2.3.1]

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

[2.3.0]

- New Feature
  - Supported platforms with multiple IRTC instances.

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Bug Fixes
  - Fixed MISRA issues.

[2.2.0]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

ITRC

[2.4.0]

- Rework the input signal definition for better flexibility

[2.3.0]

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

[2.2.0]

- Update driver to new version and input events

[2.1.0]

- Make SYSCON glitch platform dependent.

[2.0.0]

- Initial version.

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.

LPCMP

[2.3.2]

- Improvements
  - Fixed LPCMP CERT-C issues.

[2.3.1]

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

[2.3.0]

- New Feature
  - Added some new features for platforms which support
    - Plus input source selection.
    - Minus input source selection.
    - CMP to DAC link.
- Improvements
  - Removed some new features for platforms which doesn't support
    - Functional clock source selection.
    - DAC high power mode selection.

    * Round Robin clock source selection.
    * Round Robin trigger source selection.
    * Round Robin channel sample numbers setting.
    * Round Robin channel sample time threshold setting.
    * Round Robin internal trigger configuration.

[2.2.0]

- Improvements
  - Change FSL_FEATURE_LPCMP_HAS_NO_CCR0_CMP_STOP_EN to FSL_FEATURE_LPCMP_HAS_CCR0_CMP_STOP_EN.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPFLEXCOMM

[2.2.1]

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

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

LPI2C

[2.2.6]

- Bug Fixes
  - Fixed coverity issues.

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

- Improvements
  - Fixed doxygen warning in LPI2C_SlaveTransferHandleIRQ.

[2.2.1]

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

[2.2.0]

- Improvements
  - Support the normal LPI2C in LPFLEXCOMM driver.

[2.1.1]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPI2C_EDMA

[2.0.1]

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

[2.0.0]

- Initial version.

LPSPI

[2.2.9]

- Bug Fixes
  - Fixed coverity issues.

[2.2.8]

- Bug Fixes
  - Fixed coverity issues.

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

- Improvements
  - Fixed doxygen warning in LPSPI_SlaveTransferHandleIRQ.

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

- Fixed LPSPI_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

[2.0.0]

- Initial version.

LPSPI_EDMA

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

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

[2.3.4]

- Bug Fixes
  - Fixed coverity issues.

[2.3.3]

- Bug Fixes
  - Fixed coverity issues.

[2.3.2]

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

[2.3.1]

- Bug Fix
  - Fixed MISRA C-2012 violations.

[2.3.0]

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

[2.2.4]

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

[2.2.3]

- Improvements
  - Added atomic in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.2.2]

- Improvements
  - Added comment on txExtendedTimeoutValue of lpuart_timeout_config_t.

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

MAILBOX

[2.3.4]

- Improvements
  - Remove support for NXH2004 series

[2.3.3]

- Improvements
  - Added support for the MCXN556S, MCXN537, MCXN536, MCXN527 and MCXN526 series

[2.3.2]

- Improvements
  - Added support for the MCXN946 and MCXN546 series

[2.3.1]

- Improvements
  - Added support for the LPC55S66 series.

[2.3.0]

- Improvements
  - Added support for the MCXNx4x series with new value for kMAILBOX_CM33_Core0 or kMAILBOX_CM33_Core1.

[2.2.0]

- Improvements
  - Fixed missing conditional defines for the LPC5411x series.

[2.1.0]

- Improvements
  - Added support for the LPC55S69 series. cpu_id parameter can be newly assigned to kMAILBOX_CM33_Core0 or kMAILBOX_CM33_Core1.

[2.0.0]

- Initial version.

MRT

[2.0.5]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

NPX

[2.0.0]

- Initial version.

OSTIMER

[2.2.6]

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

[2.2.5]

- Improvements
  - Support binary encoded ostimer.

[2.2.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.2.3]

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

[2.2.2]

- Improvements
  - Support devices with different OSTIMER instance name.

[2.2.1]

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

[2.2.0]

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

[2.1.2]

- Bug Fixes
  - Fixed MISRA-2012 rule 10.8.

[2.1.1]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

- Improvements
  - Added support for OSTIMER0

[2.0.1]

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

[2.0.0]

- Initial version.

OTP

[2.0.1]

- Bug Fixes
  - Fixed MISRA-C 2012 violations.

[2.0.0]

- Initial version.

PDM

[2.9.3]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
  - Added new features FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ/FSL_FEATURE_PDM_HAS_DC_OUT

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - Added software buffer queue for transactional API.

[2.0.1]

- Improvements
  - Improved HWVAD feature.

[2.0.0]

- Initial version.

PDM_EDMA

[2.6.5]

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

[2.6.4]

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

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.5.0

PINT

[2.2.0]

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

[2.1.13]

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

[2.1.12]

- Bug Fixes
  - Fixed coverity issue.

[2.1.11]

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

[2.1.10]

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

[2.1.9]

- Bug Fixes
  - Fixed MISRA-2012 rule 8.4.

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

PLU

[2.2.1]

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

[2.2.0]

- Bug Fixes
  - Fixed wrong parameter of the PLU_EnableWakeIntRequest function.

[2.1.0]

- New Features
  - Added 4 new APIs to support Niobe4's wake-up/interrupt control feature, including PLU_GetDefaultWakeIntConfig() PLU_EnableWakeIntRequest(), PLU_LatchInterrupt() and PLU_ClearLatchedInterrupt().
- Other Changes
  - Changed the register name LUT_INP to LUT_INP_MUX due to register map update.

[2.0.1]

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

[2.0.0]

- Initial version.

PORT

[2.5.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.5.0]

- Bug Fixes
  - Correct the kPORT_MuxAsGpio for some platforms.

[2.4.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules: 10.1, 10.8 and 14.4.

[2.4.0]

- New Features
  - Updated port_pin_config_t to support input buffer and input invert.

[2.3.0]

- New Features
  - Added new APIs for Electrical Fast Transient(EFT) detect.
  - Added new API to configure port voltage range.

[2.2.0]

- New Features
  - Added new api PORT_EnablePinDoubleDriveStrength.

[2.1.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules: 10.1, 10.4□11.3□11.8, 14.4.

[2.1.0]

- New Features
  - Updated the driver code to adapt the case of the interrupt configurations in GPIO module. Will move the pin configuration APIs to GPIO module.

[2.0.2]

- Other Changes
  - Added feature guard macros in the driver.

[2.0.1]

- Other Changes
  - Added “const” in function parameter.
  - Updated some enumeration variables’ names.

POWERQUAD

[2.2.1]

- Bug Fixes
  - Fixed CERT-C issues.

[2.2.0]

- New Features
  - Added new API PQ_Arctan2Fixed.

[2.1.1]

- Bug Fixes
  - Remove PQ_WaitDone from PQ_ArctanFixed and PQ_ArctanhFixed because it is unnecessary.

[2.1.0]

- Improvements
  - Fixed typo issue for biquad related function name.
  - Changed operator from “%” into “&” to reduce heavy cycle for biquad functions.

[2.0.5]

- Improvements
  - Added a note in driver for FIR that powerquad has a hardware limitation, when using it for FIR increment calculation, the address of pSrc needs to be a continuous address.

[2.0.4]

- Improvements
  - Supported the platforms which don’t have PowerQuad clock and reset control.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.3, 10.4, 10.6, and so on.

[2.0.2]

- Bug Fixes
  - Fixed array size issue in fsl_powerquad_data.h file.
  - Fixed vector function pipeline issue.

[2.0.1]

- Bug Fixes
  - Fixed build error in C++ mode.

[2.0.0]

- Initial version.

PUFv3

[2.0.4]

- Fix warning.

[2.0.3]

- Update for various PUF CTRL wrapper.

[2.0.2]

- Fix MISRA issue in driver.

[2.0.1]

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

[2.0.0]

- Initial version.

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.

QDC

[2.0.0]

- Initial version. Rename from driver ENC.

RESET

[2.4.1]

- Improvements
  - Add kRST_None for the peripherals do not have reset gate.

[2.4.0]

- Improvements
  - Add RESET_ReleasePeripheralReset API.

[1.0.0]

- Initial version.

SAI

[2.4.10]

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

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

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

[2.4.7]

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

[2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

[2.4.5]

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

[2.4.4]

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

[2.4.3]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
  - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

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

[2.3.6]

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

[2.3.5]

- Improvements
  - Make driver to be aarch64 compatible.

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
  - Added new APIs for parameters collection and simplified user interfaces:
    - SAI_Init
    - SAI_SetMasterClockConfig
    - SAI_TxSetBitClockRate
    - SAI_TxSetSerialDataConfig
    - SAI_TxSetFrameSyncConfig
    - SAI_TxSetFifoConfig
    - SAI_TxSetBitclockConfig
    - SAI_TxSetConfig
    - SAI_TxSetTransferConfig
    - SAI_RxSetBitClockRate
    - SAI_RxSetSerialDataConfig
    - SAI_RxSetFrameSyncConfig
    - SAI_RxSetFifoConfig

    * SAI_RxSetBitclockConfig
    * SAI_RXSetConfig
    * SAI_RxSetTransferConfig
    * SAI_GetClassicI2SConfig
    * SAI_GetLeftJustifiedConfig
    * SAI_GetRightJustifiedConfig
    * SAI_GetTDMConfig

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - API name changes:
    - SAI_GetSendRemainingBytes -> SAI_GetSentCount.
    - SAI_GetReceiveRemainingBytes -> SAI_GetReceivedCount.
    - All names of transactional APIs were added with “Transfer” prefix.
    - All transactional APIs use base and handle as input parameter.
    - Unified the parameter names.

- Bug Fixes
  - Fixed WLC bug while reading TCSR/RCSR registers.
  - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI_TxInit/SAI_RxInit.

[2.0.0]

- Initial version.

SAI_EDMA

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
  - Removed deprecated APIs.

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2

SCTIMER

[2.5.1]

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

[2.5.0]

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

[2.4.9]

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

[2.4.8]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

- Bug Fixes
  - Fix SCT_EV_STATE_STATEMSKn macro build error.

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

[2.3.0]

- Bug Fixes
  - Fixed the potential overflow issue of pulseperiod variable in SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle API.
  - Fixed the issue of SCTIMER_CreateAndScheduleEvent API does not correctly work with 32 bit unified counter.
  - Fixed the issue of position of clear counter operation in SCTIMER_Init API.

- Improvements
  - Update SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle to support generate 0% and 100% PWM signal.
  - Add SCTIMER_SetupEventActiveDirection API to configure event activity direction.
  - Update SCTIMER_StartTimer/SCTIMER_StopTimer API to support start/stop low counter and high counter at the same time.
  - Add SCTIMER_SetCounterState/SCTIMER_GetCounterState API to write/read counter current state value.
  - Update APIs to make it meaningful.
    - SCTIMER_SetEventInState
    - SCTIMER_ClearEventInState
    - SCTIMER_GetEventInState

[2.2.0]

- Improvements
  - Updated for 16-bit register access.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.

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.

SMARTCARD

[2.3.0]

- New features:
  - Added support for USIM

[2.2.2]

- Bug fix:
  - Fixed MISRA C-2012 rule 10.4.

[2.2.1]

- Bug fix:
  - Fixed IAR warnings Pa082 in smartcard_emvsim
  - Fixed MISRA issues
  - Fixed rules 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 14.4, 16.1, 16.3, 16.4, 17.7

[2.2.0]

- New features:
  - Updated to use RX/TX FIFO

[2.1.2]

- Provided time delay function which works in microseconds.
- Bug fix:
  - Changed event to semaphore in RTOS driver (KPSDK-11634).
  - Added check if de-initialized variables are not null iSMARTCARD_RTOS_Deinit() (KPSDK-8788).
  - Changed deactivation sequence iSMARTCARD_PHY_TDA8035_Deactivate() to properly stop the clockPOSCR-35).
  - Fixed timing issue with VSEL0/1 signals in smartcard TDA803driver (KPSDK-10160)

[2.1.1]

- New features:
  - Added default phy interface selection into smartcard RTOS drivers (KPSDK-9063).
  - Replaced smartcard_phy_ncn8025 driver by smartcard_phy_tda8035.
- Bug fix:
  - Fixed protocol timers activation sequences in smartcard_emvsim and smartcard_phy_tda8035 drivers during emvl1 pre-certification tests (KPSDK-9170, KPSDK-9556).

[2.1.0]

- Initial version.

SMARTDMA

[2.14.0]

- Improvements
  - Refine the device specific firmware selection method, use device Kconfig symbol to select.

[2.13.3]

- Bug Fixes
  - Fix RT500 FlexIO QSPI MDK build warning.

[2.13.2]

- Other Changes
  - Add more MCXN devices support.

[2.13.1]

- Other Changes
  - Add more MCXA devices support.

[2.13.0]

- New Features
  - Added MCXA keyscan firmware.
  - Added functions to control CTRL[EXF].

[2.12.0]

- New Features
  - Supported MCXA mculcd and camera functions.

[2.11.0]

- Improvements
  - Make the RT500 QSPI firmware can work with other display firmware in the same project.

[2.10.0]

- New Features
  - Added new camera APIs for MCXN SoCs to support more resolutions.

[2.9.1]

- New Features
  - Supported MCXN235, MCXN236.

[2.9.0]

- New Features
  - Supported MCXN camera functions.
  - Supported user to select individual firmware for MIPI or FLEXIO alone, or both.
  - Added new API of enabling DMA from FlexIO to a Buffer.
  - Added new APIs of setting MIPI-DSI to enter and exit ultra low power state.

[2.8.0]

- New Features
  - Supported converting the pixel data from RGB565 to RGB888.
  - Supported function to turn off certain pixel in a checker board pattern.

[2.7.0]

- New Features
  - Supported data transfer in 2-dimensional way.
  - Supported data transfer in XRGB8888 format and rotate 180 degree.
  - Supported to fill data in whenever there is room in MIPI controller's FIFO rather than using the tx FIFO in double buffered way.

[2.6.3]

- Bug Fixes
  - Fixed EZH_MIPIDSI_RGB565_DMA, EZH_MIPIDSI_RGB888_DMA, EZH_MIPIDSI_ARGB8888toRGB888_DMA issues that don't support some length.

[2.6.2]

- Bug Fixes
  - Fixed MISRA C-2012 issues: 8.4, 11.6, 17.7.

[2.6.1]

- Improvements
  - Optimized MIPI DSI APIs performance.

[2.6.0]

- Improvements
  - Optimized MIPI DSI APIs performance.
- New Features
  - Added new APIs to send MIPI DSI frame with 180 degree rotation.

[2.5.0]

- Improvements
  - Supported swap or don't swap the pixel byte before written to MIPI DSI FIFO.
  - Updated MIPI DSI firmware, make sure data has been sent out before calling callback function.

[2.4.0]

- Improvements
  - Added new APIs for MIPI DSI kSMARTDMA_MIPI_XRGB2RGB_DMA.

[2.3.0]

- Improvements
  - Added new APIs for FlexIO one SHIFTBUF, kSMARTDMA_FlexIO_DMA_ONELANE.
- Bug Fixes
  - Fixed kSMARTDMA_MIPI_RGB565_DMA color bias issue.

[2.2.0]

- Improvements
  - Added new APIs for MIPI DSI, kSMARTDMA_MIPI_RGB565_DMA and kSMARTDMA_MIPI_RGB888_DMA.
  - Supported install firmware and callback function dynamically.

[2.1.0]

- Improvements
  - Removed test APIs, including kSMARTDMA_LightOn, kSMARTDMA_LightOff, kSMARTDMA_Notify, and kSMARTDMA_Test.
  - Added new APIs, including kSMARTDMA_FlexIO_DMA_Reverse, kSMARTDMA_FlexIO_DMA_ARGB2RGB, kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap, and kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap_Reverse.

[2.0.0]

- Initial version.

MCX_SPC

[2.10.0]

- New Features
  - Add feature macro to be compatible with some platforms where SPC does not support:
    1. Has no SC register SPC_LP_REQ bitfield.
    2. Has no SC register SPC_LP_MODE bitfield.
    3. Has no PD_STATUS register.
    4. Has no SRAMCTL register.

[2.9.1]

- Improvements
  - There is no need to ensure that the bandgap is enabled before setting the active mode core LDO regulator drive strength and voltage level, because the hardware will automatically turn on the bandgap function if it is required by the hardware when configuring the LDO.

[2.9.0]

- New Features
  - Add feature macro to be compatible with some platforms where SPC does not support overdrive voltage.

[2.8.1]

- Improvements
  - Fixed cert_int31_c_violation of SPC_GetVoltageDetectStatusFlag(), SPC_GetExternalDomainsStatus() and SPC_SetDCDCBurstConfig.

[2.8.0]

- New Features
  - Updated MCX_SPC driver for compatibility with SoCs without PD_STATUS[PWR_REQ_STATUS].

[2.7.0]

- New Features
  - Added new function to unretain RAM array.

[2.6.0]

- Bug Fixes
  - The enumeration kSPC_DeepPowerDownWithSysClockOff should be 0x8U.
- New Features
  - Added functions to get the last low-power mode that the power domain requested.

[2.5.0]

- Improvements
  - Updated SPC_SetLowPowerModeCoreLDORegulatorConfig() with adding check that LP_CFG[CORELDO_VDD_LVL] only be updated when CORELDO is in normal driver strength in active mode.
  - Updated SPC_SetLowPowerModeRegulatorsConfig() with adding check the if DCDC voltage level set as retention mode, CORELDO low voltage detection must be disabled.
  - Updated spc_analog_module_control with adding Opamp3 support.
- New Features
  - Added functions to mask/unmask voltage detections in active mode.

[2.4.2]

- Improvements
  - Fixed the violation of MISRA C-2012 rules.

[2.4.1]

- Improvements
  - Fixed the violation of MISRA C-2012 rules.

[2.4.0]

- Improvements
  - Refined APIs to set regulators, the input parameters will be check firstly before setting register.
  - Improved APIs' document.
  - Removed software check for DCDC's settings since there is not hardware restrictions for DCDC.
  - Added new APIs to check if glitch detector is enabled in active/low power mode.
  - Added new APIs for DCDC burst feature, make it more flexible to use.
  - Added new API to enable/disable DCDC bleed resistor.
  - Set functions of VD_SYS_CFG[LVSEL] configuration as deprecated, since this bit filed is reserved for all devices.

  - Updated details of spc_core_ldo_voltage_level_t to align with description of latest RM, added comments to reminder users that the retention voltage is reserved in some devices.

[2.3.0]

- New Features
  - Updated glitch detect API to support devices which do not have aGDET

[2.2.1]

- Bug Fixes
  - Fixed an issue of SPC_SetActiveModeRegulatorsConfig() which will cause LVD in case of setting DCDC and CORE LDO into higher voltage level.

[2.2.0]

- New Features
  - Added some macros to support some devices(such as MCXA family) do not equipped DCDC, SYS_LDO and so on.
  - Added new function SPC_EnableSRAMLdOLowPowerModeIREF(), SPC_TrimSRAMLdoRefVoltage(), SPC_EnableSRAMLdo() to support some devices(such as MCXA family) that support control of SRAM_LDO.
  - Fixed violation of MISRA C-2012 rule 17.7.
  - Fixed an issue in SPC_SetLowPowerModeRegulatorsConfig() function that set ACTIVE_CFG register by mistake.

[2.1.0]

- Improvements
  - Added new functions to set regulators' voltage level and drive strength individually.
  - Updated SPC_SetActiveModeRegulatorsConfig() and SPC_SetLowPowerModeRegulatorsConfig() based on new added functions.

[2.0.1]

- Bug Fixes
  - Fixed a bug that external burst not working properly.
  - Fixed a bug that is SPC has VDD_DS the voltage is not configured correctly.

[2.0.0]

- Initial version.

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.

TDET

[2.3.0]

- Added enum for TIF10.

[2.2.0]

- Added support for chips without active tamper pins.

[2.1.1]

- Added clearing SR_TAF and SR_DTF into TDET_Init().
  - Fix typo in kTDET_ClockType64Hz comment.

[2.1.0]

- Added setting of disabling prescaler on tamper event into TDET_SetConfig() and TDET_GetDefaultConfig functions.

[2.0.0]

- Initial version.

TRDC

[2.2.2]

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

[2.2.1]

- Bug Fixes:
  - Fix MISRA violations.

[2.2.0]

- New Features:
  - Supported SoCs that do not have all TRDC modules.

[2.1.0]

- Bug Fixes:
  - Fix MISRA violations.
  - Fixed wrong operation of domain mask in TRDC_MbcNseClearAll and TRDC_MrcDomainNseClear.

[2.0.0]

- Initial version.

TRDC_SOC

[1.0.0]

- Initial version.

TSI_V6

[2.0.1]

- The tsi_shield_t typedef has been extended to cover 4 bits shielding bit-field.

[2.0.0]

- Initial version.

USDHC

[2.8.6]

- Bug Fixes
  - Invalidate cache after blocking read.

[2.8.5]

- Improvements
  - Enable the driver to be AARCH64 compatible.

[2.8.4]

- Improvements
  - Add feature macro FSL_FEATURE_USDHC_HAS_NO_VS18.

[2.8.3]

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

[2.8.2]

- Improvements
  - Added feature macro FSL_FEATURE_USDHC_HAS_NO_VOLTAGE_SELECT.

[2.8.1]

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

[2.8.0]

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

[2.7.0]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.8]

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

[2.2.7]

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

[2.2.6]

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

[2.2.5]

- Bug Fixes
  - Fixed MDK 66-D warning.

[2.2.4]

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

[2.2.3]

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

  - Dynamic set SDCLKFS during DDR enable or disable.

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
  - Improved USDHC to support MMC boot feature.

[2.1.3]

- Bug Fixes
  - Fixed MISRA issue.

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

UTICK

[2.0.5]

- Improvements
  - Improved for SOC RW610.

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

MCX_VBAT

[2.5.0]

- New features
  - Update MCX_VBAT driver to be compatible with new platforms.

[2.4.0]

- New features
  - Update MCX_VBAT driver to be compatible with new platforms.

[2.3.1]

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

[2.3.0]

- Improvements
  - OSCCTLA[FINE_AMP_GAIN] is reserved, added new macro to support this change.
  - Defined VBAT_LDORAMC_RET in case of this macro is not defined in header file.
- Bug Fixes
  - Fixed violaltions of MISRA C-2012 rule 10.8.

[2.2.0]

- Improvements
  - Added macros to support some devices(such as MCXA family) that do not support OSC control, LDO control, bandgap timer, and tamper.

[2.1.0]

- New features
  - Added functions to support tamper and clock monitor.

[2.0.0]

- Initial version.

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.

WUU

[2.4.1]

- Improvements
  - The semantics of kWUU_FilterActiveDSPD and kWUU_ExternalPinActiveDSPD are not clear enough, because on some platforms it's not 'DSPD' (deep sleep, power down) but PDDPD (power down, deep power down). We deprecate them and will use the more generic kWUU_FilterActiveLowLeakage and kWUU_ExternalPinActiveLowLeakage in the future.

[2.4.0]

- New Features
  - Added WUU_ClearExternalWakeupPinsConfig() to clear settings of PDC and PE register.

[2.3.0]

- New Features
  - Added WUU_ClearInternalWakeUpModulesConfig() to clear settings of DM and ME register.

[2.2.1]

- Bug Fixes
  - Fixed WUU_SetPinFilterConfig() unable to set edge detection of pin filter config.
  - Fixed wrong macro used in WUU_GetPinFilterFlag() function.

[2.2.0]

- New Features
  - Added the WUU_GetExternalWakeupPinFlag() and WUU_ClearExternalWakeupPinFlag() function .

[2.1.0]

- New Features
  - Added the WUU_GetModuleInterruptFlag() function to support the devices that equipped MF register.

[2.0.0]

- Initial version.

WWDT

[2.1.10]

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

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.

MCXN547_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 Audio Voice components

components

1.7.4 Maestro Audio Framework for MCU

Maestro Audio Framework

1.7.5 eIQ

eIQ

1.7.6 FreeMASTER

freemaster

1.7.7 AWS IoT

aws_iot

1.7.8 NXP Wi-Fi

wifi-bluetooth-802.15.4

1.7.9 FreeRTOS

FreeRTOS

1.7.10 Wireless EdgeFast Bluetooth PAL

EdgeFast Bluetooth

1.7.11 lwIP

lwip

1.7.12 File systemFatfs

fatfs

Chapter 2

MCXN947

2.1 CACHE: CACHE Memory Controller

uint32_t CACHE64_GetInstance(CACHE64_POLSEL_Type *base)

Returns an instance number given peripheral base address.

Parameters

- base – The peripheral base address.

Returns

CACHE64_POLSEL instance number starting from 0.

uint32_t CACHE64_GetInstanceByAddr(uint32_t address)

brief Returns an instance number given physical memory address.

param address The physical memory address.

Returns

CACHE64_CTRL instance number starting from 0.

status_t CACHE64_Init(CACHE64_POLSEL_Type *base, const *cache64_config_t* *config)

Initializes an CACHE64 instance with the user configuration structure.

This function configures the CACHE64 module with user-defined settings. Call the CACHE64_GetDefaultConfig() function to configure the configuration structure and get the default configuration.

Parameters

- base – CACHE64_POLSEL peripheral base address.
- config – Pointer to a user-defined configuration structure.

Return values

kStatus_Success – CACHE64 initialize succeed

void CACHE64_GetDefaultConfig(*cache64_config_t* *config)

Gets the default configuration structure.

This function initializes the CACHE64 configuration structure to a default value. The default values are first region covers whole cacheable area, and policy set to write back.

Parameters

- config – Pointer to a configuration structure.

void CACHE64_EnableCache(CACHE64_CTRL_Type *base)

Enables the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_DisableCache(CACHE64_CTRL_Type *base)

Disables the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_InvalidateCache(CACHE64_CTRL_Type *base)

Invalidates the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_InvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cache by range.

Note: Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be invalidated, should be larger than 0.

void CACHE64_CleanCache(CACHE64_CTRL_Type *base)

Cleans the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cache by range.

Note: Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be cleaned, should be larger than 0.

void CACHE64_CleanInvalidateCache(CACHE64_CTRL_Type *base)

Cleans and invalidates the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate cache by range.

Note: Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

void CACHE64_EnableWriteBuffer(CACHE64_CTRL_Type *base, bool enable)

Enables/disables the write buffer.

Parameters

- base – CACHE64_CTRL peripheral base address.
- enable – The enable or disable flag. true - enable the write buffer. false - disable the write buffer.

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0.

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0.

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned, should be larger than 0.

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

FSL_CACHE_DRIVER_VERSION

cache driver version.

enum _cache64_policy

Level 2 cache controller way size.

Values:

enumerator kCACHE64_PolicyNonCacheable

Non-cacheable

enumerator kCACHE64_PolicyWriteThrough

Write through

enumerator kCACHE64_PolicyWriteBack

Write back

typedef enum *_cache64_policy* cache64_policy_t

Level 2 cache controller way size.

typedef struct *_cache64_config* cache64_config_t

CACHE64 configuration structure.

CACHE64_LINESIZE_BYTE

cache line size.

CACHE64_REGION_NUM

cache region number.

CACHE64_REGION_ALIGNMENT

cache region alignment.

struct _cache64_config

#include <fsl_cache.h> CACHE64 configuration structure.

Public Members

uint32_t boundaryAddr[(3U) - 1]

< The cache controller can divide whole memory into 3 regions. Boundary address is the FlexSPI internal address (start from 0) instead of system address (start from FlexSPI AMBA base) to split adjacent regions and must be 1KB aligned. The boundary address itself locates in upper region. Cacheable policy for each region.

2.2 CACHE: LPCAC CACHE Memory Controller

static inline void L1CACHE_EnableCodeCache(void)

Enables the processor code bus cache.

static inline void L1CACHE_DisableCodeCache(void)

Disables the processor code bus cache.

static inline void L1CACHE_InvalidateCodeCache(void)

Clears cache.

static inline void L1CACHE_EnableAllocation(void)

Enables allocation.

static inline void L1CACHE_DisableAllocation(void)

Disables allocation.

static inline void L1CACHE_EnableParity(void)

Enables parity.

static inline void L1CACHE_DisableParity(void)

Disable parity.

FSL_CACHE_LPCAC_DRIVER_VERSION

cache driver version

2.3 CDOG

status_t CDOG_Init(CDOG_Type *base, *cdog_config_t* *conf)

Initialize CDOG.

This function initializes CDOG block and setting.

Parameters

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

Returns

Status of the init operation

void CDOG_Deinit(CDOG_Type *base)

Deinitialize CDOG.

This function deinitializes CDOG secure counter.

Parameters

- base – CDOG peripheral base address

void CDOG_GetDefaultConfig(*cdog_config_t* *conf)

Sets the default configuration of CDOG.

This function initialize CDOG config structure to default values.

Parameters

- conf – CDOG configuration structure

void CDOG_Stop(CDOG_Type *base, uint32_t stop)

Stops secure counter and instruction timer.

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

Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter

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

Sets secure counter and instruction timer values.

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

Parameters

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

void CDOG_Check(CDOG_Type *base, uint32_t check)

Checks secure counter.

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

Parameters

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

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

Sets secure counter and instruction timer values.

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

Parameters

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

void CDOG_Add(CDOG_Type *base, uint32_t add)

Add value to secure counter.

This function add specified value to secure counter.

Parameters

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

void CDOG_Add1(CDOG_Type *base)

Add 1 to secure counter.

This function add 1 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add16(CDOG_Type *base)

Add 16 to secure counter.

This function add 16 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add256(CDOG_Type *base)

Add 256 to secure counter.

This function add 256 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub(CDOG_Type *base, uint32_t sub)

brief Substract value to secure counter

This function substract specified value to secure counter.

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

void CDOG_Sub1(CDOG_Type *base)

Substract 1 from secure counter.

This function substract specified 1 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub16(CDOG_Type *base)

Substract 16 from secure counter.

This function substract specified 16 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub256(CDOG_Type *base)

Substract 256 from secure counter.

This function substract specified 256 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_WritePersistent(CDOG_Type *base, uint32_t value)

Set the CDOG persistent word.

Parameters

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

uint32_t CDOG_ReadPersistent(CDOG_Type *base)

Get the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.

Returns

The persistent word.

FSL_CDOG_DRIVER_VERSION

Defines CDOG driver version 2.1.3.

Change log:

- Version 2.1.3
  - Re-design multiple instance IRQs and Clocks
  - Add fix for RESTART command errata
- Version 2.1.2
  - Support multiple IRQs
  - Fix default CONTROL values
- Version 2.1.1
  - Remove bit CONTROL[CONTROL_CTRL]
- Version 2.1.0
  - Rename CWT to CDOG
- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - Fix doxygen issues
- Version 2.0.0
  - initial version

enum ___cdog__debug__Action__ctrl__enum

Values:

enumerator kCDOG_DebugHaltCtrl_Run

enumerator kCDOG_DebugHaltCtrl_Pause

enum ___cdog__irq__pause__ctrl__enum

Values:

enumerator kCDOG_IrqPauseCtrl_Run

enumerator kCDOG_IrqPauseCtrl_Pause

enum ___cdog__fault__ctrl__enum

Values:

enumerator kCDOG__FaultCtrl__EnableReset

enumerator kCDOG__FaultCtrl__EnableInterrupt

enumerator kCDOG__FaultCtrl__NoAction

enum ___code__lock__ctrl__enum

Values:

enumerator kCDOG__LockCtrl__Lock

enumerator kCDOG__LockCtrl__Unlock

typedef uint32_t secure__counter__t

SC__ADD(add)

SC__ADD1

SC__ADD16

SC__ADD256

SC__SUB(sub)

SC__SUB1

SC__SUB16

SC__SUB256

SC__CHECK(val)

struct cdog__config__t

#include <fsl_cdog.h>

2.4 Clock Driver

enum __clock__ip__name

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

enumerator kCLOCK__IpInvalid

Invalid Ip Name.

enumerator kCLOCK__None

None clock gate.

enumerator kCLOCK__Rom

Clock gate name: Rom.

enumerator kCLOCK__Sram1

Clock gate name: Sram1.

enumerator kCLOCK__Sram2

Clock gate name: Sram2.

enumerator kCLOCK_Sram3
Clock gate name: Sram3.

enumerator kCLOCK_Sram4
Clock gate name: Sram4.

enumerator kCLOCK_Sram5
Clock gate name: Sram5.

enumerator kCLOCK_Sram6
Clock gate name: Sram6.

enumerator kCLOCK_Sram7
Clock gate name: Sram7.

enumerator kCLOCK_Fmu
Clock gate name: Fmu.

enumerator kCLOCK_Fmc
Clock gate name: Fmc.

enumerator kCLOCK_Flexspi
Clock gate name: Flexspi.

enumerator kCLOCK_InputMux0
Clock gate name: InputMux0.

enumerator kCLOCK_InputMux
Clock gate name: InputMux0.

enumerator kCLOCK_Port0
Clock gate name: Port0.

enumerator kCLOCK_Port1
Clock gate name: Port1.

enumerator kCLOCK_Port2
Clock gate name: Port2.

enumerator kCLOCK_Port3
Clock gate name: Port3.

enumerator kCLOCK_Port4
Clock gate name: Port4.

enumerator kCLOCK_Gpio0
Clock gate name: Gpio0.

enumerator kCLOCK_Gpio1
Clock gate name: Gpio1.

enumerator kCLOCK_Gpio2
Clock gate name: Gpio2.

enumerator kCLOCK_Gpio3
Clock gate name: Gpio3.

enumerator kCLOCK_Gpio4
Clock gate name: Gpio4.

enumerator kCLOCK_Pint
Clock gate name: Pint.

enumerator kCLOCK_Dma0
Clock gate name: Dma0.

enumerator kCLOCK_Crc0
Clock gate name: Crc.

enumerator kCLOCK_Wwdt0
Clock gate name: Wwdt0.

enumerator kCLOCK_Wwdt1
Clock gate name: Wwdt1.

enumerator kCLOCK_Mailbox
Clock gate name: Mailbox.

enumerator kCLOCK_Mrt
Clock gate name: Mrt.

enumerator kCLOCK_OsTimer
Clock gate name: OsTimer.

enumerator kCLOCK_Sct
Clock gate name: Sct.

enumerator kCLOCK_Adc0
Clock gate name: Adc0.

enumerator kCLOCK_Adc1
Clock gate name: Adc1.

enumerator kCLOCK_Dac0
Clock gate name: Dac0.

enumerator kCLOCK_Rtc0
Clock gate name: Rtc.

enumerator kCLOCK_Evsim0
Clock gate name: Evsim0.

enumerator kCLOCK_Evsim1
Clock gate name: Evsim1.

enumerator kCLOCK_Utick
Clock gate name: Utick.

enumerator kCLOCK_LPFlexComm0
Clock gate name: LPFlexComm0.

enumerator kCLOCK_LPFlexComm1
Clock gate name: LPFlexComm1.

enumerator kCLOCK_LPFlexComm2
Clock gate name: LPFlexComm2.

enumerator kCLOCK_LPFlexComm3
Clock gate name: LPFlexComm3.

enumerator kCLOCK_LPFlexComm4
Clock gate name: LPFlexComm4.

enumerator kCLOCK_LPFlexComm5
Clock gate name: LPFlexComm5.

enumerator kCLOCK_LPFlexComm6
Clock gate name: LPFlexComm6.

enumerator kCLOCK_LPFlexComm7
Clock gate name: LPFlexComm7.

enumerator kCLOCK_LPFlexComm8
Clock gate name: LPFlexComm8.

enumerator kCLOCK_LPFlexComm9
Clock gate name: LPFlexComm9.

enumerator kCLOCK_LPUart0
Clock gate name: LPUart0.

enumerator kCLOCK_LPUart1
Clock gate name: LPUart1.

enumerator kCLOCK_LPUart2
Clock gate name: LPUart2.

enumerator kCLOCK_LPUart3
Clock gate name: LPUart3.

enumerator kCLOCK_LPUart4
Clock gate name: LPUart4.

enumerator kCLOCK_LPUart5
Clock gate name: LPUart5.

enumerator kCLOCK_LPUart6
Clock gate name: LPUart6.

enumerator kCLOCK_LPUart7
Clock gate name: LPUart7.

enumerator kCLOCK_LPUart8
Clock gate name: LPUart8.

enumerator kCLOCK_LPUart9
Clock gate name: LPUart9.

enumerator kCLOCK_LPSpi0
Clock gate name: LPSpi0.

enumerator kCLOCK_LPSpi1
Clock gate name: LPSpi1.

enumerator kCLOCK_LPSpi2
Clock gate name: LPSpi2.

enumerator kCLOCK_LPSpi3
Clock gate name: LPSpi3.

enumerator kCLOCK_LPSpi4
Clock gate name: LPSpi4.

enumerator kCLOCK_LPSpi5
Clock gate name: LPSpi5.

enumerator kCLOCK_LPSpi6
Clock gate name: LPSpi6.

enumerator kCLOCK_LPSpi7
Clock gate name: LPSpi7.

enumerator kCLOCK_LPSpi8
Clock gate name: LPSpi8.

enumerator kCLOCK_LPSpi9
Clock gate name: LSpi9.

enumerator kCLOCK_LPI2c0
Clock gate name: LPI2c0.

enumerator kCLOCK_LPI2c1
Clock gate name: LPI2c1.

enumerator kCLOCK_LPI2c2
Clock gate name: LPI2c2.

enumerator kCLOCK_LPI2c3
Clock gate name: LPI2c3.

enumerator kCLOCK_LPI2c4
Clock gate name: LPI2c4.

enumerator kCLOCK_LPI2c5
Clock gate name: LPI2c5.

enumerator kCLOCK_LPI2c6
Clock gate name: LPI2c6.

enumerator kCLOCK_LPI2c7
Clock gate name: LPI2c7.

enumerator kCLOCK_LPI2c8
Clock gate name: LPI2c8.

enumerator kCLOCK_LPI2c9
Clock gate name: LPI2c9.

enumerator kCLOCK_Micfil
Clock gate name: Micfil.

enumerator kCLOCK_Timer2
Clock gate name: Timer2.

enumerator kCLOCK_Usb0Ram
Clock gate name: Usb0Ram.

enumerator kCLOCK_Usb0FsDcd
Clock gate name: Usb0FsDcd.

enumerator kCLOCK_Usb0Fs
Clock gate name: Usb0Fs.

enumerator kCLOCK_Timer0
Clock gate name: Timer0.

enumerator kCLOCK_Timer1
Clock gate name: Timer1.

enumerator kCLOCK_PkcRam
Clock gate name: PkcRam.

enumerator kCLOCK_Smartdma
Clock gate name: SmartDma.

enumerator kCLOCK_Espi
Clock gate name: Espi.

enumerator kCLOCK_Dma1
Clock gate name: Dma1.

enumerator kCLOCK_Enet
Clock gate name: Enet.

enumerator kCLOCK_uSdhc
Clock gate name: uSdhc.

enumerator kCLOCK_Flexio
Clock gate name: Flexio.

enumerator kCLOCK_Sai0
Clock gate name: Sai0.

enumerator kCLOCK_Sai1
Clock gate name: Sai1.

enumerator kCLOCK_Tro
Clock gate name: Tro.

enumerator kCLOCK_Freqme
Clock gate name: Freqme.

enumerator kCLOCK_Trng
Clock gate name: Trng.

enumerator kCLOCK_Flexcan0
Clock gate name: Flexcan0.

enumerator kCLOCK_Flexcan1
Clock gate name: Flexcan1.

enumerator kCLOCK_UsbHs
Clock gate name: UsbHs.

enumerator kCLOCK_UsbHsPhy
Clock gate name: UsbHsPhy.

enumerator kCLOCK_Css
Clock gate name: Css.

enumerator kCLOCK_PowerQuad
Clock gate name: PowerQuad.

enumerator kCLOCK_PluLut
Clock gate name: PluLut.

enumerator kCLOCK_Timer3
Clock gate name: Timer3.

enumerator kCLOCK_Timer4
Clock gate name: Timer4.

enumerator kCLOCK_Puf
Clock gate name: Puf.

enumerator kCLOCK_Pkc
Clock gate name: Pkc.

enumerator kCLOCK_Scg
Clock gate name: Scg.

enumerator kCLOCK_Gdet
Clock gate name: Gdet.

enumerator kCLOCK_Sm3
Clock gate name: Sm3.

enumerator kCLOCK_I3c0
Clock gate name: I3c0.

enumerator kCLOCK_I3c1
Clock gate name: I3c1.

enumerator kCLOCK_Sinc
Clock gate name: Sinc.

enumerator kCLOCK_CoolFlux
Clock gate name: CoolFlux.

enumerator kCLOCK_Qdc0
Clock gate name: Qdc0.

enumerator kCLOCK_Qdc1
Clock gate name: Qdc1.

enumerator kCLOCK_Pwm0
Clock gate name: Pwm0.

enumerator kCLOCK_Pwm1
Clock gate name: Pwm1.

enumerator kCLOCK_Evtg
Clock gate name: Evtg.

enumerator kCLOCK_Dac1
Clock gate name: Dac1.

enumerator kCLOCK_Dac2
Clock gate name: Dac2.

enumerator kCLOCK_Opamp0
Clock gate name: Opamp0.

enumerator kCLOCK_Opamp1
Clock gate name: Opamp1.

enumerator kCLOCK_Opamp2
Clock gate name: Opamp2.

enumerator kCLOCK_Cmp2
Clock gate name: Cmp2.

enumerator kCLOCK_Vref
Clock gate name: Vref.

enumerator kCLOCK_CoolFluxApb
Clock gate name: CoolFluxApb.

enumerator kCLOCK_Neutron
Clock gate name: Neutron.

enumerator kCLOCK_Tsi
Clock gate name: Tsi.

enumerator kCLOCK_Ewm
Clock gate name: Ewm.

enumerator kCLOCK_Ewm0
Clock gate name: Ewm.

enumerator kCLOCK_Eim
Clock gate name: Eim.

enumerator kCLOCK_Erm
Clock gate name: Erm.

enumerator kCLOCK_Intm
Clock gate name: Intm.

enumerator kCLOCK_Sema42
Clock gate name: Sema42.

enumerator kCLOCK_Pwm0_Sm0
Clock gate name: PWM0 SM0.

enumerator kCLOCK_Pwm0_Sm1
Clock gate name: PWM0 SM1.

enumerator kCLOCK_Pwm0_Sm2
Clock gate name: PWM0 SM2.

enumerator kCLOCK_Pwm0_Sm3
Clock gate name: PWM0 SM3.

enumerator kCLOCK_Pwm1_Sm0
Clock gate name: PWM1 SM0.

enumerator kCLOCK_Pwm1_Sm1
Clock gate name: PWM1 SM1.

enumerator kCLOCK_Pwm1_Sm2
Clock gate name: PWM1 SM2.

enumerator kCLOCK_Pwm1_Sm3
Clock gate name: PWM1 SM3.

enum _clock_name
Clock name used to get clock frequency.

Values:

enumerator kCLOCK_MainClk
Main clock

enumerator kCLOCK_CoreSysClk
Core/system clock

enumerator kCLOCK_BusClk
Bus clock (AHB clock)

enumerator kCLOCK_SystickClk0
Systick clock0

enumerator kCLOCK_SystickClk1
Systick clock1

enumerator kCLOCK_ClockOut
CLOCKOUT

enumerator kCLOCK_Fro12M
FRO12M

enumerator kCLOCK_Clk1M
CLK1M

enumerator kCLOCK_FroHf
FRO48/144

enumerator kCLOCK_Clk48M
CLK48M

enumerator kCLOCK_Clk144M
CLK144M

enumerator kCLOCK_Clk16K0
CLK16K[0]

enumerator kCLOCK_Clk16K1
CLK16K[1]

enumerator kCLOCK_Clk16K2
CLK16K[2]

enumerator kCLOCK_Clk16K3
CLK16K[3]

enumerator kCLOCK_ExtClk
External Clock

enumerator kCLOCK_Osc32K0
OSC32K[0]

enumerator kCLOCK_Osc32K1
OSC32K[1]

enumerator kCLOCK_Osc32K2
OSC32K[2]

enumerator kCLOCK_Osc32K3
OSC32K[3]

enumerator kCLOCK_Pll0Out
PLL0 Output

enumerator kCLOCK_Pll1Out
PLL1 Output

enumerator kCLOCK_UsbPllOut
USB PLL Output

enumerator kCLOCK_LpOsc
lp_osc

enum _clock_attach_id

The enumerator of clock attach Id.

Values:

enumerator kCLK_IN_to_MAIN_CLK
Attach clk_in to MAIN_CLK.

enumerator kFRO12M_to_MAIN_CLK
Attach FRO_12M to MAIN_CLK.

enumerator kFRO_HF_to_MAIN_CLK
Attach FRO_HF to MAIN_CLK.

enumerator kXTAL32K2_to_MAIN_CLK
Attach xtal32k[2] to MAIN_CLK.

enumerator kPLL0_to_MAIN_CLK
Attach PLL0 to MAIN_CLK.

enumerator kPLL1_to_MAIN_CLK
Attach PLL1 to MAIN_CLK.

enumerator kUSB_PLL_to_MAIN_CLK
Attach USB PLL to MAIN_CLK.

enumerator kNONE_to_MAIN_CLK
Attach NONE to MAIN_CLK.

enumerator kSYSTICK_DIV0_to_SYSTICK0
Attach SYSTICK_DIV0 to SYSTICK0.

enumerator kCLK_1M_to_SYSTICK0
Attach Clk 1 MHz to SYSTICK0.

enumerator kLPOSC_to_SYSTICK0
Attach LP Oscillator to SYSTICK0.

enumerator kNONE_to_SYSTICK0
Attach NONE to SYSTICK0.

enumerator kSYSTICK_DIV1_to_SYSTICK1
Attach SYSTICK_DIV1 to SYSTICK1.

enumerator kCLK_1M_to_SYSTICK1
Attach Clk 1 MHz to SYSTICK1.

enumerator kLPOSC_to_SYSTICK1
Attach LP Oscillator to SYSTICK1.

enumerator kNONE_to_SYSTICK1
Attach NONE to SYSTICK1.

enumerator kTRACE_DIV_to_TRACE
Attach TRACE_DIV to TRACE.

enumerator kCLK_1M_to_TRACE
Attach Clk 1 MHz to TRACE.

enumerator kLPOSC_to_TRACE
Attach LP Oscillator to TRACE.

enumerator kNONE_to_TRACE
Attach NONE to TRACE.

enumerator kCLK_1M_to_CTIMER0
Attach CLK_1M to CTIMER0.

enumerator kPLL0_to_CTIMER0
Attach PLL0 to CTIMER0.

enumerator kPLL1_CLK0_to_CTIMER0
Attach PLL1_clk0 to CTIMER0.

enumerator kFRO_HF_to_CTIMER0
Attach FRO_HF to CTIMER0.

enumerator kFRO12M_to_CTIMER0
Attach FRO 12MHz to CTIMER0.

enumerator kSAI0_MCLK_IN_to_CTIMER0
Attach SAI0 MCLK IN to CTIMER0.

enumerator kLPOSC_to_CTIMER0
Attach LP Oscillator to CTIMER0.

enumerator kSAI1_MCLK_IN_to_CTIMER0
Attach SAI1 MCLK IN to CTIMER0.

enumerator kSAI0_TX_BCLK_to_CTIMER0
Attach SAI0 TX_BCLK to CTIMER0.

enumerator kSAI0_RX_BCLK_to_CTIMER0
Attach SAI0 RX_BCLK to CTIMER0.

enumerator kSAI1_TX_BCLK_to_CTIMER0
Attach SAI1 TX_BCLK to CTIMER0.

enumerator kSAI1_RX_BCLK_to_CTIMER0
Attach SAI1 RX_BCLK to CTIMER0.

enumerator kNONE_to_CTIMER0
Attach NONE to CTIMER0.

enumerator kCLK_1M_to_CTIMER1
Attach CLK_1M to CTIMER1.

enumerator kPLL0_to_CTIMER1
Attach PLL0 to CTIMER1.

enumerator kPLL1_CLK0_to_CTIMER1
Attach PLL1_clk0 to CTIMER1.

enumerator kFRO_HF_to_CTIMER1
Attach FRO_HF to CTIMER1.

enumerator kFRO12M_to_CTIMER1
Attach FRO 12MHz to CTIMER1.

enumerator kSAI0_MCLK_IN_to_CTIMER1
Attach SAI0 MCLK IN to CTIMER1.

enumerator kLPOSC_to_CTIMER1
Attach LP Oscillator to CTIMER1.

enumerator kSAI1_MCLK_IN_to_CTIMER1
Attach SAI1 MCLK IN to CTIMER1.

enumerator kSAI0_TX_BCLK_to_CTIMER1
Attach SAI0 TX_BCLK to CTIMER1.

enumerator kSAI0_RX_BCLK_to_CTIMER1
Attach SAI0 RX_BCLK to CTIMER1.

enumerator kSAI1_TX_BCLK_to_CTIMER1
Attach SAI1 TX_BCLK to CTIMER1.

enumerator kSAI1_RX_BCLK_to_CTIMER1
Attach SAI1 RX_BCLK to CTIMER1.

enumerator kNONE_to_CTIMER1
Attach NONE to CTIMER1.

enumerator kCLK_1M_to_CTIMER2
Attach CLK_1M to CTIMER2.

enumerator kPLL0_to_CTIMER2
Attach PLL0 to CTIMER2.

enumerator kPLL1_CLK0_to_CTIMER2
Attach PLL1_clk0 to CTIMER2.

enumerator kFRO_HF_to_CTIMER2
Attach FRO_HF to CTIMER2.

enumerator kFRO12M_to_CTIMER2
Attach FRO 12MHz to CTIMER2.

enumerator kSAI0_MCLK_IN_to_CTIMER2
Attach SAI0 MCLK IN to CTIMER2.

enumerator kLPOSC_to_CTIMER2
Attach LP Oscillator to CTIMER2.

enumerator kSAI1_MCLK_IN_to_CTIMER2
Attach SAI1 MCLK IN to CTIMER2.

enumerator kSAI0_TX_BCLK_to_CTIMER2
Attach SAI0 TX_BCLK to CTIMER2.

enumerator kSAI0_RX_BCLK_to_CTIMER2
Attach SAI0 RX_BCLK to CTIMER2.

enumerator kSAI1_TX_BCLK_to_CTIMER2
Attach SAI1 TX_BCLK to CTIMER2.

enumerator kSAI1_RX_BCLK_to_CTIMER2
Attach SAI1 RX_BCLK to CTIMER2.

enumerator kNONE_to_CTIMER2
Attach NONE to CTIMER2.

enumerator kCLK_1M_to_CTIMER3
Attach CLK_1M to CTIMER3.

enumerator kPLL0_to_CTIMER3
Attach PLL0 to CTIMER3.

enumerator kPLL1_CLK0_to_CTIMER3
Attach PLL1_clk0 to CTIMER3.

enumerator kFRO_HF_to_CTIMER3
Attach FRO_HF to CTIMER3.

enumerator kFRO12M_to_CTIMER3
Attach FRO 12MHz to CTIMER3.

enumerator kSAI0_MCLK_IN_to_CTIMER3
Attach SAI0 MCLK IN to CTIMER3.

enumerator kLPOSC_to_CTIMER3
Attach LP Oscillator to CTIMER3.

enumerator kSAI1_MCLK_IN_to_CTIMER3
Attach SAI1 MCLK IN to CTIMER3.

enumerator kSAI0_TX_BCLK_to_CTIMER3
Attach SAI0 TX_BCLK to CTIMER3.

enumerator kSAI0_RX_BCLK_to_CTIMER3
Attach SAI0 RX_BCLK to CTIMER3.

enumerator kSAI1_TX_BCLK_to_CTIMER3
Attach SAI1 TX_BCLK to CTIMER3.

enumerator kSAI1_RX_BCLK_to_CTIMER3
Attach SAI1 RX_BCLK to CTIMER3.

enumerator kNONE_to_CTIMER3
Attach NONE to CTIMER3.

enumerator kCLK_1M_to_CTIMER4
Attach CLK_1M to CTIMER4.

enumerator kPLL0_to_CTIMER4
Attach PLL0 to CTIMER4.

enumerator kPLL1_CLK0_to_CTIMER4
Attach PLL1_clk0 to CTIMER4.

enumerator kFRO_HF_to_CTIMER4
Attach FRO_HF to CTIMER4.

enumerator kFRO12M_to_CTIMER4
Attach FRO 12MHz to CTIMER4.

enumerator kSAI0_MCLK_IN_to_CTIMER4
Attach SAI0 MCLK IN to CTIMER4.

enumerator kLPOSC_to_CTIMER4
Attach LP Oscillator to CTIMER4.

enumerator kSAI1_MCLK_IN_to_CTIMER4
Attach SAI1 MCLK IN to CTIMER4.

enumerator kSAI0_TX_BCLK_to_CTIMER4
Attach SAI0 TX_BCLK to CTIMER4.

enumerator kSAI0_RX_BCLK_to_CTIMER4
Attach SAI0 RX_BCLK to CTIMER4.

enumerator kSAI1_TX_BCLK_to_CTIMER4
Attach SAI1 TX_BCLK to CTIMER4.

enumerator kSAI1_RX_BCLK_to_CTIMER4
Attach SAI1 RX_BCLK to CTIMER4.

enumerator kNONE_to_CTIMER4
Attach NONE to CTIMER4.

enumerator kMAIN_CLK_to_CLKOUT
Attach MAIN_CLK to CLKOUT.

enumerator kPLL0_to_CLKOUT
Attach PLL0 to CLKOUT.

enumerator kCLK_IN_to_CLKOUT
Attach Clk_in to CLKOUT.

enumerator kFRO_HF_to_CLKOUT
Attach FRO_HF to CLKOUT.

enumerator kFRO12M_to_CLKOUT
Attach FRO 12 MHz to CLKOUT.

enumerator kPLL1_CLK0_to_CLKOUT
Attach PLL1_clk0 to CLKOUT.

enumerator kLPOSC_to_CLKOUT
Attach LP Oscillator to CLKOUT.

enumerator kUSB_PLL_to_CLKOUT
Attach USB_PLL to CLKOUT.

enumerator kNONE_to_CLKOUT
Attach NONE to CLKOUT.

enumerator kPLL0_to_ADC0
Attach PLL0 to ADC0.

enumerator kFRO_HF_to_ADC0
Attach FRO_HF to ADC0.

enumerator kFRO12M_to_ADC0
Attach FRO 12 MHz to ADC0.

enumerator kCLK_IN_to_ADC0
Attach Clk_in to ADC0.

enumerator kPLL1_CLK0_to_ADC0
Attach PLL1_clk0 to ADC0.

enumerator kUSB_PLL_to_ADC0
Attach USB PLL to ADC0.

enumerator kNONE_to_ADC0
Attach NONE to ADC0.

enumerator kPLL0_to_USB0
Attach PLL0 to USB0.

enumerator kCLK_48M_to_USB0
Attach Clk 48 MHz to USB0.

enumerator kCLK_IN_to_USB0
Attach Clk_in to USB0.

enumerator kPLL1_CLK0_to_USB0
Attach PLL1_clk0 to USB0.

enumerator kUSB_PLL_to_USB0
Attach USB PLL to USB0.

enumerator kNONE_to_USB0
Attach NONE to USB0.

enumerator kPLL_DIV_to_FLEXCOMM0
Attach PLL_DIV to FLEXCOMM0.

enumerator kFRO12M_to_FLEXCOMM0
Attach FRO12M to FLEXCOMM0.

enumerator kFRO_HF_DIV_to_FLEXCOMM0
Attach FRO_HF_DIV to FLEXCOMM0.

enumerator kCLK_1M_to_FLEXCOMM0
Attach CLK_1MHz to FLEXCOMM0.

enumerator kUSB_PLL_to_FLEXCOMM0
Attach USB_PLL to FLEXCOMM0.

enumerator kLPOSC_to_FLEXCOMM0
Attach LP Oscillator to FLEXCOMM0.

enumerator kNONE_to_FLEXCOMM0
Attach NONE to FLEXCOMM0.

enumerator kPLL_DIV_to_FLEXCOMM1
Attach PLL_DIV to FLEXCOMM1.

enumerator kFRO12M_to_FLEXCOMM1
Attach FRO12M to FLEXCOMM1.

enumerator kFRO_HF_DIV_to_FLEXCOMM1
Attach FRO_HF_DIV to FLEXCOMM1.

enumerator kCLK_1M_to_FLEXCOMM1
Attach CLK_1MHz to FLEXCOMM1.

enumerator kUSB_PLL_to_FLEXCOMM1
Attach USB_PLL to FLEXCOMM1.

enumerator kLPOSC_to_FLEXCOMM1
Attach LP Oscillator to FLEXCOMM1.

enumerator kNONE_to_FLEXCOMM1
Attach NONE to FLEXCOMM1.

enumerator kPLL_DIV_to_FLEXCOMM2
Attach PLL_DIV to FLEXCOMM2.

enumerator kFRO12M_to_FLEXCOMM2
Attach FRO12M to FLEXCOMM2.

enumerator kFRO_HF_DIV_to_FLEXCOMM2
Attach FRO_HF_DIV to FLEXCOMM2.

enumerator kCLK_1M_to_FLEXCOMM2
Attach CLK_1MHz to FLEXCOMM2.

enumerator kUSB_PLL_to_FLEXCOMM2
Attach USB_PLL to FLEXCOMM2.

enumerator kLPOSC_to_FLEXCOMM2
Attach LP Oscillator to FLEXCOMM2.

enumerator kNONE_to_FLEXCOMM2
Attach NONE to FLEXCOMM2.

enumerator kPLL_DIV_to_FLEXCOMM3
Attach PLL_DIV to FLEXCOMM3.

enumerator kFRO12M_to_FLEXCOMM3
Attach FRO12M to FLEXCOMM3.

enumerator kFRO_HF_DIV_to_FLEXCOMM3
Attach FRO_HF_DIV to FLEXCOMM3.

enumerator kCLK_1M_to_FLEXCOMM3
Attach CLK_1MHz to FLEXCOMM3.

enumerator kUSB_PLL_to_FLEXCOMM3
Attach USB_PLL to FLEXCOMM3.

enumerator kLPOSC_to_FLEXCOMM3
Attach LP Oscillator to FLEXCOMM3.

enumerator kNONE_to_FLEXCOMM3
Attach NONE to FLEXCOMM3.

enumerator kPLL_DIV_to_FLEXCOMM4
Attach PLL_DIV to FLEXCOMM4.

enumerator kFRO12M_to_FLEXCOMM4
Attach FRO12M to FLEXCOMM4.

enumerator kFRO_HF_DIV_to_FLEXCOMM4
Attach FRO_HF_DIV to FLEXCOMM4.

enumerator kCLK_1M_to_FLEXCOMM4
Attach CLK_1MHz to FLEXCOMM4.

enumerator kUSB_PLL_to_FLEXCOMM4
Attach USB_PLL to FLEXCOMM4.

enumerator kLPOSC_to_FLEXCOMM4
Attach LP Oscillator to FLEXCOMM4.

enumerator kNONE_to_FLEXCOMM4
Attach NONE to FLEXCOMM4.

enumerator kPLL_DIV_to_FLEXCOMM5
Attach PLL_DIV to FLEXCOMM5.

enumerator kFRO12M_to_FLEXCOMM5
Attach FRO12M to FLEXCOMM5.

enumerator kFRO_HF_DIV_to_FLEXCOMM5
Attach FRO_HF_DIV to FLEXCOMM5.

enumerator kCLK_1M_to_FLEXCOMM5
Attach CLK_1MHz to FLEXCOMM5.

enumerator kUSB_PLL_to_FLEXCOMM5
Attach USB_PLL to FLEXCOMM5.

enumerator kLPOSC_to_FLEXCOMM5
Attach LP Oscillator to FLEXCOMM5.

enumerator kNONE_to_FLEXCOMM5
Attach NONE to FLEXCOMM5.

enumerator kPLL_DIV_to_FLEXCOMM6
Attach PLL_DIV to FLEXCOMM6.

enumerator kFRO12M_to_FLEXCOMM6
Attach FRO12M to FLEXCOMM6.

enumerator kFRO_HF_DIV_to_FLEXCOMM6
Attach FRO_HF_DIV to FLEXCOMM6.

enumerator kCLK_1M_to_FLEXCOMM6
Attach CLK_1MHz to FLEXCOMM6.

enumerator kUSB_PLL_to_FLEXCOMM6
Attach USB_PLL to FLEXCOMM6.

enumerator kLPOSC_to_FLEXCOMM6
Attach LP Oscillator to FLEXCOMM6.

enumerator kNONE_to_FLEXCOMM6
Attach NONE to FLEXCOMM6.

enumerator kPLL_DIV_to_FLEXCOMM7
Attach PLL_DIV to FLEXCOMM7.

enumerator kFRO12M_to_FLEXCOMM7
Attach FRO12M to FLEXCOMM7.

enumerator kFRO_HF_DIV_to_FLEXCOMM7
Attach FRO_HF_DIV to FLEXCOMM7.

enumerator kCLK_1M_to_FLEXCOMM7
Attach CLK_1MHz to FLEXCOMM7.

enumerator kUSB_PLL_to_FLEXCOMM7
Attach USB_PLL to FLEXCOMM7.

enumerator kLPOSC_to_FLEXCOMM7
Attach LP Oscillator to FLEXCOMM7.

enumerator kNONE_to_FLEXCOMM7
Attach NONE to FLEXCOMM7.

enumerator kPLL_DIV_to_FLEXCOMM8
Attach PLL_DIV to FLEXCOMM8.

enumerator kFRO12M_to_FLEXCOMM8
Attach FRO12M to FLEXCOMM8.

enumerator kFRO_HF_DIV_to_FLEXCOMM8
Attach FRO_HF_DIV to FLEXCOMM8.

enumerator kCLK_1M_to_FLEXCOMM8
Attach CLK_1MHz to FLEXCOMM8.

enumerator kUSB_PLL_to_FLEXCOMM8
Attach USB_PLL to FLEXCOMM8.

enumerator kLPOSC_to_FLEXCOMM8
Attach LP Oscillator to FLEXCOMM8.

enumerator kNONE_to_FLEXCOMM8
Attach NONE to FLEXCOMM8.

enumerator kPLL_DIV_to_FLEXCOMM9
Attach PLL_DIV to FLEXCOMM9.

enumerator kFRO12M_to_FLEXCOMM9
Attach FRO12M to FLEXCOMM9.

enumerator kFRO_HF_DIV_to_FLEXCOMM9
Attach FRO_HF_DIV to FLEXCOMM9.

enumerator kCLK_1M_to_FLEXCOMM9
Attach CLK_1MHz to FLEXCOMM9.

enumerator kUSB_PLL_to_FLEXCOMM9
Attach USB_PLL to FLEXCOMM9.

enumerator kLPOSC_to_FLEXCOMM9
Attach LP Oscillator to FLEXCOMM9.

enumerator kNONE_to_FLEXCOMM9
Attach NONE to FLEXCOMM9.

enumerator kPLL0_to_SCT
Attach NONE to SCT.

enumerator kCLK_IN_to_SCT
Attach CLK_in to SCT.

enumerator kFRO_HF_to_SCT
Attach FRO_HF to SCT.

enumerator kPLL1_CLK0_to_SCT
Attach PLL1_clk0 to SCT.

enumerator kSAI0_MCLK_IN_to_SCT
Attach SAI0 MCLK_IN to SCT.

enumerator kUSB_PLL_to_SCT
Attach USB PLL to SCT.

enumerator kSAI1_MCLK_IN_to_SCT
Attach SAI1 MCLK_IN to SCT.

enumerator kNONE_to_SCT
Attach NONE to SCT.

enumerator kCLK_IN_to_TSI
Attach clk_in to TSI.

enumerator kFRO12M_to_TSI
Attach FRO_12Mhz to TSI.

enumerator kNONE_to_TSI
Attach NONE to TSI.

enumerator kPLL0_to_SINCFILT
Attach PLL0 to SINCFILT.

enumerator kCLK_IN_to_SINCFILT
Attach clk_in to SINCFILT.

enumerator kFRO_HF_to_SINCFILT
Attach FRO_HF to SINCFILT.

enumerator kFRO12M_to_SINCFILT
Attach FRO_12Mhz to SINCFILT.

enumerator kPLL1_CLK0_to_SINCFILT
Attach PLL1_clk0 to SINCFILT.

enumerator kUSB_PLL_to_SINCFILT
Attach USB PLL to SINCFILT.

enumerator kNONE_to_SINCFILT
Attach NONE to SINCFILT.

enumerator kPLL0_to_ADC1
Attach PLL0 to ADC1.

enumerator kFRO_HF_to_ADC1
Attach FRO_HF to ADC1.

enumerator kFRO12M_to_ADC1
Attach FRO12M to ADC1.

enumerator kCLK_IN_to_ADC1
Attach clk_in to ADC1.

enumerator kPLL1_CLK0_to_ADC1
Attach PLL1_clk0 to ADC1.

enumerator kUSB_PLL_to_ADC1
Attach USB PLL to ADC1.

enumerator kNONE_to_ADC1
Attach NONE to ADC1.

enumerator kPLL0_to_DAC0
Attach PLL0 to DAC0.

enumerator kCLK_IN_to_DAC0
Attach Clk_in to DAC0.

enumerator kFRO_HF_to_DAC0
Attach FRO_HF to DAC0.

enumerator kFRO12M_to_DAC0
Attach FRO_12M to DAC0.

enumerator kPLL1_CLK0_to_DAC0
Attach PLL1_clk0 to DAC0.

enumerator kNONE_to_DAC0
Attach NONE to DAC0.

enumerator kPLL0_to_DAC1
Attach PLL0 to DAC1.

enumerator kCLK_IN_to_DAC1
Attach Clk_in to DAC1.

enumerator kFRO_HF_to_DAC1
Attach FRO_HF to DAC1.

enumerator kFRO12M_to_DAC1
Attach FRO_12M to DAC1.

enumerator kPLL1_CLK0_to_DAC1
Attach PLL1_clk0 to DAC1.

enumerator kNONE_to_DAC1
Attach NONE to DAC1.

enumerator kPLL0_to_DAC2
Attach PLL0 to DAC2.

enumerator kCLK_IN_to_DAC2
Attach Clk_in to DAC2.

enumerator kFRO_HF_to_DAC2
Attach FRO_HF to DAC2.

enumerator kFRO12M_to_DAC2
Attach FRO_12M to DAC2.

enumerator kPLL1_CLK0_to_DAC2
Attach PLL1_clk0 to DAC2.

enumerator kNONE_to_DAC2
Attach NONE to DAC2.

enumerator kPLL0_to_FLEXSPI
Attach PLL0 to FLEXSPI.

enumerator kFRO_HF_to_FLEXSPI
Attach FRO_HF to FLEXSPI.

enumerator kPLL1_to_FLEXSPI
Attach PLL1 to FLEXSPI.

enumerator kUSB_PLL_to_FLEXSPI
Attach USB PLL to FLEXSPI.

enumerator kNONE_to_FLEXSPI
Attach NONE to FLEXSPI.

enumerator kPLL0_to_PLLCLKDIV
Attach PLL0 to PLLCLKDIV.

enumerator kPLL1_CLK0_to_PLLCLKDIV
Attach pll1_clk0 to PLLCLKDIV.

enumerator kNONE_to_PLLCLKDIV
Attach NONE to PLLCLKDIV.

enumerator kPLL0_to_I3C0FCLK
Attach PLL0 to I3C0FCLK.

enumerator kCLK_IN_to_I3C0FCLK
Attach Clk_in to I3C0FCLK.

enumerator kFRO_HF_to_I3C0FCLK
Attach FRO_HF to I3C0FCLK.

enumerator kPLL1_CLK0_to_I3C0FCLK
Attach PLL1_clk0 to I3C0FCLK.

enumerator kUSB_PLL_to_I3C0FCLK
Attach USB PLL to I3C0FCLK.

enumerator kNONE_to_I3C0FCLK
Attach NONE to I3C0FCLK.

enumerator kI3C0FCLK_to_I3C0FCLKSTC
Attach I3C0FCLK to I3C0FCLKSTC.

enumerator kCLK_1M_to_I3C0FCLKSTC
Attach CLK_1M to I3C0FCLKSTC.

enumerator kNONE_to_I3C0FCLKSTC
Attach NONE to I3C0FCLKSTC.

enumerator kCLK_1M_to_I3C0FCLKS
Attach CLK_1M to I3C0FCLKS.

enumerator kNONE_to_I3C0FCLKS
Attach NONE to I3C0FCLKS.

enumerator kFRO12M_to_MICFILF
Attach FRO_12M to MICFILF.

enumerator kPLL0_to_MICFILF
Attach PLL0 to MICFILF.

enumerator kCLK_IN_to_MICFILF
Attach Clk_in to MICFILF.

enumerator kFRO_HF_to_MICFILF
Attach FRO_HF to MICFILF.

enumerator kPLL1_CLK0_to_MICFILF
Attach PLL1_clk0 to MICFILF.

enumerator kSAI0_MCLK_IN_to_MICFILF
Attach SAI0_MCLK to MICFILF.

enumerator kUSB_PLL_to_MICFILF
Attach USB PLL to MICFILF.

enumerator kSAI1_MCLK_IN_to_MICFILF
Attach SAI1_MCLK to MICFILF.

enumerator kNONE_to_MICFILF
Attach NONE to MICFILF.

enumerator kPLL0_to_ESPI
Attach PLL0 to ESPI.

enumerator kCLK_48M_to_ESPI
Attach CLK_48M to ESPI.

enumerator kPLL1_CLK0_to_ESPI
Attach PLL1_clk0 to ESPI.

enumerator kUSB_PLL_to_ESPI
Attach USB PLL to ESPI.

enumerator kNONE_to_ESPI
Attach NONE to ESPI.

enumerator kPLL0_to_USDHC
Attach PLL0 to uSDHC.

enumerator kCLK_IN_to_USDHC
Attach Clk_in to uSDHC.

enumerator kFRO_HF_to_USDHC
Attach FRO_HF to uSDHC.

enumerator kFRO12M_to_USDHC
Attach FRO_12M to uSDHC.

enumerator kPLL1_CLK1_to_USDHC
Attach pll1_clk1 to uSDHC.

enumerator kUSB_PLL_to_USDHC
Attach USB PLL to uSDHC.

enumerator kNONE_to_USDHC
Attach NONE to uSDHC.

enumerator kPLL0_to_FLEXIO
Attach PLL0 to FLEXIO.

enumerator kCLK_IN_to_FLEXIO
Attach Clk_in to FLEXIO.

enumerator kFRO_HF_to_FLEXIO
Attach FRO_HF to FLEXIO.

enumerator kFRO12M_to_FLEXIO
Attach FRO_12M to FLEXIO.

enumerator kPLL1_CLK0_to_FLEXIO
Attach pll1_clk0 to FLEXIO.

enumerator kUSB_PLL_to_FLEXIO
Attach USB PLL to FLEXIO.

enumerator kNONE_to_FLEXIO
Attach NONE to FLEXIO.

enumerator kPLL0_to_FLEXCAN0
Attach PLL0 to FLEXCAN0.

enumerator kCLK_IN_to_FLEXCAN0
Attach Clk_in to FLEXCAN0.

enumerator kFRO_HF_to_FLEXCAN0
Attach FRO_HF to FLEXCAN0.

enumerator kPLL1_CLK0_to_FLEXCAN0
Attach pll1_clk0 to FLEXCAN0.

enumerator kUSB_PLL_to_FLEXCAN0
Attach USB PLL to FLEXCAN0.

enumerator kNONE_to_FLEXCAN0
Attach NONE to FLEXCAN0.

enumerator kPLL0_to_FLEXCAN1
Attach PLL0 to FLEXCAN1.

enumerator kCLK_IN_to_FLEXCAN1
Attach Clk_in to FLEXCAN1.

enumerator kFRO_HF_to_FLEXCAN1
Attach FRO_HF to FLEXCAN1.

enumerator kPLL1_CLK0_to_FLEXCAN1
Attach pll1_clk0 to FLEXCAN1.

enumerator kUSB_PLL_to_FLEXCAN1
Attach USB PLL to FLEXCAN1.

enumerator kNONE_to_FLEXCAN1
Attach NONE to FLEXCAN1.

enumerator kNONE_to_ENETRMII
Attach NONE to ENETRMII.

enumerator kPLL0_to_ENETRMII
Attach PLL0 to ENETRMII.

enumerator kCLK_IN_to_ENETRMII
Attach Clk_in to ENETRMII.

enumerator kPLL1_CLK0_to_ENETRMII
Attach pll1_clk0 to ENETRMII.

enumerator kPLL0_to_ENETPTPREF
Attach PLL0 to ENETPTPREF.

enumerator kCLK_IN_to_ENETPTPREF
Attach Clk_in to ENETPTPREF.

enumerator kENET0_TX_CLK_to_ENETPTPREF
Attach enet0_tx_clk to ENETPTPREF.

enumerator kPLL1_CLK1_to_ENETPTPREF
Attach pll1_clk1 to ENETPTPREF.

enumerator kNONE_to_ENETPTPREF
Attach NONE to ENETPTPREF.

enumerator kCLK_16K2_to_EWM0
Attach clk_16k[2] to EWM0.

enumerator kXTAL32K2_to_EWM0
Attach xtal32k[2] to EWM0.

enumerator kCLK_16K2_to_WDT1
Attach FRO16K clock 2 to WDT1.

enumerator kFRO_HF_DIV_to_WDT1
Attach FRO_HF_DIV to WDT1.

enumerator kCLK_1M_to_WDT1
Attach clk_1m to WDT1.

enumerator kCLK_1M_2_to_WDT1
Attach clk_1m to WDT1.

enumerator kCLK_16K2_to_OSTIMER
Attach clk_16k[2] to OSTIMER.

enumerator kXTAL32K2_to_OSTIMER
Attach xtal32k[2] to OSTIMER.

enumerator kCLK_1M_to_OSTIMER
Attach clk_1m to OSTIMER.

enumerator kNONE_to_OSTIMER
Attach NONE to OSTIMER.

enumerator kPLL0_to_CMP0F
Attach PLL0 to CMP0F.

enumerator kFRO_HF_to_CMP0F
Attach FRO_HF to CMP0F.

enumerator kFRO12M_to_CMP0F
Attach FRO_12M to CMP0F.

enumerator kCLK_IN_to_CMP0F
Attach Clk_in to CMP0F.

enumerator kPLL1_CLK0_to_CMP0F
Attach PLL1_clk0 to CMP0F.

enumerator kUSB_PLL_to_CMP0F
Attach USB PLL to CMP0F.

enumerator kNONE_to_CMP0F
Attach NONE to CMP0F.

enumerator kPLL0_to_CMP0RR
Attach PLL0 to CMP0RR.

enumerator kFRO_HF_to_CMP0RR
Attach FRO_HF to CMP0RR.

enumerator kFRO12M_to_CMP0RR
Attach FRO_12M to CMP0RR.

enumerator kCLK_IN_to_CMP0RR
Attach Clk_in to CMP0RR.

enumerator kPLL1_CLK0_to_CMP0RR
Attach PLL1_clk0 to CMP0RR.

enumerator kUSB_PLL_to_CMP0RR
Attach USB PLL to CMP0RR.

enumerator kNONE_to_CMP0RR
Attach NONE to CMP0RR.

enumerator kPLL0_to_CMP1F
Attach PLL0 to CMP1F.

enumerator kFRO_HF_to_CMP1F
Attach FRO_HF to CMP1F.

enumerator kFRO12M_to_CMP1F
Attach FRO_12M to CMP1F.

enumerator kCLK_IN_to_CMP1F
Attach Clk_in to CMP1F.

enumerator kPLL1_CLK0_to_CMP1F
Attach PLL1_clk0 to CMP1F.

enumerator kUSB_PLL_to_CMP1F
Attach USB PLL to CMP1F.

enumerator kNONE_to_CMP1F
Attach NONE to CMP1F.

enumerator kPLL0_to_CMP1RR
Attach PLL0 to CMP1RR.

enumerator kFRO_HF_to_CMP1RR
Attach FRO_HF to CMP1RR.

enumerator kFRO12M_to_CMP1RR
Attach FRO_12M to CMP1RR.

enumerator kCLK_IN_to_CMP1RR
Attach Clk_in to CMP1RR.

enumerator kPLL1_CLK0_to_CMP1RR
Attach PLL1_clk0 to CMP1RR.

enumerator kUSB_PLL_to_CMP1RR
Attach USB PLL to CMP1RR.

enumerator kNONE_to_CMP1RR
Attach NONE to CMP1RR.

enumerator kPLL0_to_CMP2F
Attach PLL0 to CMP2F.

enumerator kFRO_HF_to_CMP2F
Attach FRO_HF to CMP2F.

enumerator kFRO12M_to_CMP2F
Attach FRO_12M to CMP2F.

enumerator kCLK_IN_to_CMP2F
Attach Clk_in to CMP2F.

enumerator kPLL1_CLK0_to_CMP2F
Attach PLL1_clk0 to CMP2F.

enumerator kUSB_PLL_to_CMP2F
Attach USB PLL to CMP2F.

enumerator kNONE_to_CMP2F
Attach NONE to CMP2F.

enumerator kPLL0_to_CMP2RR
Attach PLL0 to CMP2RR.

enumerator kFRO_HF_to_CMP2RR
Attach FRO_HF to CMP2RR.

enumerator kFRO12M_to_CMP2RR
Attach FRO_12M to CMP2RR.

enumerator kCLK_IN_to_CMP2RR
Attach Clk_in to CMP2RR.

enumerator kPLL1_CLK0_to_CMP2RR
Attach PLL1_clk0 to CMP2RR.

enumerator kUSB_PLL_to_CMP2RR
Attach USB PLL to CMP2RR.

enumerator kNONE_to_CMP2RR
Attach NONE to CMP2RR.

enumerator kPLL0_to_SAI0
Attach PLL0 to SAI0.

enumerator kCLK_IN_to_SAI0
Attach Clk_in to SAI0.

enumerator kFRO_HF_to_SAI0
Attach FRO_HF to SAI0.

enumerator kPLL1_CLK0_to_SAI0
Attach PLL1_clk0 to SAI0.

enumerator kUSB_PLL_to_SAI0
Attach USB PLL to SAI0.

enumerator kNONE_to_SAI0
Attach NONE to SAI0.

enumerator kPLL0_to_SAI1
Attach PLL0 to SAI1.

enumerator kCLK_IN_to_SAI1
Attach Clk_in to SAI1.

enumerator kFRO_HF_to_SAI1
Attach FRO_HF to SAI1.

enumerator kPLL1_CLK0_to_SAI1
Attach PLL1_clk0 to SAI1.

enumerator kUSB_PLL_to_SAI1
Attach USB PLL to SAI1.

enumerator kNONE_to_SAI1
Attach NONE to SAI1.

enumerator kPLL0_to_EMVSIM0
Attach PLL0 to EMVSIM0.

enumerator kCLK_IN_to_EMVSIM0
Attach Clk_in to EMVSIM0.

enumerator kFRO_HF_to_EMVSIM0
Attach FRO_HF to EMVSIM0.

enumerator kFRO12M_to_EMVSIM0
Attach FRO_12M to EMVSIM0.

enumerator kPLL1_CLK0_to_EMVSIM0
Attach PLL1_clk0 to EMVSIM0.

enumerator kNONE_to_EMVSIM0
Attach NONE to EMVSIM0.

enumerator kPLL0_to_EMVSIM1
Attach PLL0 to EMVSIM1.

enumerator kCLK_IN_to_EMVSIM1
Attach Clk_in to EMVSIM1.

enumerator kFRO_HF_to_EMVSIM1
Attach FRO_HF to EMVSIM1.

enumerator kFRO12M_to_EMVSIM1
Attach FRO_12M to EMVSIM1.

enumerator kPLL1_CLK0_to_EMVSIM1
Attach PLL1_clk0 to EMVSIM1.

enumerator kNONE_to_EMVSIM1
Attach NONE to EMVSIM1.

enumerator kPLL0_to_I3C1FCLK
Attach PLL0 to I3C1FCLK.

enumerator kCLK_IN_to_I3C1FCLK
Attach Clk_in to I3C1FCLK.

enumerator kFRO_HF_to_I3C1FCLK
Attach FRO_HF to I3C1FCLK.

enumerator kPLL1_CLK0_to_I3C1FCLK
Attach PLL1_clk0 to I3C1FCLK.

enumerator kUSB_PLL_to_I3C1FCLK
Attach USB PLL to I3C1FCLK.

enumerator kNONE_to_I3C1FCLK
Attach NONE to I3C1FCLK.

enumerator kI3C1FCLK_to_I3C1FCLKSTC
Attach I3C1FCLK to I3C1FCLKSTC.

enumerator kCLK_1M_to_I3C1FCLKSTC
Attach CLK_1M to I3C1FCLKSTC.

enumerator kNONE_to_I3C1FCLKSTC
Attach NONE to I3C1FCLKSTC.

enumerator kCLK_1M_to_I3C1FCLKS
Attach CLK_1M to I3C1FCLKS.

enumerator kNONE_to_I3C1FCLKS
Attach NONE to I3C1FCLKS.

enumerator kNONE_to_NONE
Attach NONE to NONE.

enum _clock_div_name

Clock dividers.

Values:

enumerator kCLOCK_DivSystickClk0
Systick Clk0 Divider.

enumerator kCLOCK_DivSystickClk1
Systick Clk1 Divider.

enumerator kCLOCK_DivTraceClk
Trace Clk Divider.

enumerator kCLOCK_DivSlowClk
SLOW CLK Divider.

enumerator kCLOCK_DivTsiClk
Tsi Clk Divider.

enumerator kCLOCK_DivAhbClk
Ahb Clk Divider.

enumerator kCLOCK_DivClkOut
ClkOut Clk Divider.

enumerator kCLOCK_DivFrohfClk
Frohf Clk Divider.

enumerator kCLOCK_DivWdt0Clk
Wdt0 Clk Divider.

enumerator kCLOCK_DivAdc0Clk
Adc0 Clk Divider.

enumerator kCLOCK_DivUsb0Clk
Usb0 Clk Divider.

enumerator kCLOCK_DivSctClk
Sct Clk Divider.

enumerator kCLOCK_DivPllClk
Pll Clk Divider.

enumerator kCLOCK_DivCtimer0Clk
Ctimer0 Clk Divider.

enumerator kCLOCK_DivCtimer1Clk
Ctimer1 Clk Divider.

enumerator kCLOCK_DivCtimer2Clk
Ctimer2 Clk Divider.

enumerator kCLOCK_DivCtimer3Clk
Ctimer3 Clk Divider.

enumerator kCLOCK_DivCtimer4Clk
Ctimer4 Clk Divider.

enumerator kCLOCK_DivPLL1Clk0
PLL1 Clk0 Divider.

enumerator kCLOCK_DivPLL1Clk1
Pll1 Clk1 Divider.

enumerator kCLOCK_DivAdc1Clk
Adc1 Clk Divider.

enumerator kCLOCK_DivDac0Clk
Dac0 Clk Divider.

enumerator kCLOCK_DivDac1Clk
Dac1 Clk Divider.

enumerator kCLOCK_DivDac2Clk
Dac2 Clk Divider.

enumerator kCLOCK_DivFlexspiClk
Flexspi Clk Divider.

enumerator kCLOCK_DivI3c0FClkStc
I3C0 FCLK STC Divider.

enumerator kCLOCK_DivI3c0FClkS
I3C0 FCLK S Divider.

enumerator kCLOCK_DivI3c0FClk
I3C0 FClk Divider.

enumerator kCLOCK_DivMicfilFClk
MICFILFCLK Divider.

enumerator kCLOCK_DivEspiClk
Espi Clk Divider.

enumerator kCLOCK_DivUSdhcClk
USdhc Clk Divider.

enumerator kCLOCK_DivFlexioClk
Flexio Clk Divider.

enumerator kCLOCK_DivFlexcan0Clk
Flexcan0 Clk Divider.

enumerator kCLOCK_DivFlexcan1Clk
Flexcan1 Clk Divider.

enumerator kCLOCK_DivEnetrmiiClk
Enetrmii Clk Divider.

enumerator kCLOCK_DivEnetptprefClk
Enetptpref Clk Divider.

enumerator kCLOCK_DivWdt1Clk
Wdt1 Clk Divider.

enumerator kCLOCK_DivCmp0FClk
Cmp0 FClk Divider.

enumerator kCLOCK_DivCmp0rrClk
Cmp0rr Clk Divider.

enumerator kCLOCK_DivCmp1FClk
Cmp1 FClk Divider.

enumerator kCLOCK_DivCmp1rrClk
Cmp1rr Clk Divider.

enumerator kCLOCK_DivCmp2FClk
Cmp2 FClk Divider.

enumerator kCLOCK_DivCmp2rrClk
Cmp2rr Clk Divider.

enumerator kCLOCK_DivFlexcom0Clk
Flexcom0 Clk Divider.

enumerator kCLOCK_DivFlexcom1Clk
Flexcom1 Clk Divider.

enumerator kCLOCK_DivFlexcom2Clk
Flexcom2 Clk Divider.

enumerator kCLOCK_DivFlexcom3Clk
Flexcom3 Clk Divider.

enumerator kCLOCK_DivFlexcom4Clk
Flexcom4 Clk Divider.

enumerator kCLOCK_DivFlexcom5Clk
Flexcom5 Clk Divider.

enumerator kCLOCK_DivFlexcom6Clk
Flexcom6 Clk Divider.

enumerator kCLOCK_DivFlexcom7Clk
Flexcom7 Clk Divider.

enumerator kCLOCK_DivFlexcom8Clk
Flexcom8 Clk Divider.

enumerator kCLOCK_DivFlexcom9Clk
Flexcom9 Clk Divider.

enumerator kCLOCK_DivSai0Clk
Sai0 Clk Divider.

enumerator kCLOCK_DivSai1Clk
Sai1 Clk Divider.

enumerator kCLOCK_DivEmvsim0Clk
Emvsim0 Clk Divider.

enumerator kCLOCK_DivEmvsim1Clk
Emvsim1 Clk Divider.

enumerator kCLOCK_DivI3c1FClkStc
I3C1 FCLK STC Divider.

enumerator kCLOCK_DivI3c1FClkS
I3C1 FCLK S Divider.

enumerator kCLOCK_DivI3c1FClk
I3C1 FClk Divider.

enum _osc32k_clk_gate_id
OSC32K clock gate.

Values:

enumerator kCLOCK_Osc32kToVbat
OSC32K[0] to VBAT domain.

enumerator kCLOCK_Osc32kToVsys
OSC32K[1] to VSYS domain.

enumerator kCLOCK_Osc32kToWake
OSC32K[2] to WAKE domain.

enumerator kCLOCK_Osc32kToMain
OSC32K[3] to MAIN domain.

enumerator kCLOCK_Osc32kToAll
OSC32K to VBAT,VSYS,WAKE,MAIN domain.

enum _clk16k_clk_gate_id
CLK16K clock gate.

Values:

enumerator kCLOCK_Clk16KToVbat
Clk16k[0] to VBAT domain.

enumerator kCLOCK_Clk16KToVsys
Clk16k[1] to VSYS domain.

enumerator kCLOCK_Clk16KToWake
Clk16k[2] to WAKE domain.

enumerator kCLOCK_Clk16KToMain
Clk16k[3] to MAIN domain.

enumerator kCLOCK_Clk16KToAll
Clk16k to VBAT,VSYS,WAKE,MAIN domain.

enum _clock_ctrl_enable
system clocks enable controls

Values:

enumerator kCLOCK_PLU_DEGLITCH_CLK_ENA
Enables clocks FRO_1MHz and FRO_12MHz for PLU deglitching.

enumerator kCLOCK_FRO1MHZ_CLK_ENA
Enables FRO_1MHz clock for clock muxing in clock gen.

enumerator kCLOCK_CLKIN_ENA
Enables clk_in clock for MICD, EMVSIM0/1, CAN0/1, I3C0/1, SAI0/1, SINC Filter (SINC), TSI, USBFS, SCT, uSDHC, clkout..

enumerator kCLOCK_FRO_HF_ENA
Enables FRO HF clock for the Frequency Measure module.

enumerator kCLOCK_FRO12MHZ_ENA
Enables the FRO_12MHz clock for the Flash, LPTIMER0/1, and Frequency Measurement modules.

enumerator kCLOCK_FRO1MHZ_ENA
Enables the FRO_1MHz clock for RTC module and for UTICK.

enumerator kCLOCK_CLKIN_ENA_FM_USBH_LPT
Enables the clk_in clock for the Frequency Measurement, USB HS and LPTIMER0/1 modules.

enum _clock_usb_phy_src

Source of the USB HS PHY.

Values:

enumerator kCLOCK_Usbphy480M

Use 480M.

enum _scg_status

SCG status return codes.

Values:

enumerator kStatus_SCG_Busy

Clock is busy.

enumerator kStatus_SCG_InvalidSrc

Invalid source.

enum _firc_trim_mode

firc trim mode.

Values:

enumerator kSCG_FircTrimNonUpdate

Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure trim_config_t.

enumerator kSCG_FircTrimUpdate

Trim enable and trim value update enable. In this mode, the trim value is auto update.

enum _firc_trim_src

firc trim source.

Values:

enumerator kSCG_FircTrimSrcUsb0

USB0 start of frame (1kHz).

enumerator kSCG_FircTrimSrcSysOsc

System OSC.

enumerator kSCG_FircTrimSrcRtcOsc

RTC OSC (32.768 kHz).

enum _sirc_trim_mode

sirc trim mode.

Values:

enumerator kSCG_SircTrimNonUpdate

Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure trim_config_t.

enumerator kSCG_SircTrimUpdate

Trim enable and trim value update enable. In this mode, the trim value is auto update.

enum _sirc_trim_src

sirc trim source.

Values:

enumerator kSCG_SircTrimSrcSysOsc
System OSC.

enumerator kSCG_SircTrimSrcRtcOsc
RTC OSC (32.768 kHz).

enum _scg_sosc_monitor_mode
SCG system OSC monitor mode.

Values:

enumerator kSCG_SysOscMonitorDisable
Monitor disabled.

enumerator kSCG_SysOscMonitorInt
Interrupt when the SOSC error is detected.

enumerator kSCG_SysOscMonitorReset
Reset when the SOSC error is detected.

enum _scg_rosc_monitor_mode
SCG ROSC monitor mode.

Values:

enumerator kSCG_RoscMonitorDisable
Monitor disabled.

enumerator kSCG_RoscMonitorInt
Interrupt when the RTC OSC error is detected.

enumerator kSCG_RoscMonitorReset
Reset when the RTC OSC error is detected.

enum _scg_upll_monitor_mode
SCG UPLL monitor mode.

Values:

enumerator kSCG_UpllMonitorDisable
Monitor disabled.

enumerator kSCG_UpllMonitorInt
Interrupt when the UPLL error is detected.

enumerator kSCG_UpllMonitorReset
Reset when the UPLL error is detected.

enum _scg_pll0_monitor_mode
SCG PLL0 monitor mode.

Values:

enumerator kSCG_Pll0MonitorDisable
Monitor disabled.

enumerator kSCG_Pll0MonitorInt
Interrupt when the PLL0 Clock error is detected.

enumerator kSCG_Pll0MonitorReset
Reset when the PLL0 Clock error is detected.

enum _scg_pll1_monitor_mode

SCG PLL1 monitor mode.

Values:

enumerator kSCG_Pll1MonitorDisable

Monitor disabled.

enumerator kSCG_Pll1MonitorInt

Interrupt when the PLL1 Clock error is detected.

enumerator kSCG_Pll1MonitorReset

Reset when the PLL1 Clock error is detected.

enum _vbat_osc_xtal_cap

The enumerator of internal capacitance of OSC's XTAL pin.

Values:

enumerator kVBAT_OscXtal0pFCap

The internal capacitance for XTAL pin is 0pF.

enumerator kVBAT_OscXtal2pFCap

The internal capacitance for XTAL pin is 2pF.

enumerator kVBAT_OscXtal4pFCap

The internal capacitance for XTAL pin is 4pF.

enumerator kVBAT_OscXtal6pFCap

The internal capacitance for XTAL pin is 6pF.

enumerator kVBAT_OscXtal8pFCap

The internal capacitance for XTAL pin is 8pF.

enumerator kVBAT_OscXtal10pFCap

The internal capacitance for XTAL pin is 10pF.

enumerator kVBAT_OscXtal12pFCap

The internal capacitance for XTAL pin is 12pF.

enumerator kVBAT_OscXtal14pFCap

The internal capacitance for XTAL pin is 14pF.

enumerator kVBAT_OscXtal16pFCap

The internal capacitance for XTAL pin is 16pF.

enumerator kVBAT_OscXtal18pFCap

The internal capacitance for XTAL pin is 18pF.

enumerator kVBAT_OscXtal20pFCap

The internal capacitance for XTAL pin is 20pF.

enumerator kVBAT_OscXtal22pFCap

The internal capacitance for XTAL pin is 22pF.

enumerator kVBAT_OscXtal24pFCap

The internal capacitance for XTAL pin is 24pF.

enumerator kVBAT_OscXtal26pFCap

The internal capacitance for XTAL pin is 26pF.

enumerator kVBAT_OscXtal28pFCap

The internal capacitance for XTAL pin is 28pF.

enumerator kVBAT_OscXtal30pFCap
The internal capacitance for XTAL pin is 30pF.

enum _vbat_osc_extal_cap
The enumerator of internal capacitance of OSC's EXTAL pin.

Values:

enumerator kVBAT_OscExtal0pFCap
The internal capacitance for EXTAL pin is 0pF.

enumerator kVBAT_OscExtal2pFCap
The internal capacitance for EXTAL pin is 2pF.

enumerator kVBAT_OscExtal4pFCap
The internal capacitance for EXTAL pin is 4pF.

enumerator kVBAT_OscExtal6pFCap
The internal capacitance for EXTAL pin is 6pF.

enumerator kVBAT_OscExtal8pFCap
The internal capacitance for EXTAL pin is 8pF.

enumerator kVBAT_OscExtal10pFCap
The internal capacitance for EXTAL pin is 10pF.

enumerator kVBAT_OscExtal12pFCap
The internal capacitance for EXTAL pin is 12pF.

enumerator kVBAT_OscExtal14pFCap
The internal capacitance for EXTAL pin is 14pF.

enumerator kVBAT_OscExtal16pFCap
The internal capacitance for EXTAL pin is 16pF.

enumerator kVBAT_OscExtal18pFCap
The internal capacitance for EXTAL pin is 18pF.

enumerator kVBAT_OscExtal20pFCap
The internal capacitance for EXTAL pin is 20pF.

enumerator kVBAT_OscExtal22pFCap
The internal capacitance for EXTAL pin is 22pF.

enumerator kVBAT_OscExtal24pFCap
The internal capacitance for EXTAL pin is 24pF.

enumerator kVBAT_OscExtal26pFCap
The internal capacitance for EXTAL pin is 26pF.

enumerator kVBAT_OscExtal28pFCap
The internal capacitance for EXTAL pin is 28pF.

enumerator kVBAT_OscExtal30pFCap
The internal capacitance for EXTAL pin is 30pF.

enum _vbat_osc_fine_adjustment_value
The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by modifying the automatic gain control (AGC).

Values:

enumerator kVBAT_OscCoarseAdjustment05

enumerator kVBAT_OscCoarseAdjustment10

enumerator kVBAT_OscCoarseAdjustment18

enumerator kVBAT_OscCoarseAdjustment33

enum _run_mode

The active run mode (voltage level).

Values:

enumerator kMD_Mode

Midvoltage (1.0 V).

enumerator kSD_Mode

Normal voltage (1.1 V).

enumerator kOD_Mode

Overdrive voltage (1.2 V).

enum _vbat_osc_init_trim

The enumerator of Initialization Trim.

Values:

enumerator kVBAT_OscInitTrim8000ms

Configures the start-up time of the oscillator to 8s.

enumerator kVBAT_OscInitTrim4000ms

Configures the start-up time of the oscillator to 4s.

enumerator kVBAT_OscInitTrim2000ms

Configures the start-up time of the oscillator to 2s.

enumerator kVBAT_OscInitTrim1000ms

Configures the start-up time of the oscillator to 1s.

enumerator kVBAT_OscInitTrim500ms

Configures the start-up time of the oscillator to 0.5s.

enumerator kVBAT_OscInitTrim250ms

Configures the start-up time of the oscillator to 0.25s.

enumerator kVBAT_OscInitTrim125ms

Configures the start-up time of the oscillator to 0.125s.

enumerator kVBAT_OscInitTrimHalfms

Configures the start-up time of the oscillator to 0.5ms.

enum _vbat_osc_cap_trim

The enumerator of Capacitor Trim.

Values:

enumerator kVBAT_OscCapTrimDefault

enumerator kVBAT_OscCapTrim1us

enumerator kVBAT_OscCapTrim2us

enumerator kVBAT_OscCapTrim2andhalfus

enum _vbat_osc_dly_trim

The enumerator of Delay Trim.

Values:

enumerator kVBAT_OscDlyTrim0
P current 9(nA) and N Current 6(nA).

enumerator kVBAT_OscDlyTrim1
P current 13(nA) and N Current 6(nA).

enumerator kVBAT_OscDlyTrim3
P current 4(nA) and N Current 6(nA).

enumerator kVBAT_OscDlyTrim4
P current 9(nA) and N Current 4(nA).

enumerator kVBAT_OscDlyTrim5
P current 13(nA) and N Current 4(nA).

enumerator kVBAT_OscDlyTrim6
P current 4(nA) and N Current 4(nA).

enumerator kVBAT_OscDlyTrim7
P current 9(nA) and N Current 2(nA).

enumerator kVBAT_OscDlyTrim8
P current 13(nA) and N Current 2(nA).

enumerator kVBAT_OscDlyTrim9
P current 4(nA) and N Current 2(nA).

enum _vbat_osc_cap2_trim
The enumerator of CAP2_TRIM.

Values:

enumerator kVBAT_OscCap2Trim0

enumerator kVBAT_OscCap2Trim1

enum _vbat_osc_cmp_trim
The enumerator of Comparator Trim.

Values:

enumerator kVBAT_OscCmpTrim760mv

enumerator kVBAT_OscCmpTrim770mv

enumerator kVBAT_OscCmpTrim740mv

enum _vbat_osc_mode_en
The enumerator of configures Crystal Oscillator mode..

Values:

enumerator kVBAT_OscNormalModeEnable

enumerator kVBAT_OscStartupModeEnable

enumerator kVBAT_OscLowpowerModeEnable

enum _pll_clk_src
PLL clock source.

Values:

enumerator kPll_ClkSrcSysOsc
System OSC.

enumerator kPll_ClkSrcFirc
Fast IRC.

enumerator kPll_ClkSrcRosc
RTC OSC.

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

Values:

enumerator kSS_MF_512
Nss = 512 (fm ~= 3.9 - 7.8 kHz)

enumerator kSS_MF_384
Nss ~= 384 (fm ~= 5.2 - 10.4 kHz)

enumerator kSS_MF_256
Nss = 256 (fm ~= 7.8 - 15.6 kHz)

enumerator kSS_MF_128
Nss = 128 (fm ~= 15.6 - 31.3 kHz)

enumerator kSS_MF_64
Nss = 64 (fm ~= 32.3 - 64.5 kHz)

enumerator kSS_MF_32
Nss = 32 (fm ~= 62.5 - 125 kHz)

enumerator kSS_MF_24
Nss ~= 24 (fm ~= 83.3 - 166.6 kHz)

enumerator kSS_MF_16
Nss = 16 (fm ~= 125 - 250 kHz)

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

Values:

enumerator kSS_MR_K0
k = 0 (no spread spectrum)

enumerator kSS_MR_K1
k ~= 1

enumerator kSS_MR_K1_5
k ~= 1.5

enumerator kSS_MR_K2
k ~= 2

enumerator kSS_MR_K3
k ~= 3

enumerator kSS_MR_K4
k ~= 4

enumerator kSS_MR_K6
k ~= 6

enumerator kSS_MR_K8
k ~= 8

enum _ss_modwvctrl

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

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

Values:

enumerator kSS_MC_NOC
no compensation

enumerator kSS_MC_RECC
recommended setting

enumerator kSS_MC_MAXC
max. compensation

enum _pll_error

PLL status definitions.

Values:

enumerator kStatus_PLL_Success
PLL operation was successful

enumerator kStatus_PLL_OutputTooLow
PLL output rate request was too low

enumerator kStatus_PLL_OutputTooHigh
PLL output rate request was too high

enumerator kStatus_PLL_OutputError
PLL output rate error

enumerator kStatus_PLL_InputTooLow
PLL input rate is too low

enumerator kStatus_PLL_InputTooHigh
PLL input rate is too high

enumerator kStatus_PLL_OutsideIntLimit
Requested output rate isn't possible

enumerator kStatus_PLL_CCOTooLow
Requested CCO rate isn't possible

enumerator kStatus_PLL_CCOTooHigh
Requested CCO rate isn't possible

typedef enum *_clock_ip_name* clock_ip_name_t

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

typedef enum *_clock_name* clock_name_t

Clock name used to get clock frequency.

typedef enum *_clock_attach_id* clock_attach_id_t

The enumerator of clock attach Id.

typedef enum *_clock_div_name* clock_div_name_t
Clock dividers.

typedef enum *_osc32k_clk_gate_id* osc32k_clk_gate_id_t
OSC32K clock gate.

typedef enum *_clk16k_clk_gate_id* clk16k_clk_gate_id_t
CLK16K clock gate.

typedef enum *_clock_ctrl_enable* clock_ctrl_enable_t
system clocks enable controls

typedef enum *_clock_usb_phy_src* clock_usb_phy_src_t
Source of the USB HS PHY.

typedef enum *_firc_trim_mode* firc_trim_mode_t
firc trim mode.

typedef enum *_firc_trim_src* firc_trim_src_t
firc trim source.

typedef struct *_firc_trim_config* firc_trim_config_t
firc trim configuration.

typedef enum *_sirc_trim_mode* sirc_trim_mode_t
sirc trim mode.

typedef enum *_sirc_trim_src* sirc_trim_src_t
sirc trim source.

typedef struct *_sirc_trim_config* sirc_trim_config_t
sirc trim configuration.

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

typedef enum *_scg_rosc_monitor_mode* scg_rosc_monitor_mode_t
SCG ROSC monitor mode.

typedef enum *_scg_upll_monitor_mode* scg_upll_monitor_mode_t
SCG UPLL monitor mode.

typedef enum *_scg_pll0_monitor_mode* scg_pll0_monitor_mode_t
SCG PLL0 monitor mode.

typedef enum *_scg_pll1_monitor_mode* scg_pll1_monitor_mode_t
SCG PLL1 monitor mode.

typedef enum *_vbat_osc_xtal_cap* vbat_osc_xtal_cap_t
The enumerator of internal capacitance of OSC's XTAL pin.

typedef enum *_vbat_osc_extal_cap* vbat_osc_extal_cap_t
The enumerator of internal capacitance of OSC's EXTAL pin.

typedef enum *_vbat_osc_fine_adjustment_value* vbat_osc_coarse_adjustment_value_t
The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by modifying the automatic gain control (AGC).

typedef struct *_vbat_osc_config* vbat_osc_config_t
The structure of oscillator configuration.

typedef enum *_run_mode* run_mode_t
The active run mode (voltage level).

typedef enum *_vbat_osc_init_trim* vbat_osc_init_trim_t
The enumerator of Initialization Trim.

typedef enum *_vbat_osc_cap_trim* vbat_osc_cap_trim_t
The enumerator of Capacitor Trim.

typedef enum *_vbat_osc_dly_trim* vbat_osc_dly_trim_t
The enumerator of Delay Trim.

typedef enum *_vbat_osc_cap2_trim* vbat_osc_cap2_trim_t
The enumerator of CAP2_TRIM.

typedef enum *_vbat_osc_cmp_trim* vbat_osc_cmp_trim_t
The enumerator of Comparator Trim.

typedef enum *_vbat_osc_mode_en* vbat_osc_mode_en_t
The enumerator of configures Crystal Oscillator mode..

typedef struct *_osc_32k_config* osc_32k_config_t
The structure of oscillator configuration.

typedef enum *_pll_clk_src* pll_clk_src_t
PLL clock source.

typedef enum *_ss_progmodfm* ss_progmodfm_t
PLL Spread Spectrum (SS) Programmable modulation frequency See (MF) field in the PLL0SSCG1 register in the UM.

typedef enum *_ss_progmoddp* ss_progmoddp_t
PLL Spread Spectrum (SS) Programmable frequency modulation depth See (MR) field in the PLL0SSCG1 register in the UM.

typedef enum *_ss_modwvctrl* ss_modwvctrl_t

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

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

typedef struct *_pll_config* pll_config_t
PLL configuration structure.

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

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

typedef enum *_pll_error* pll_error_t
PLL status definitions.

static inline void CLOCK_EnableClock(*clock_ip_name_t* clk)
Enable the clock for specific IP.

Parameters

- clk – : Clock to be enabled.

Returns
Nothing

static inline void CLOCK_DisableClock(*clock_ip_name_t* clk)

Disable the clock for specific IP.

Parameters

- clk – : Clock to be Disabled.

Returns

Nothing

status_t CLOCK_SetupFROHFClocking(uint32_t iFreq)

Initialize the Core clock to given frequency (48 or 144 MHz). This function turns on FIRC and select the given frequency as the source of fro_hf.

Parameters

- iFreq – : Desired frequency (must be one of CLK_FRO_44MHZ or CLK_FRO_144MHZ)

Returns

returns success or fail status.

status_t CLOCK_SetupExtClocking(uint32_t iFreq)

Initialize the external osc clock to given frequency.

Parameters

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

Returns

returns success or fail status.

status_t CLOCK_SetupExtRefClocking(uint32_t iFreq)

Initialize the external reference clock to given frequency.

Parameters

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

Returns

returns success or fail status.

status_t CLOCK_SetupOsc32KClocking(uint32_t id)

Initialize the XTAL32/EXTAL32 input clock to given frequency.

Parameters

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

Returns

returns success or fail status.

void CLOCK_GetDefaultOsc32KConfig(*osc_32k_config_t* *config)

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

Parameters

- config – Pointer to a configuration structure

status_t CLOCK_SetupOsc32KClockingConfig(*osc_32k_config_t* config)

Initialize the OSC 32K with user-defined settings.

Parameters

- config – : OSC 32K configuration structure

Returns

returns success or fail status.

status_t CLOCK_SetupClk16KClocking(uint32_t id)

Initialize the FRO16K input clock to given frequency.

Parameters

- id – : FRO 16 kHz output clock to specified modules, it should use clk16k_clk_gate_id_t value

Returns

returns success or fail status.

status_t CLOCK_FROHFTrimConfig(*firc_trim_config_t* config)

Setup FROHF trim.

Parameters

- config – : FROHF trim value

Returns

returns success or fail status.

status_t CLOCK_FRO12MTrimConfig(*sirc_trim_config_t* config)

Setup FRO 12M trim.

Parameters

- config – : FRO 12M trim value

Returns

returns success or fail status.

void CLOCK_SetSysOscMonitorMode(*scg_sosc_monitor_mode_t* mode)

Sets the system OSC monitor mode.

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

Parameters

- mode – Monitor mode to set.

void CLOCK_SetRoscMonitorMode(*scg_rosc_monitor_mode_t* mode)

Sets the ROSC monitor mode.

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

Parameters

- mode – Monitor mode to set.

void CLOCK_SetUpllMonitorMode(*scg_upll_monitor_mode_t* mode)

Sets the UPLL monitor mode.

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

Parameters

- mode – Monitor mode to set.

void CLOCK_SetPll0MonitorMode(*scg_pll0_monitor_mode_t* mode)

Sets the PLL0 monitor mode.

This function sets the PLL0 monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

void CLOCK_SetPll1MonitorMode(*scg_pll1_monitor_mode_t* mode)

Sets the PLL1 monitor mode.

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

Parameters

- mode – Monitor mode to set.

void VBAT_SetOscConfig(VBAT_Type *base, const *vbat_osc_config_t* *config)

Config 32k Crystal Oscillator.

Parameters

- base – VBAT peripheral base address.
- config – The pointer to the structure vbat_osc_config_t.

status_t CLOCK_SetFLASHAccessCyclesForFreq(uint32_t system_freq_hz, *run_mode_t* mode)

Set the additional number of wait-states added to account for the ratio of system clock period to flash access time during full speed power mode.

Parameters

- system_freq_hz – : Input frequency
- mode – : Active run mode (voltage level).

Returns

success or fail status

void CLOCK_AttachClk(*clock_attach_id_t* connection)

Configure the clock selection muxes.

Parameters

- connection – : Clock to be configured.

Returns

Nothing

clock_attach_id_t CLOCK_GetClockAttachId(*clock_attach_id_t* attachId)

Get the actual clock attach id. This fuction uses the offset in input attach id, then it reads the actual source value in the register and combine the offset to obtain an actual attach id.

Parameters

- attachId – : Clock attach id to get.

Returns

Clock source value.

void CLOCK_SetClkDiv(*clock_div_name_t* div_name, uint32_t divided_by_value)

Setup peripheral clock dividers.

Parameters

- div_name – : Clock divider name

- divided_by_value – Value to be divided

Returns
Nothing

uint32_t CLOCK_GetClkDiv(*clock_div_name_t* div_name)

Get peripheral clock dividers.

Parameters

- div_name – : Clock divider name

Returns
peripheral clock dividers

void CLOCK_HaltClkDiv(*clock_div_name_t* div_name)

Halt peripheral clock dividers.

Parameters

- div_name – : Clock divider name

Returns
Nothing

void CLOCK_SetupClockCtrl(uint32_t mask)

system clocks enable controls.

Parameters

- mask – : system clocks enable value, it should use clock_ctrl_enable_t value

Returns
Nothing

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Return Frequency of selected clock.

Returns
Frequency of selected clock

uint32_t CLOCK_GetMainClkFreq(void)

Return Frequency of main.

Returns
Frequency of the main

uint32_t CLOCK_GetCoreSysClkFreq(void)

Return Frequency of core.

Returns
Frequency of the core

uint32_t CLOCK_GetCTimerClkFreq(uint32_t id)

Return Frequency of CTimer functional Clock.

Returns
Frequency of CTimer functional Clock

uint32_t CLOCK_GetAdcClkFreq(uint32_t id)

Return Frequency of Adc Clock.

Returns
Frequency of Adc.

uint32_t CLOCK_GetUsb0ClkFreq(void)

Return Frequency of Usb Clock.

Returns
Frequency of Adc.

uint32_t CLOCK_GetLPFlexCommClkFreq(uint32_t id)

Return Frequency of LPFlexComm Clock.

Returns
Frequency of LPFlexComm Clock

uint32_t CLOCK_GetSctClkFreq(void)

Return Frequency of SCTimer Clock.

Returns
Frequency of SCTimer Clock.

uint32_t CLOCK_GetTsiClkFreq(void)

Return Frequency of TSI Clock.

Returns
Frequency of TSI Clock.

uint32_t CLOCK_GetSincFilterClkFreq(void)

Return Frequency of SINC FILTER Clock.

Returns
Frequency of SINC FILTER Clock.

uint32_t CLOCK_GetDacClkFreq(uint32_t id)

Return Frequency of DAC Clock.

Returns
Frequency of DAC Clock

uint32_t CLOCK_GetFlexspiClkFreq(void)

Return Frequency of FlexSPI.

Returns
Frequency of FlexSPI Clock

uint32_t CLOCK_GetPll0OutFreq(void)

Return Frequency of PLL.

Returns
Frequency of PLL

uint32_t CLOCK_GetPll1OutFreq(void)

Return Frequency of USB PLL.

Returns
Frequency of PLL

uint32_t CLOCK_GetPllClkDivFreq(void)

Return Frequency of PLLCLKDIV.

Returns
Frequency of PLLCLKDIV Clock

uint32_t CLOCK_GetI3cClkFreq(uint32_t id)

Return Frequency of I3C function Clock.

Returns
Frequency of I3C function Clock

uint32_t CLOCK_GetI3cSTCClkFreq(uint32_t id)

Return Frequency of I3C function slow TC Clock.

Returns

Frequency of I3C function slow TC Clock

uint32_t CLOCK_GetI3cSClkFreq(uint32_t id)

Return Frequency of I3C function slow Clock.

Returns

Frequency of I3C function slow Clock

uint32_t CLOCK_GetMicfilClkFreq(void)

Return Frequency of MICFIL Clock.

Returns

Frequency of MICFIL.

uint32_t CLOCK_GetUsdhcClkFreq(void)

Return Frequency of uSDHC.

Returns

Frequency of uSDHC Clock

uint32_t CLOCK_GetFlexioClkFreq(void)

Return Frequency of FLEXIO.

Returns

Frequency of FLEXIO Clock

uint32_t CLOCK_GetFlexcanClkFreq(uint32_t id)

Return Frequency of FLEXCAN.

Returns

Frequency of FLEXCAN Clock

uint32_t CLOCK_GetEnetRmiiClkFreq(void)

Return Frequency of Ethernet RMII Clock.

Returns

Frequency of Ethernet RMII.

uint32_t CLOCK_GetEnetPtpRefClkFreq(void)

Return Frequency of Ethernet PTP REF Clock.

Returns

Frequency of Ethernet PTP REF.

void CLOCK_SetupEnetTxClk(uint32_t iFreq)

Initialize the ENET TX CLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns

Nothing

uint32_t CLOCK_GetEnetTxClkFreq(void)

Return Frequency of ENET TX CLK.

Returns

Frequency of ENET TX CLK

uint32_t CLOCK_GetEwm0ClkFreq(void)

Return Frequency of EWM0 Clock.

Returns
Frequency of EWM0.

uint32_t CLOCK_GetWdtClkFreq(uint32_t id)

Return Frequency of Watchdog.

Returns
Frequency of Watchdog

uint32_t CLOCK_GetOstimerClkFreq(void)

Return Frequency of OSTIMER.

Returns
Frequency of OSTIMER Clock

uint32_t CLOCK_GetCmpFClkFreq(uint32_t id)

Return Frequency of CMP Function Clock.

Returns
Frequency of CMP Function.

uint32_t CLOCK_GetCmpRRClkFreq(uint32_t id)

Return Frequency of CMP Round Robin Clock.

Returns
Frequency of CMP Round Robin.

uint32_t CLOCK_GetSaiClkFreq(uint32_t id)

Return Frequency of SAI Clock.

Returns
Frequency of SAI Clock.

void CLOCK_SetupSaiMclk(uint32_t id, uint32_t iFreq)

Initialize the SAI MCLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns
Nothing

void CLOCK_SetupSaiTxBclk(uint32_t id, uint32_t iFreq)

Initialize the SAI TX BCLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns
Nothing

void CLOCK_SetupSaiRxBclk(uint32_t id, uint32_t iFreq)

Initialize the SAI RX BCLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns
Nothing

uint32_t CLOCK_GetSaiMclkFreq(uint32_t id)

Return Frequency of SAI MCLK.

Returns
Frequency of SAI MCLK

uint32_t CLOCK_GetSaiTxBclkFreq(uint32_t id)

Return Frequency of SAI TX BCLK.

Returns
Frequency of SAI TX BCLK

uint32_t CLOCK_GetSaiRxBclkFreq(uint32_t id)

Return Frequency of SAI RX BCLK.

Returns
Frequency of SAI RX BCLK

uint32_t CLOCK_GetEmvsimClkFreq(uint32_t id)

Return Frequency of EMVSIM Clock.

Returns
Frequency of EMVSIM Clock.

uint32_t CLOCK_GetPLL0InClockRate(void)

Return PLL0 input clock rate.

Returns
PLL0 input clock rate

uint32_t CLOCK_GetPLL1InClockRate(void)

Return PLL1 input clock rate.

Returns
PLL1 input clock rate

uint32_t CLOCK_GetExtUpllFreq(void)

Gets the external UPLL frequency.

Returns
The frequency of the external UPLL.

void CLOCK_SetExtUpllFreq(uint32_t freq)

Sets the external UPLL frequency.

Parameters

- The – frequency of external UPLL.

___STATIC_INLINE bool CLOCK_IsPLL0Locked (void)

Check if PLL is locked or not.

Returns
true if the PLL is locked, false if not locked

___STATIC_INLINE bool CLOCK_IsPLL1Locked (void)

Check if PLL1 is locked or not.

Returns
true if the PLL1 is locked, false if not locked

uint32_t CLOCK_GetPLLOutFromSetup(*pll_setup_t* *pSetup)

Return PLL0 output clock rate from setup structure.

Parameters

- pSetup – : Pointer to a PLL setup structure

Returns
System PLL output clock rate the setup structure will generate

pll_error_t CLOCK_SetupPLLData(*pll_config_t* *pControl, *pll_setup_t* *pSetup)

Set PLL output based on the passed PLL setup data.

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

Parameters

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

Returns
PLL_ERROR_SUCCESS on success, or PLL setup error code

pll_error_t CLOCK_SetPLL0Freq(const *pll_setup_t* *pSetup)

Set PLL output from PLL setup structure (precise frequency)

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

Parameters

- pSetup – : Pointer to populated PLL setup structure

Returns
kStatus_PLL_Success on success, or PLL setup error code

pll_error_t CLOCK_SetPLL1Freq(const *pll_setup_t* *pSetup)

Set PLL output from PLL setup structure (precise frequency)

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

Parameters

- pSetup – : Pointer to populated PLL setup structure

Returns
kStatus_PLL_Success on success, or PLL setup error code

void CLOCK_EnableOstimer32kClock(void)

Enable the OSTIMER 32k clock.

Returns
Nothing

bool CLOCK_EnableUsbfsClock(void)

brief Enable USB FS clock. Enable USB Full Speed clock.

bool CLOCK_EnableUsbhsPhyPllClock(*clock_usb_phy_src_t* src, uint32_t freq)

brief Enable USB HS PHY PLL clock.

This function enables the internal 480MHz USB PHY PLL clock.

param src USB HS PHY PLL clock source. param freq The frequency specified by src. retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

void CLOCK_DisableUsbhsPhyPllClock(void)

brief Disable USB HS PHY PLL clock.

This function disables USB HS PHY PLL clock.

bool CLOCK_EnableUsbhsClock(void)

brief Enable USB HS clock. retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

status_t CLOCK_FIRCAutoTrimWithSOF(void)

FIRC Auto Trim With SOF.

Returns

returns success or fail status.

static inline void CLOCK_EnableCpu1Clock(SYSCON_Type *base, bool enable)

Enable/disable the CPU1 clock.

Parameters

- enable – True to enable the clock, false to disable the clock.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.0.1.

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL

Configure whether driver controls clock.

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note: All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

CLOCK_USR_CFG_PLL_CONFIG_CACHE_COUNT

User-defined the size of cache for CLOCK_PllGetConfig() function.

Once define this MACRO to be non-zero value, CLOCK_PllGetConfig() function would cache the recent calulation and accelerate the execution to get the right settings.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

ROM_CLOCKS

Clock ip name array for ROM.

SRAM_CLOCKS

Clock ip name array for SRAM.

FMC_CLOCKS

Clock ip name array for FMC.

INPUTMUX_CLOCKS

Clock ip name array for INPUTMUX.

ETH_CLOCKS
: Clock ip name array for ENET.

GPIO_CLOCKS
: Clock ip name array for GPIO.

GDET_CLOCKS
: Clock ip name array for GDET.

PINT_CLOCKS
: Clock ip name array for PINT.

DMA_CLOCKS
: Clock ip name array for DMA.

EDMA_CLOCKS
: Clock gate name array for EDMA.

CRC_CLOCKS
: Clock ip name array for CRC.

WWDT_CLOCKS
: Clock ip name array for WWDT.

MAILBOX_CLOCKS
: Clock ip name array for Mailbox.

LPADC_CLOCKS
: Clock ip name array for LPADC.

MRT_CLOCKS
: Clock ip name array for MRT.

OSTIMER_CLOCKS
: Clock ip name array for OSTIMER.

SCT_CLOCKS
: Clock ip name array for SCT0.

UTICK_CLOCKS
: Clock ip name array for UTICK.

LP_FLEXCOMM_CLOCKS
: Clock ip name array for LP_FLEXCOMM.

LPUART_CLOCKS
: Clock ip name array for LPUART.

LPI2C_CLOCKS
: Clock ip name array for LPI2C.

LPSPI_CLOCKS
: Clock ip name array for LSPI.

CTIMER_CLOCKS
: Clock ip name array for CTIMER.

FREQME_CLOCKS
: Clock ip name array for FREQME.

POWERQUAD_CLOCKS
: Clock ip name array for PowerQuad.

PLU_CLOCKS
: Clock ip name array for PLU.

PUF_CLOCKS
: Clock ip name array for PUF.

VREF_CLOCKS
: Clock ip name array for VREF.

LPDAC_CLOCKS
: Clock ip name array for LPDAC.

HPDAC_CLOCKS
: Clock ip name array for HPDAC.

PWM_CLOCKS
: Clock ip name array for PWM.

QDC_CLOCKS
: Clock ip name array for QDC.

FLEXIO_CLOCKS
: Clock ip name array for FLEXIO.

FLEXCAN_CLOCKS
: Clock ip name array for FLEXCAN.

EMVSIM_CLOCKS
: Clock ip name array for EMVSIM.

I3C_CLOCKS
: Clock ip name array for I3C.

USDHC_CLOCKS
: Clock ip name array for USDHC.

FLEXSPI_CLOCKS
: Clock ip name array for FLEXSPI.

SAI_CLOCKS
: Clock ip name array for SAI.

RTC_CLOCKS
: Clock ip name array for RTC.

PDM_CLOCKS
: Clock ip name array for PDM.

ERM_CLOCKS
: Clock ip name array for ERM.

EIM_CLOCKS
: Clock ip name array for EIM.

OPAMP_CLOCKS
: Clock ip name array for OPAMP.

TSI_CLOCKS
: Clock ip name array for TSI.

TRNG_CLOCKS
: Clock ip name array for TRNG.

LPCMP_CLOCKS
: Clock ip name array for LPCMP.

SINC_CLOCKS
: Clock ip name array for SINC.

SEMA42_CLOCKS
: Clock ip name array for SEMA42.

CLK_GATE_REG_OFFSET_SHIFT
: Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

CLK_GATE_REG_OFFSET_MASK

CLK_GATE_BIT_SHIFT_SHIFT

CLK_GATE_BIT_SHIFT_MASK

CLK_GATE_DEFINE(reg_offset, bit_shift)

CLK_GATE_ABSTRACT_REG_OFFSET(x)

CLK_GATE_ABSTRACT_BITS_SHIFT(x)

AHB_CLK_CTRL0

AHB_CLK_CTRL1

AHB_CLK_CTRL2

AHB_CLK_CTRL3

REG_PWM0SUBCTL

REG_PWM1SUBCTL

BUS_CLK
: Peripherals clock source definition.

I2C0_CLK_SRC

CLK_ATTACH_ID(mux, sel, pos)
: Clock Mux Switches The encoding is as follows each connection identified is 32bits wide while 24bits are valuable starting from LSB upwards.

  [4 bits for choice, 0 means invalid choice] [8 bits mux ID]*

MUX_A(mux, sel)

MUX_B(mux, sel, selector)

GET_ID_ITEM(connection)

GET_ID_NEXT_ITEM(connection)

GET_ID_ITEM_MUX(connection)

GET_ID_ITEM_SEL(connection)

GET_ID_SELECTOR(connection)

CM_SYSTICKCLKSEL0

CM_SYSTICKCLKSEL1

CM_TRACECLKSEL

CM_CTIMERCLKSEL0

CM_CTIMERCLKSEL1

CM_CTIMERCLKSEL2

CM_CTIMERCLKSEL3

CM_CTIMERCLKSEL4

CM_CLKOUTCLKSEL

CM_ADC0CLKSEL

CM_USB0CLKSEL

CM_FCCLKSEL0

CM_FCCLKSEL1

CM_FCCLKSEL2

CM_FCCLKSEL3

CM_FCCLKSEL4

CM_FCCLKSEL5

CM_FCCLKSEL6

CM_FCCLKSEL7

CM_FCCLKSEL8

CM_FCCLKSEL9

CM_SCTCLKSEL

CM_TSICLKSEL

CM_SINCFILTCLKSEL

CM_ADC1CLKSEL

CM_DAC0CLKSEL

CM_DAC1CLKSEL

CM_DAC2CLKSEL

CM_FLEXSPICLKSEL

CM_PLLCLKDIVSEL

CM_I3C0FCLKSEL

CM_I3C0FCLKSTCSEL

CM_I3C0FCLKSSEL

CM_MICFILFCLKSEL

CM_ESPICLKSEL

CM_USDHCCLKSEL

CM_FLEXIOCLKSEL

CM_FLEXCAN0CLKSEL

CM_FLEXCAN1CLKSEL

CM_ENETRMIICLKSEL

CM_ENETPTPREFCLKSEL

CM_EWM0CLKSEL

CM_WDT1CLKSEL

CM_OSTIMERCLKSEL

CM_CMP0FCLKSEL

CM_CMP0RRCLKSEL

CM_CMP1FCLKSEL

CM_CMP1RRCLKSEL

CM_CMP2FCLKSEL

CM_CMP2RRCLKSEL

CM_SAI0CLKSEL

CM_SAI1CLKSEL

CM_EMVSIM0CLKSEL

CM_EMVSIM1CLKSEL

CM_I3C1FCLKSEL

CM_I3C1FCLKSTCSEL

CM_I3C1FCLKSSEL

CM_SCGRCCRSCSCLKSEL

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

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

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

firc_trim_mode_t trimMode
: Trim mode.

firc_trim_src_t trimSrc
: Trim source.

uint16_t trimDiv
: Divider of SOSC.

uint8_t trimCoar
: Trim coarse value; Irrelevant if trimMode is kSCG_TrimUpdate.

uint8_t trimFine
: Trim fine value; Irrelevant if trimMode is kSCG_TrimUpdate.

sirc_trim_mode_t trimMode
: Trim mode.

sirc_trim_src_t trimSrc
: Trim source.

uint16_t trimDiv
: Divider of SOSC.

uint8_t cltrim
: Trim coarse value; Irrelevant if trimMode is kSCG_TrimUpdate.

uint8_t ccotrim
: Trim fine value; Irrelevant if trimMode is kSCG_TrimUpdate.

bool enableInternalCapBank
: enable/disable the internal capacitance bank.

bool enableCrystalOscillatorBypass
: enable/disable the crystal oscillator bypass.

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

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

vbat_osc_coarse_adjustment_value_t coarseAdjustment
: 32kHz crystal oscillator amplifier coarse adjustment value.

vbat_osc_init_trim_t initTrim

vbat_osc_cap_trim_t capTrim

vbat_osc_dly_trim_t dlyTrim

vbat_osc_cap2_trim_t cap2Trim

vbat_osc_cmp_trim_t cmpTrim

vbat_osc_mode_en_t mode

vbat_osc_xtal_cap_t xtalCap

vbat_osc_extal_cap_t extalCap

vbat_osc_coarse_adjustment_value_t ampGain

osc32k_clk_gate_id_t id

uint32_t desiredRate
: Desired PLL rate in Hz

uint32_t inputSource
: PLL input source

uint32_t flags
: PLL configuration flags, Or'ed value of PLL_CONFIGFLAG_* definitions

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

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

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

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

uint32_t pllctrl
: PLL Control register APLLCTRL

uint32_t pllndiv
: PLL N Divider register APLLNDIV

uint32_t pllpdiv
: PLL P Divider register APLLPDIV

uint32_t pllmdiv
: PLL M Divider register APLLMDIV

uint32_t pllsscg[2]
: PLL Spread Spectrum Control registers APLLSSCG

uint32_t pllRate
: Acutal PLL rate

struct _firc_trim_config
: *#include <fsl_clock.h>* firc trim configuration.

struct _sirc_trim_config
: *#include <fsl_clock.h>* sirc trim configuration.

struct _vbat_osc_config
: *#include <fsl_clock.h>* The structure of oscillator configuration.

struct _osc_32k_config
: *#include <fsl_clock.h>* The structure of oscillator configuration.

struct _pll_config
: *#include <fsl_clock.h>* PLL configuration structure.

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

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

2.5 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- Version 2.0.4
  - Release peripheral from reset if necessary in init function.
- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues
- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

enum _crc_bits

CRC bit width.

Values:

enumerator kCrcBits16

Generate 16-bit CRC code

enumerator kCrcBits32

Generate 32-bit CRC code

enum _crc_result

CRC result type.

Values:

enumerator kCrcFinalChecksum

CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

enumerator kCrcIntermediateChecksum

CRC data register read value is intermediate checksum (raw value). Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for CRC_Init() to continue adding data to this checksum.

typedef enum *_crc_bits* crc_bits_t

CRC bit width.

typedef enum *_crc_result* crc_result_t

CRC result type.

typedef struct *_crc_config* crc_config_t

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void CRC_Init(CRC_Type *base, const *crc_config_t* *config)

Enables and configures the CRC peripheral module.

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

- base – CRC peripheral address.
- config – CRC module configuration structure.

static inline void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

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

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

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

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

Parameters

- config – CRC protocol configuration structure.

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

Writes data to the CRC module.

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

Parameters

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

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

Returns

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

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

Returns

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

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

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

struct _crc_config

#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

CRC Polynomial, MSBit first. Example polynomial: 0x1021 = 1_0000_0010_0001 = x^12+x^5+1

uint32_t seed

Starting checksum value

bool reflectIn

Reflect bits on input.

bool reflectOut

Reflect bits on output.

bool complementChecksum

True if the result shall be complement of the actual checksum.

crc_bits_t crcBits

Selects 16- or 32- bit CRC protocol.

crc_result_t crcResult

Selects final or intermediate checksum return from CRC_Get16bitResult() or CRC_Get32bitResult()

2.6 CTIMER: Standard counter/timers

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

Ungates the clock and configures the peripheral for basic operation.

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

Parameters

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

void CTIMER_Deinit(CTIMER_Type *base)

Gates the timer clock.

Parameters

- base – Ctimer peripheral base address

void CTIMER_GetDefaultConfig(*ctimer_config_t* *config)

Fills in the timers configuration structure with the default settings.

The default values are:

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

Parameters

- config – Pointer to the user configuration structure.

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

Configures the PWM signal parameters.

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

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

---

Parameters

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

Returns

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

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

Configures the PWM signal parameters.

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

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

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period

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

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

Updates the pulse period of an active PWM signal.

Parameters

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

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

Updates the duty cycle of an active PWM signal.

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

Parameters

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

Returns

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

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

Enables the selected Timer interrupts.

Parameters

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

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

Disables the selected Timer interrupts.

Parameters

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

static inline uint32_t CTIMER_GetEnabledInterrupts(CTIMER_Type *base)

Gets the enabled Timer interrupts.

Parameters

- base – Ctimer peripheral base address

Returns

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

static inline uint32_t CTIMER_GetStatusFlags(CTIMER_Type *base)

Gets the Timer status flags.

Parameters

- base – Ctimer peripheral base address

Returns

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

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

Clears the Timer status flags.

Parameters

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

static inline void CTIMER_StartTimer(CTIMER_Type *base)

Starts the Timer counter.

Parameters

- base – Ctimer peripheral base address

static inline void CTIMER_StopTimer(CTIMER_Type *base)

Stops the Timer counter.

Parameters

- base – Ctimer peripheral base address

FSL_CTIMER_DRIVER_VERSION

Version 2.3.3

enum _ctimer_capture_channel

List of Timer capture channels.

Values:

enumerator kCTIMER_Capture_0

Timer capture channel 0

enumerator kCTIMER_Capture_1

Timer capture channel 1

enumerator kCTIMER_Capture_3

Timer capture channel 3

enum _ctimer_capture_edge

List of capture edge options.

Values:

enumerator kCTIMER_Capture_RiseEdge
Capture on rising edge

enumerator kCTIMER_Capture_FallEdge
Capture on falling edge

enumerator kCTIMER_Capture_BothEdge
Capture on rising and falling edge

enum _ctimer_match
List of Timer match registers.

Values:

enumerator kCTIMER_Match_0
Timer match register 0

enumerator kCTIMER_Match_1
Timer match register 1

enumerator kCTIMER_Match_2
Timer match register 2

enumerator kCTIMER_Match_3
Timer match register 3

enum _ctimer_external_match
List of external match.

Values:

enumerator kCTIMER_External_Match_0
External match 0

enumerator kCTIMER_External_Match_1
External match 1

enumerator kCTIMER_External_Match_2
External match 2

enumerator kCTIMER_External_Match_3
External match 3

enum _ctimer_match_output_control
List of output control options.

Values:

enumerator kCTIMER_Output_NoAction
No action is taken

enumerator kCTIMER_Output_Clear
Clear the EM bit/output to 0

enumerator kCTIMER_Output_Set
Set the EM bit/output to 1

enumerator kCTIMER_Output_Toggle
Toggle the EM bit/output

enum _ctimer_timer_mode
List of Timer modes.

Values:

enumerator kCTIMER_TimerMode

enumerator kCTIMER_IncreaseOnRiseEdge

enumerator kCTIMER_IncreaseOnFallEdge

enumerator kCTIMER_IncreaseOnBothEdge

enum _ctimer_interrupt_enable

List of Timer interrupts.

Values:

enumerator kCTIMER_Match0InterruptEnable

Match 0 interrupt

enumerator kCTIMER_Match1InterruptEnable

Match 1 interrupt

enumerator kCTIMER_Match2InterruptEnable

Match 2 interrupt

enumerator kCTIMER_Match3InterruptEnable

Match 3 interrupt

enum _ctimer_status_flags

List of Timer flags.

Values:

enumerator kCTIMER_Match0Flag

Match 0 interrupt flag

enumerator kCTIMER_Match1Flag

Match 1 interrupt flag

enumerator kCTIMER_Match2Flag

Match 2 interrupt flag

enumerator kCTIMER_Match3Flag

Match 3 interrupt flag

enum ctimer_callback_type_t

Callback type when registering for a callback. When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Values:

enumerator kCTIMER_SingleCallback

Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently

enumerator kCTIMER_MultipleCallback

Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

typedef enum *_ctimer_capture_channel* ctimer_capture_channel_t

List of Timer capture channels.

typedef enum *_ctimer_capture_edge* ctimer_capture_edge_t

List of capture edge options.

typedef enum *_ctimer_match* ctimer_match_t

List of Timer match registers.

typedef enum *_ctimer_external_match* ctimer_external_match_t

List of external match.

typedef enum *_ctimer_match_output_control* ctimer_match_output_control_t

List of output control options.

typedef enum *_ctimer_timer_mode* ctimer_timer_mode_t

List of Timer modes.

typedef enum *_ctimer_interrupt_enable* ctimer_interrupt_enable_t

List of Timer interrupts.

typedef enum *_ctimer_status_flags* ctimer_status_flags_t

List of Timer flags.

typedef void (*ctimer_callback_t)(uint32_t flags)

typedef struct *_ctimer_match_config* ctimer_match_config_t

Match configuration.

This structure holds the configuration settings for each match register.

typedef struct *_ctimer_config* ctimer_config_t

Timer configuration structure.

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

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

void CTIMER_SetupMatch(CTIMER_Type *base, *ctimer_match_t* matchChannel, const *ctimer_match_config_t* *config)

Setup the match register.

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

- base – Ctimer peripheral base address
- matchChannel – Match register to configure
- config – Pointer to the match configuration structure

uint32_t CTIMER_GetOutputMatchStatus(CTIMER_Type *base, uint32_t matchChannel)

Get the status of output match.

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

Parameters

- base – Ctimer peripheral base address
- matchChannel – External match channel, user can obtain the status of multiple match channels at the same time by using the logic of "|" enumeration ctimer_external_match_t

Returns

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

void CTIMER_SetupCapture(CTIMER_Type *base, *ctimer_capture_channel_t* capture, *ctimer_capture_edge_t* edge, bool enableInt)

Setup the capture.

Parameters

- base – Ctimer peripheral base address
- capture – Capture channel to configure
- edge – Edge on the channel that will trigger a capture
- enableInt – Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

static inline uint32_t CTIMER_GetTimerCountValue(CTIMER_Type *base)

Get the timer count value from TC register.

Parameters

- base – Ctimer peripheral base address.

Returns

return the timer count value.

void CTIMER_RegisterCallBack(CTIMER_Type *base, *ctimer_callback_t* *cb_func, *ctimer_callback_type_t* cb_type)

Register callback.

This function configures CTimer Callback in following modes:

- Single Callback: cb_func should be pointer to callback function pointer For example: ctimer_callback_t ctimer_callback = pwm_match_callback; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback, kCTIMER_SingleCallback);
- Multiple Callback: cb_func should be pointer to array of callback function pointers Each element corresponds to Interrupt Flag in IR register. For example: ctimer_callback_t ctimer_callback_table[] = { ctimer_match0_callback, NULL, NULL, ctimer_match3_callback, NULL, NULL, NULL, NULL}; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback_table[0], kCTIMER_MultipleCallback);

Parameters

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

static inline void CTIMER_Reset(CTIMER_Type *base)

Reset the counter.

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

Parameters

- base – Ctimer peripheral base address

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

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

Parameters

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

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

Get capture channel value.

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

Parameters

- base – Ctimer peripheral base address
- capture – Select capture channel

Returns

The timer count capture value.

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

Enable reset match channel.

Set the specified match channel reset operation.

Parameters

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

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

Enable stop match channel.

Set the specified match channel stop operation.

Parameters

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

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

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

Parameters

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

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

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

Parameters

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

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

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

Parameters

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

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

Set the specified match shadow channel.

Parameters

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

struct __ctimer_match_config

#include <fsl_ctimer.h> Match configuration.

This structure holds the configuration settings for each match register.

Public Members

uint32_t matchValue

This is stored in the match register

bool enableCounterReset

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

bool enableCounterStop

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

ctimer_match_output_control_t outControl

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

bool outPinInitState

Initial value of the EM bit/output

bool enableInterrupt

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

struct __ctimer_config

#include <fsl_ctimer.h> Timer configuration structure.

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

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

Public Members

ctimer_timer_mode_t mode
Timer mode

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

uint32_t prescale
Prescale value

2.7 DAC: Digital-to-Analog Converter Driver

void DAC_Init(LPDAC_Type *base, const *dac_config_t* *config)
Initialize the DAC module with common configuartion.

The clock will be enabled in this function.

Parameters

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

void DAC_GetDefaultConfig(*dac_config_t* *config)
Get the default settings for initialization's configuration.

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

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

Parameters

- config – Pointer to configuration structure.

void DAC_Deinit(LPDAC_Type *base)
De-initialize the DAC module.

The clock will be disabled in this function.

Parameters

- base – DAC peripheral base address.

static inline void DAC_SetReset(LPDAC_Type *base, uint32_t mask)
Assert the reset control to part hardware.

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

Parameters

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

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

Clear the reset control to part hardware.

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

Parameters

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

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

Enable the DAC hardware system or not.

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

Parameters

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

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

Enable the interrupts.

Parameters

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

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

Disable the interrupts.

Parameters

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

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

Enable the DMA switchers or not.

Parameters

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

static inline uint32_t DAC_GetStatusFlags(LPDAC_Type *base)

Get status flags of DAC module.

Parameters

- base – DAC peripheral base address.

Returns

Mask value of status flags. See to _dac_status_flags.

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

Clear status flags of DAC module.

Parameters

- base – DAC peripheral base address.

- flags – Mask value of status flags to be cleared. See to _dac_status_flags.

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

Set data into the entry of FIFO buffer.

Parameters

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

static inline uint32_t DAC_GetFIFOWritePointer(LPDAC_Type *base)

Get the value of the FIFO write pointer.

Parameters

- base – DAC peripheral base address.

Returns

Current value of the FIFO write pointer.

static inline uint32_t DAC_GetFIFOReadPointer(LPDAC_Type *base)

Get the value of the FIFO read pointer.

Parameters

- base – DAC peripheral base address.

Returns

Current value of the FIFO read pointer.

static inline void DAC_DoSoftwareTriggerFIFO(LPDAC_Type *base)

Do software trigger to FIFO when in software mode.

Parameters

- base – DAC peripheral base address.

FSL_DAC_DRIVER_VERSION

DAC driver version 2.1.3.

DAC reset control.

Values:

enumerator kDAC_ResetFIFO

Resets the FIFO pointers and flags.

enumerator kDAC_ResetLogic

Resets all DAC registers and internal logic.

DAC interrupts.

Values:

enumerator kDAC_FIFOFullInterruptEnable

FIFO full interrupt enable.

enumerator kDAC_FIFOEmptyInterruptEnable

FIFO empty interrupt enable.

enumerator kDAC_FIFOWatermarkInterruptEnable

FIFO watermark interrupt enable.

enumerator kDAC_SwingBackInterruptEnable

Swing back one cycle complete interrupt enable.

enumerator kDAC_FIFOOverflowInterruptEnable
FIFO overflow interrupt enable.

enumerator kDAC_FIFOUnderflowInterruptEnable
FIFO underflow interrupt enable.

enumerator kDAC_PeriodTriggerCompleteInterruptEnable
Period trigger mode conversion complete interrupt enable

DAC DMA switchers.

Values:

enumerator kDAC_FIFOEmptyDMAEnable
FIFO empty DMA enable.

enumerator kDAC_FIFOWatermarkDMAEnable
FIFO watermark DMA enable.

DAC status flags.

Values:

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

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

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

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

enumerator kDAC_FIFOEmptyFlag
FIFO empty flag.

enumerator kDAC_FIFOFullFlag
FIFO full flag.

enumerator kDAC_PeriodTriggerCompleteFlag
Period trigger mode conversion complete flag.

enum _dac_fifo_trigger_mode
DAC FIFO trigger mode.

Values:

enumerator kDAC_FIFOTriggerByHardwareMode

Buffer would be triggered by hardware.

enumerator kDAC_FIFOTriggerBySoftwareMode

Buffer would be triggered by software.

enum _dac_fifo_work_mode

DAC FIFO work mode.

Values:

enumerator kDAC_FIFODisabled

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

enumerator kDAC_FIFOWorkAsNormalMode

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

enumerator kDAC_FIFOWorkAsSwingMode

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

enumerator kDAC_FIFOWorkAsPeriodTriggerMode

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

enumerator kDAC_FIFOWorkAsPeriodTriggerAndSwingMode

Periodically trigger DAC and swing back.

enum _dac_reference_voltage_source

DAC reference voltage source.

Values:

enumerator kDAC_ReferenceVoltageSourceAlt1

The DAC selects VREFH_INT as the reference voltage.

enumerator kDAC_ReferenceVoltageSourceAlt2

The DAC selects VREFH_EXT as the reference voltage.

enum _dac_reference_current_source

Values:

enumerator kDAC_ReferenceCurrentSourcePtat

enumerator kDAC_ReferenceCurrentSourceZtc

typedef enum *_dac_fifo_trigger_mode* dac_fifo_trigger_mode_t

DAC FIFO trigger mode.

typedef enum *_dac_fifo_work_mode* dac_fifo_work_mode_t

DAC FIFO work mode.

typedef enum *_dac_reference_voltage_source* dac__reference__voltage__source__t

DAC reference voltage source.

typedef enum *_dac_reference_current_source* dac__reference__current__source__t

typedef struct *_dac_config* dac__config__t

DAC configuration structure.

struct __dac__config

#include <fsl_dac.h> DAC configuration structure.

Public Members

uint32_t fifoWatermarkLevel

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

dac_fifo_trigger_mode_t fifoTriggerMode

Select the trigger mode for FIFO.

dac_fifo_work_mode_t fifoWorkMode

Select the work mode for FIFO.

bool enableOpampBuffer

Opamp is used as buffer.

bool enableLowerLowPowerMode

Enable the lower low power mode.

uint32_t periodicTriggerNumber

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

uint32_t periodicTriggerWidth

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

uint32_t syncTime

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

dac_reference_current_source_t referenceCurrentSource

Select the internal reference current source.

dac_reference_voltage_source_t referenceVoltageSource

Select the reference voltage source.

2.8 DAC14: 14-bit Digital-to-Analog Converter Driver

void DAC14__Init(HPDAC_Type *base, const *dac14_config_t* *config)

Initialize the DAC14 module with common configuartion.

The clock will be enabled in this function.

Parameters

- base – DAC14 peripheral base address.
- config – Pointer to configuration structure.

void DAC14_GetDefaultConfig(*dac14_config_t* *config)

Get the default settings for initialization's configuration.

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

```
config->fifoWatermarkLevel = 0U;
config->TriggerSource    = kDAC14_HardwareTriggerSource;
config->WorkMode         = kDAC14_BufferWorkMode;
config->enableOpampBuffer  = false;
config->enableADC  = false;
config->periodicTriggerNumber = 0U;
config->periodicTriggerWidth  = 0U;
```

Parameters

- config – Pointer to configuration structure.

void DAC14_Deinit(HPDAC_Type *base)

De-initialize the DAC14 module.

The clock will be disabled in this function.

Parameters

- base – DAC14 peripheral base address.

static inline void DAC14_DoSoftwareReset(HPDAC_Type *base)

Do software reset .

This function is resets all DAC registers and internal logic.

Parameters

- base – DAC14 peripheral base address.

static inline void DAC14_DoFIFOReset(HPDAC_Type *base)

Do FIFO reset.

This function is resets the FIFO pointers and flags in FIFO Status.

Parameters

- base – DAC14 peripheral base address.

static inline void DAC14_AbortPeriodTriggerConvSequence(HPDAC_Type *base)

Abort DAC14 period trigger conversion sequence.

This function is write 0 to PTGEN to terminate the current conversion sequence.

Parameters

- base – DAC14 peripheral base address.

static inline void DAC14_Enable(HPDAC_Type *base, bool enable)

Enable the DAC14 system.

Parameters

- base – DAC14 peripheral base address.
- enable – true to enable and false to disable.

static inline void DAC14_EnableSwingBackMode(HPDAC_Type *base, bool enable)

Enable swing back mode.

Parameters

- base – DAC14 peripheral base address.
- enable – true to enable and false to disable.

static inline void DAC14_EnableFIFOMode(HPDAC_Type *base, bool enable)

Enable FIFO mode.

Parameters

- base – DAC14 peripheral base address.
- enable – true to enable and false to disable.

static inline void DAC14_EnableInterrupts(HPDAC_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – DAC14 peripheral base address.
- mask – Mask value of indicated interrupt events, please see _dac14_interrupt_enable for details.

static inline void DAC14_DisableInterrupts(HPDAC_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – DAC14 peripheral base address.
- mask – Mask value of indicated interrupt events, please see _dac14_interrupt_enable for details.

static inline void DAC14_EnableDMA(HPDAC_Type *base, uint32_t mask, bool enable)

Enable the DMA switchers or not.

Parameters

- base – DAC14 peripheral base address.
- mask – Mask value of indicated DMA requeset, please see _dac14_dma_enable for details.
- enable – true to enable and false to disable.

static inline uint32_t DAC14_GetStatusFlags(HPDAC_Type *base)

Get status flags of DAC14 module.

Parameters

- base – DAC14 peripheral base address.

Returns

Current DAC status flags.

static inline void DAC14_ClearStatusFlags(HPDAC_Type *base, uint32_t flags)

Clear status flags of DAC14 module.

Parameters

- base – DAC14 peripheral base address.
- flags – Mask value of status flags to be cleared, please see _dac14_status_flags for details.

static inline void DAC14_SetData(HPDAC_Type *base, uint32_t value)

Set data into the entry of FIFO buffer.

Parameters

- base – DAC14 peripheral base address.
- value – Setting value into FIFO buffer.

static inline uint32_t DAC14_GetFIFOWritePointer(HPDAC_Type *base)

Get the value of the FIFO write pointer.

Parameters

- base – DAC14 peripheral base address.

Returns

Current value of the FIFO write pointer.

static inline uint32_t DAC14_GetFIFOReadPointer(HPDAC_Type *base)

Get the value of the FIFO read pointer.

Parameters

- base – DAC14 peripheral base address.

Returns

Current value of the FIFO read pointer.

static inline void DAC14_DoSoftwareTrigger(HPDAC_Type *base)

Do software trigger.

Parameters

- base – DAC14 peripheral base address.

FSL_DAC14_DRIVER_VERSION

DAC14 driver version 2.0.0.

enum _dac14_interrupt_enable

DAC14 interrupts enumeration.

Values:

enumerator kDAC14_PeriodTriggerCompleteInterruptEnable

Period trigger mode conversion complete interrupt enable

enumerator kDAC14_FIFOUnderflowInterruptEnable

FIFO underflow interrupt enable.

enumerator kDAC14_FIFOOverflowInterruptEnable

FIFO overflow interrupt enable.

enumerator kDAC14_SwingBackInterruptEnable

Swing back one cycle complete interrupt enable.

enumerator kDAC14_FIFOWatermarkInterruptEnable

FIFO watermark interrupt enable.

enumerator kDAC14_FIFOEmptyInterruptEnable

FIFO empty interrupt enable.

enumerator kDAC14_FIFOFullInterruptEnable

FIFO full interrupt enable.

enum _dac14_dma_enable

DAC14 DMA switchers.

Values:

enumerator kDAC14_FIFOWatermarkDMAEnable

FIFO watermark DMA enable.

enumerator kDAC14_FIFOEmptyDMAEnable

FIFO empty DMA enable.

enum _dac14_status_flags

DAC14 status flags.

Values:

enumerator kDAC14_PeriodTriggerCompleteFlag

Period trigger mode conversion complete flag.

enumerator kDAC14_FIFOUnderflowFlag

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

enumerator kDAC14_FIFOOverflowFlag

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

enumerator kDAC14_SwingBackCompleteFlag

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

enumerator kDAC14_FIFOWaterMarkFlag

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

enumerator kDAC14_FIFOEmptyFlag

FIFO empty flag, when CPU or DMA writes data to FIFO, this bit will automatically clear.

enumerator kDAC14_FIFOFullFlag

FIFO full flag, when software trigger and hardware trigger read FIFO automatically clears this flag.

enum _dac14_trigger_source

DAC14 trigger source, include software and hardware.

Values:

enumerator kDAC14_HardwareTriggerSource

Trigger source selection hardware .

enumerator kDAC14_SoftwareTriggerSource

Trigger source selection software .

enum _dac14_work_mode

DAC14 work mode.

Values:

enumerator kDAC14_BufferWorkMode

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

enumerator kDAC14_FIFOWorkMode

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

enumerator kDAC14_SwingBackWorkMode

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

enumerator kDAC14_PeriodTriggerWorkMode

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

enumerator kDAC14_PeriodTriggerAndSwingBackWorkMode

Periodically trigger DAC and swing back.

typedef enum *_dac14_trigger_source* dac14_trigger_source_t

DAC14 trigger source, include software and hardware.

typedef enum *_dac14_work_mode* dac14_work_mode_t

DAC14 work mode.

typedef struct *_dac14_config* dac14_config_t

DAC14 configuration structure.

struct _dac14_config

#include <fsl_dac14.h> DAC14 configuration structure.

Public Members

uint16_t periodicTriggerNumber

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

uint16_t periodicTriggerWidth

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

uint8_t fifoWatermarkLevel

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

bool enableOpampBuffer

Opamp is used as buffer.

bool enableDAC
Enable the DAC system.

dac14_work_mode_t WorkMode
Select DAC work mode.

dac14_trigger_source_t TriggerSource
Select DAC trigger source.

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, ....)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

d. submit loop transfer

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

Parameters

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

Return values

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

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

Submits the eDMA transfer request.

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

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

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

status_t EDMA_SubmitLoopTransfer(*edma_handle_t* *handle, *edma_transfer_config_t* *transfer, uint32_t transferLoopCount)

Submits the eDMA scatter gather transfer configurations.

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
edma_channel_transfer_config_t transfer[2];
EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

---

Note: Application should check the return value of this function to avoid transfer request submit failed

---

Parameters

- handle – eDMA handle pointer
- transfer – pointer to user's eDMA channel configure structure, see edma_channel_transfer_config_t for detail
- transferLoopCount – the count of the transfer ring, if loop count is 1, that means that the one will link to itself.

Return values

- kStatus_Success – It means submit transfer request succeed
- kStatus_EDMA_Busy – channel is in busy status
- kStatus_InvalidArgument – Invalid Argument

void EDMA_StartTransfer(*edma_handle_t* *handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

void EDMA_StopTransfer(*edma_handle_t* *handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call EDMA_StartTransfer() again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

void EDMA_AbortTransfer(*edma_handle_t* *handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns

The next TCD address.

void EDMA_HandleIRQ(*edma_handle_t* *handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the “tcdUsed” updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major

loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- handle – eDMA handle pointer.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.10.7.

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull

TCD queue is full.

enumerator kStatus_EDMA_Busy

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

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes

Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA_TransferSize128Bytes

Source/Destination data transfer size is 128 bytes every time

enum _edma_modulo

eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable

Disable modulo

enumerator kEDMA_Modulo2bytes

Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes
Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes
Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes
Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes
Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone
No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle
eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type
Channel link type.

Values:

enumerator kEDMA_LinkNone
No channel link

enumerator kEDMA_MinorLink
Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.

Values:

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

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

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

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

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

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

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

_edma_interrupt_enable eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum __edma_transfer_type
eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

enum edma_channel_memory_attribute
eDMA channel memory attribute

Values:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

enum _edma_channel_swap_size
eDMA4 channel swap size

Values:

enumerator kEDMA_ChannelSwapDisabled
Swap is disabled.

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

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

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

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

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

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

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

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

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

enum _edma_channel_access_type
eDMA4 channel access type

Values:

enumerator kEDMA_ChannelDataAccess
Data access for eDMA4 transfers.

enumerator kEDMA_ChannelInstructionAccess
Instruction access for eDMA4 transfers.

enum _edma_channel_protection_level
eDMA4 channel protection level

Values:

enumerator kEDMA_ChannelProtectionLevelUser
user protection level for eDMA transfers.

enumerator kEDMA_ChannelProtectionLevelPrivileged
Privileged protection level eDMA transfers.

typedef enum *_edma_transfer_size* edma_transfer_size_t
eDMA transfer configuration

typedef enum *_edma_modulo* edma_modulo_t
eDMA modulo configuration

typedef enum *_edma_bandwidth* edma_bandwidth_t
Bandwidth control.

typedef enum *_edma_channel_link_type* edma_channel_link_type_t
Channel link type.

typedef enum *_edma_transfer_type* edma_transfer_type_t
eDMA transfer type

typedef struct *_edma_channel_Preemption_config* edma_channel_Preemption_config_t
eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* edma_minor_offset_config_t
eDMA minor offset configuration

typedef enum *edma_channel_memory_attribute* edma_channel_memory_attribute_t
eDMA channel memory attribute

typedef enum *_edma_channel_swap_size* edma_channel_swap_size_t
eDMA4 channel swap size

typedef enum *_edma_channel_access_type* edma_channel_access_type_t
eDMA4 channel access type

typedef enum *_edma_channel_protection_level* edma_channel_protection_level_t
eDMA4 channel protection level

typedef struct *_edma_channel_config* edma_channel_config_t
eDMA4 channel configuration

typedef *edma_core_tcd_t* edma_tcd_t
eDMA TCD.

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

typedef struct *_edma_transfer_config* edma_transfer_config_t
edma4 channel transfer configuration

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

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

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

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

a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0

b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width

c. the totalBytes should align with the bytesEachRequest

d. the srcAddr should align with the srcWidthOfEachTransfer

e. the dstAddr should align with the dstWidthOfEachTransfer

f. the srcAddr should align with srcAddrModulo if modulo feature is enabled

g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.

typedef struct *_edma_config* edma_config_t

eDMA global configuration structure.

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

Define callback function for eDMA.

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

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

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

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

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

typedef struct *_edma_handle* edma_handle_t

eDMA transfer handle structure

FSL_EDMA_DRIVER_EDMA4

eDMA driver name

EDMA_ALLOCATE_TCD(name, number)

Macro used for allocate edma TCD.

DMA_DCHPRI_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct _edma_channel_Preemption_config

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption

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

bool enablePreemptAbility

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

uint8_t channelPriority

Channel priority

struct _edma_minor_offset_config

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

bool enableSrcMinorOffset

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

bool enableDestMinorOffset

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

uint32_t minorOffset

Offset for a minor loop mapping.

struct _edma_channel_config

#include <fsl_edma.h> eDMA4 channel configuration

Public Members

edma_channel_Preemption_config_t channelPreemptionConfig

channel preemption configuration

edma_channel_memory_attribute_t channelReadMemoryAttribute

channel memory read attribute configuration

edma_channel_memory_attribute_t channelWriteMemoryAttribute

channel memory write attribute configuration

edma_channel_swap_size_t channelSwapSize

channel swap size configuration

edma_channel_access_type_t channelAccessType

channel access type configuration

uint8_t channelDataSignExtensionBitPosition

channel data sign extension bit psition configuration

uint32_t channelRequestSource

hardware service request source for the channel

bool enableMasterIDReplication

enable master ID replication

edma_channel_protection_level_t protectionLevel

protection level

struct _edma_transfer_config

#include <fsl_edma.h> edma4 channel transfer configuration

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

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

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

---

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

a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0

b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width

c. the totalBytes should align with the bytesEachRequest

d. the srcAddr should align with the srcWidthOfEachTransfer

e. the dstAddr should align with the dstWidthOfEachTransfer

f. the srcAddr should align with srcAddrModulo if modulo feature is enabled

g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

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

int16_t destOffset
: Sign-extended offset value in byte unit applied to the current destination address to form the next-state value as each destination write is completed.

uint32_t minorLoopBytes
: bytes in each minor loop or each request range: 1 - (2^30 -1) when minor loop mapping is enabled range: 1 - (2^10 - 1) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - (2^32 - 1) when minor loop mapping is disabled

uint32_t majorLoopCounts
: minor loop counts in each major loop, should be 1 at least for each transfer range: (0 - (2^15 - 1)) when minor loop channel link is disabled range: (0 - (2^9 - 1)) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajor-Loop * bytesEachMinorLoop

uint16_t enabledInterruptMask
: channel interrupt to enable, can be OR'ed value of _edma_interrupt_enable

edma_modulo_t srcAddrModulo
: source circular data queue range

int32_t srcMajorLoopOffset
: source major loop offset

edma_modulo_t dstAddrModulo
: destination circular data queue range

int32_t dstMajorLoopOffset
: destination major loop offset

bool enableSrcMinorLoopOffset
: enable source minor loop offset

bool enableDstMinorLoopOffset
: enable dest minor loop offset

int32_t minorLoopOffset
: burst offset, the offset will be applied after minor loop update

bool enableChannelMajorLoopLink
: channel link when major loop complete

uint32_t majorLoopLinkChannel
: major loop link channel number

bool enableChannelMinorLoopLink
: channel link when minor loop complete

uint32_t minorLoopLinkChannel
: minor loop link channel number

edma_tcd_t *linkTCD
: pointer to the link transfer control descriptor

struct __edma_config
: *#include <fsl_edma.h>* eDMA global configuration structure.

Public Members

bool enableMasterIdReplication

Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

bool enableGlobalChannelLink

Enable(true) channel linking is available and controlled by each channel's link settings.

bool enableHaltOnError

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableDebugMode

Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

bool enableRoundRobinArbitration

Enable(true) channel linking is available and controlled by each channel's link settings.

edma_channel_config_t *channelConfig[1]

channel preemption configuration

struct __edma_handle

#include <fsl_edma.h> eDMA transfer handle structure

Public Members

edma_callback callback

Callback function for major count exhausted.

void *userData

Callback function parameter.

EDMA_ChannelType *channelBase

eDMA peripheral channel base address.

EDMA_Type *base

eDMA peripheral base address

EDMA_TCDType *tcdBase

eDMA peripheral tcd base address.

edma_tcd_t *tcdPool

Pointer to memory stored TCDs.

uint32_t channel

eDMA channel number.

volatile int8_t header

The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail

The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed

The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize

The total number of TCD slots in the queue.

2.10 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

FSL_EDMA_SOC_DRIVER_VERSION
Driver version 2.0.0.

FSL_EDMA_SOC_IP_DMA3
DMA IP version.

FSL_EDMA_SOC_IP_DMA4

EDMA_BASE_PTRS
DMA base table.

EDMA_CHN_IRQS

EDMA_CHANNEL_OFFSET
EDMA base address convert macro.

EDMA_CHANNEL_ARRAY_STEP(base)

2.12 Efuse_driver

status_t EFUSE_Init(void)
Initialize EFUSE controller.

This function enables EFUSE Controller clock.

Return values

- kStatus_Success – 0 Operation succeeded without error.
- kStatus_Fail – 1 The operation failed with a generic error.
- kStatus_ReadOnly – 2 Requested value cannot be changed because it is read-only.
- kStatus_OutOfRange – 3 Requested value is out of range.
- kStatus_InvalidArgument – 4 The requested command's argument is undefined.
- kStatus_Timeout – An invalid 5 A timeout occurred
- kStatus_NoTransferInProgress – 6 No send in progress.

status_t EFUSE_Deinit(void)

De-Initialize EFUSE controller.

This functin disables EFUSE Controller Clock.

Return values

- kStatus_Success – 0 Operation succeeded without error.
- kStatus_Fail – 1 The operation failed with a generic error.
- kStatus_ReadOnly – 2 Requested value cannot be changed because it is read-only.
- kStatus_OutOfRange – 3 Requested value is out of range.
- kStatus_InvalidArgument – 4 The requested command's argument is undefined.
- kStatus_Timeout – An invalid 5 A timeout occurred
- kStatus_NoTransferInProgress – 6 No send in progress.

status_t EFUSE_Read(uint32_t addr, uint32_t *data)

Read Fuse value from eFuse word.

This function read fuse data from eFuse word to specified data buffer.

Parameters

- addr – Fuse address
- data – Buffer to hold the data read from eFuse word

Return values

- kStatus_Success – 0 Operation succeeded without error.
- kStatus_Fail – 1 The operation failed with a generic error.
- kStatus_ReadOnly – 2 Requested value cannot be changed because it is read-only.
- kStatus_OutOfRange – 3 Requested value is out of range.
- kStatus_InvalidArgument – 4 The requested command's argument is undefined.
- kStatus_Timeout – An invalid 5 A timeout occurred
- kStatus_NoTransferInProgress – 6 No send in progress.

status_t EFUSE_Program(uint32_t addr, uint32_t data)

Program value to eFuse block.

This function program data to specified eFuse address.

Parameters

- addr – Fuse address
- data – data to be programmed into eFuse Fuse block

Return values

- kStatus_Success – 0 Operation succeeded without error.
- kStatus_Fail – 1 The operation failed with a generic error.
- kStatus_ReadOnly – 2 Requested value cannot be changed because it is read-only.
- kStatus_OutOfRange – 3 Requested value is out of range.

- kStatus_InvalidArgument – 4 The requested command's argument is undefined.
- kStatus_Timeout – An invalid 5 A timeout occurred
- kStatus_NoTransferInProgress – 6 No send in progress.

2.13 EIM: error injection module

FSL_ERM_DRIVER_VERSION
Driver version.

void EIM_Init(EIM_Type *base)
EIM module initialization function.

Parameters

- base – EIM base address.

void EIM_Deinit(EIM_Type *base)
De-initializes the EIM.

2.14 ERM: error recording module

void ERM_Init(ERM_Type *base)
ERM module initialization function.

Parameters

- base – ERM base address.

void ERM_Deinit(ERM_Type *base)
De-initializes the ERM.

static inline void ERM_EnableInterrupts(ERM_Type *base, uint32_t channel, uint32_t mask)
ERM enable interrupts.

Parameters

- base – ERM peripheral base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to "_erm_interrupt_enable" enumeration.

static inline void ERM_DisableInterrupts(ERM_Type *base, uint32_t channel, uint32_t mask)
ERM module disable interrupts.

Parameters

- base – ERM base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to "_erm_interrupt_enable" enumeration.

static inline uint32_t ERM_GetInterruptStatus(ERM_Type *base, uint32_t channel)

Gets ERM interrupt flags.

Parameters

- base – ERM peripheral base address.

Returns

ERM event flags.

static inline void ERM_ClearInterruptStatus(ERM_Type *base, uint32_t channel, uint32_t mask)

ERM module clear interrupt status flag.

Parameters

- base – ERM base address.
- mask – event flag to clear. Refer to "_erm_interrupt_flag" enumeration.

uint32_t ERM_GetMemoryErrorAddr(ERM_Type *base, uint32_t channel)

ERM get memory error absolute address, which capturing the address of the last ECC event in Memory n.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

memory – error absolute address.

uint32_t ERM_GetSyndrome(ERM_Type *base, uint32_t channel)

ERM get syndrome, which identifies the pertinent bit position on a correctable, single-bit data inversion or a non-correctable, single-bit address inversion. The syndrome value does not provide any additional diagnostic information on non-correctable, multi-bit inversions.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

syndrome – value.

uint32_t ERM_GetErrorCount(ERM_Type *base, uint32_t channel)

ERM get error count, which records the count value of the number of correctable ECC error events for Memory n. Non-correctable errors are considered a serious fault, so the ERM does not provide any mechanism to count non-correctable errors. Only correctable errors are counted.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

error – count.

void ERM_ResetErrorCount(ERM_Type *base, uint32_t channel)

ERM reset error count.

Parameters

- base – ERM base address.
- channel – memory channel.

FSL_ERM_DRIVER_VERSION
Driver version.

ERM interrupt configuration structure, default settings all disabled, _erm_interrupt_enable.

This structure contains the settings for all of the ERM interrupt configurations.

Values:

enumerator kERM_SingleCorrectionIntEnable
Single Correction Interrupt Notification enable.

enumerator kERM_NonCorrectableIntEnable
Non-Correction Interrupt Notification enable.

enumerator kERM_AllInterruptsEnable
All Interrupts enable

ERM interrupt status, _erm_interrupt_flag.

This provides constants for the ERM event status for use in the ERM functions.

Values:

enumerator kERM_SingleBitCorrectionIntFlag
Single-Bit Correction Event.

enumerator kERM_NonCorrectableErrorIntFlag
Non-Correctable Error Event.

enumerator kERM_AllIntsFlag
All Events.

2.15 EVTG: Event Generator Driver

FSL_EVTG_DRIVER_VERSION
EVTG driver version.

Version 2.0.2.

enum _evtg_index
EVTG instance index.

Values:

enumerator kEVTG_Index0
EVTG instance index 0.

enumerator kEVTG_Index1
EVTG instance index 1.

enumerator kEVTG_Index2
EVTG instance index 2.

enumerator kEVTG_Index3
EVTG instance index 3.

enum _evtg_input_index
EVTG input index.

Values:

enumerator kEVTG_InputA
EVTG input A.

enumerator kEVTG_InputB
EVTG input B.

enumerator kEVTG_InputC
EVTG input C.

enumerator kEVTG_InputD
EVTG input D.

enum _evtg_aoi_index
EVTG AOI index.

Values:

enumerator kEVTG_AOI0
EVTG AOI index 0.

enumerator kEVTG_AOI1
EVTG AOI index 1.

enum _evtg_aoi_product_term
EVTG AOI product term index.

Values:

enumerator kEVTG_ProductTerm0
EVTG AOI product term index 0.

enumerator kEVTG_ProductTerm1
EVTG AOI product term index 1.

enumerator kEVTG_ProductTerm2
EVTG AOI product term index 2.

enumerator kEVTG_ProductTerm3
EVTG AOI product term index 3.

enum _evtg_aoi_input_config
EVTG input configuration.

Values:

enumerator kEVTG_InputLogicZero
Force input in product term to a logical zero.

enumerator kEVTG_InputDirectPass
Pass input in product term.

enumerator kEVTG_InputComplement
Complement input in product term.

enumerator kEVTG_InputLogicOne
Force input in product term to a logical one.

enum _evtg_aoi_outfilter_count
EVTG AOI Output Filter Sample Count.

Values:

enumerator kEVTG_AOIOutFilterSampleCount3
EVTG AOI output filter sample count is 3.

enumerator kEVTG_AOIOutFilterSampleCount4
EVTG AOI output filter sample count is 4.

enumerator kEVTG_AOIOutFilterSampleCount5
EVTG AOI output filter sample count is 5.

enumerator kEVTG_AOIOutFilterSampleCount6
EVTG AOI output filter sample count is 6.

enumerator kEVTG_AOIOutFilterSampleCount7
EVTG AOI output filter sample count is 7.

enumerator kEVTG_AOIOutFilterSampleCount8
EVTG AOI output filter sample count is 8.

enumerator kEVTG_AOIOutFilterSampleCount9
EVTG AOI output filter sample count is 9.

enumerator kEVTG_AOIOutFilterSampleCount10
EVTG AOI output filter sample count is 10.

enum _evtg_outfdbk_override_input
EVTG output feedback override control mode. When FF is configured as JK-FF mode, need EVTG_OUTA feedback to EVTG input and replace one of the four inputs.

Values:

enumerator kEVTG_OutputOverrideInputA
Replace input A.

enumerator kEVTG_OutputOverrideInputB
Replace input B.

enumerator kEVTG_OutputOverrideInputC
Replace input C.

enumerator kEVTG_OutputOverrideInputD
Replace input D.

enum _evtg_flipflop_mode
EVTG flip flop mode configuration.

Values:

enumerator kEVTG_FFModeBypass
Bypass mode (default).In this mode, user can choose to enable or disable input sync logic and filter function.

enumerator kEVTG_FFModeRSTrigger
RS trigger mode. In this mode, user can choose to enable or disable input sync logic and filter function.

enumerator kEVTG_FFModeTFF
T-FF mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enumerator kEVTG_FFModeDFF
D-FF mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enumerator kEVTG_FFModeJKFF
JK-FF mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enumerator kEVTG_FFModeLatch

Latch mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enum _evtg_flipflop_init_output

EVTG flip-flop initial value.

Values:

enumerator kEVTG_FFInitOut0

Configure the positive output of flip-flop as 0.

enumerator kEVTG_FFInitOut1

Configure the positive output of flip-flop as 1.

typedef enum *_evtg_index* evtg_index_t

EVTG instance index.

typedef enum *_evtg_input_index* evtg_input_index_t

EVTG input index.

typedef enum *_evtg_aoi_index* evtg_aoi_index_t

EVTG AOI index.

typedef enum *_evtg_aoi_product_term* evtg_aoi_product_term_t

EVTG AOI product term index.

typedef enum *_evtg_aoi_input_config* evtg_aoi_input_config_t

EVTG input configuration.

typedef enum *_evtg_aoi_outfilter_count* evtg_aoi_outfilter_count_t

EVTG AOI Output Filter Sample Count.

typedef enum *_evtg_outfdbk_override_input* evtg_outfdbk_override_input_t

EVTG output feedback override control mode. When FF is configured as JK-FF mode, need EVTG_OUTA feedback to EVTG input and replace one of the four inputs.

typedef enum *_evtg_flipflop_mode* evtg_flipflop_mode_t

EVTG flip flop mode configuration.

typedef enum *_evtg_flipflop_init_output* evtg_flipflop_init_output_t

EVTG flip-flop initial value.

typedef struct *_evtg_aoi_outfilter_config* evtg_aoi_outfilter_config_t

The structure for configuring an AOI output filter sample.

AOI output filter sample count represent the number of consecutive samples that must agree prior to the AOI output filter accepting an transition. AOI output filter sample period represent the sampling period (in IP bus clock cycles) of the AOI output signals. Each AOI output is sampled multiple times at the rate specified by this period.

For the modes with Filter function enabled, filter delay is “(FILT_CNT + 3) x FILT_PER + 2”.

typedef struct *_evtg_aoi_product_term_config* evtg_aoi_product_term_config_t

The structure for configuring an AOI product term.

typedef struct *_evtg_aoi_config* evtg_aoi_config_t

EVTG AOI configuration structure.

typedef struct *_evtg_config* evtg_config_t

EVTG configuration covering all configurable fields.

struct _evtg_aoi_outfilter_config

#include <fsl_evtg.h> The structure for configuring an AOI output filter sample.

AOI output filter sample count represent the number of consecutive samples that must agree prior to the AOI output filter accepting an transition. AOI output filter sample period represent the sampling period (in IP bus clock cycles) of the AOI output signals. Each AOI output is sampled multiple times at the rate specified by this period.

For the modes with Filter function enabled, filter delay is “(FILT_CNT + 3) x FILT_PER + 2”.

Public Members

evtg_aoi_outfilter_count_t sampleCount

EVTG AOI output filter sample count. refer to evtg_aoi_outfilter_count_t.

uint8_t samplePeriod

EVTG AOI output filter sample period, within 0~255. If sample period value is 0x00 (default), then the input filter is bypassed.

struct _evtg_aoi_product_term_config

#include <fsl_evtg.h> The structure for configuring an AOI product term.

Public Members

evtg_aoi_input_config_t aInput

Input A configuration.

evtg_aoi_input_config_t bInput

Input B configuration.

evtg_aoi_input_config_t cInput

Input C configuration.

evtg_aoi_input_config_t dInput

Input D configuration.

struct _evtg_aoi_config

#include <fsl_evtg.h> EVTG AOI configuration structure.

Public Members

evtg_aoi_outfilter_config_t aoiOutFilterConfig

EVTG AOI output filter sample configuration structure.

evtg_aoi_product_term_config_t productTerm0

Configure AOI product term0.

evtg_aoi_product_term_config_t productTerm1

Configure AOI product term1.

evtg_aoi_product_term_config_t productTerm2

Configure AOI product term2.

evtg_aoi_product_term_config_t productTerm3

Configure AOI product term3.

struct _evtg_config

#include <fsl_evtg.h> EVTG configuration covering all configurable fields.

Public Members

bool enableInputASync
: Enable/Disable EVTG A input synchronous with bus clk.

bool enableInputBSync
: Enable/Disable EVTG B input synchronous with bus clk.

bool enableInputCSync
: Enable/Disable EVTG C input synchronous with bus clk.

bool enableInputDSync
: Enable/Disable EVTG D input synchronous with bus clk.

evtg_outfdbk_override_input_t outfdbkOverideinput
: EVTG output feedback to EVTG input and replace one of the four inputs.

evtg_flipflop_mode_t flipflopMode
: Flip-Flop can be configured as one of Bypass mode, RS trigger mode, T-FF mode, D-FF mode, JK-FF mode, Latch mode.

bool enableFlipflopInitOutput
: Flip-flop initial output value enable/disable.

evtg_flipflop_init_output_t flipflopInitOutputValue
: Flip-flop initial output value configuration.

bool enableForceBypassFlipFlopAOI0
: Enable/Disable force bypass Flip-Flop and route the AOI_0(Filter_0) value directly to EVTG_OUTA

bool enableForceBypassFlipFlopAOI1
: Enable/Disable force bypass Flip-Flop and route the AOI_1(Filter_1) value directly to EVTG_OUTB

evtg_aoi_config_t aoi0Config
: Configure EVTG AOI0.

evtg_aoi_config_t aoi1Config
: Configure EVTG AOI1.

2.16 EVTG Driver

void EVTG_Init(EVTG_Type *base, *evtg_index_t* evtgIndex, *evtg_config_t* *psConfig)
: Initialize EVTG with a user configuration structure.

  Parameters

  - base – EVTG base address.
  - evtgIndex – EVTG instance index.
  - psConfig – EVTG initial configuration structure pointer.

void EVTG_GetDefaultConfig(*evtg_config_t* *psConfig, *evtg_flipflop_mode_t* flipflopMode)
: Loads default values to the EVTG configuration structure.

  The purpose of this API is to initialize the configuration structure to default value for EVTG_Init() to use. The Flip-Flop can be configured as Bypass mode, RS trigger mode, T-FF mode, D-FF mode, JK-FF mode, Latch mode. Please check RM INTC chapter for more details.

Parameters

- psConfig – EVTG initial configuration structure pointer.
- flipflopMode – EVTG flip flop mode. see @ ref _evtg_flipflop_mode

static inline void EVTG_ForceFlipflopInitOutput(EVTG_Type *base, *evtg_index_t* evtgIndex, *evtg_flipflop_init_output_t* flipflopInitOutputValue)

Force Flip-flop initial output value to be presented on flip-flop positive output.

Parameters

- base – EVTG base address.
- evtgIndex – EVTG instance index.
- flipflopInitOutputValue – EVTG flip-flop initial output control. see evtg_flipflop_init_output_t

static inline void EVTG_SetProductTermInput(EVTG_Type *base, *evtg_index_t* evtgIndex, *evtg_aoi_index_t* aoiIndex, *evtg_aoi_product_term_t* productTerm, *evtg_input_index_t* inputIndex, *evtg_aoi_input_config_t* input)

Configure each input value of AOI product term. Each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. Adapt to some simple aoi expressions.

Parameters

- base – EVTG base address.
- evtgIndex – EVTG instance index.
- aoiIndex – EVTG AOI index. see enum ref evtg_aoi_index_t
- productTerm – EVTG product term index.
- inputIndex – EVTG input index.
- input – EVTG input configuration with enum evtg_aoi_input_config_t.

void EVTG_ConfigAOIProductTerm(EVTG_Type *base, *evtg_index_t* evtgIndex, *evtg_aoi_index_t* aoiIndex, *evtg_aoi_product_term_t* productTerm, *evtg_aoi_product_term_config_t* *psProductTermConfig)

Configure AOI product term by initializing the product term configuration structure.

Parameters

- base – EVTG base address.
- evtgIndex – EVTG instance index.
- aoiIndex – EVTG AOI index. see enum evtg_aoi_index_t
- productTerm – EVTG AOI product term index.
- psProductTermConfig – Pointer to EVTG product term configuration structure. see ref _evtg_aoi_product_term_config

2.17 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const *ewm_config_t* *config)

Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

ewm_config_t

Parameters

- config – Pointer to the EWM configuration structure.

static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to disable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

_ewm_status_flags_t

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)

Services the EWM.

This function resets the EWM counter to zero.

Parameters

- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION

EWM driver version 2.0.4.

enum _ewm_interrupt_enable_t

EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable

Enable the EWM to generate an interrupt

enum _ewm_status_flags_t

EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator kEWM_RunningFlag
Running flag, set when EWM is enabled

typedef struct *_ewm_config* ewm_config_t
Describes EWM clock source.

Data structure for EWM configuration.

This structure is used to configure the EWM.

struct _ewm_config
#include <fsl_ewm.h> Describes EWM clock source.

Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

bool enableEwm
Enable EWM module

bool enableEwmInput
Enable EWM_in input

bool setInputAssertLogic
EWM_in signal assertion state

bool enableInterrupt
Enable EWM interrupt

uint8_t compareLowValue
Compare low-register value

uint8_t compareHighValue
Compare high-register value

2.18 FGPIO Driver

2.19 Flash_driver

enum _flash_driver_version_constants
Flash driver version for ROM.

Values:

enumerator kFLASH_DriverVersionName
Flash driver version name.

enumerator kFLASH_DriverVersionMajor
Major flash driver version.

enumerator kFLASH_DriverVersionMinor
Minor flash driver version.

enumerator kFLASH_DriverVersionBugfix
Bugfix for flash driver version.

FSL_ROMAPI_FLASH_DRIVER_VERSION
ROMAPI_FLASH driver version 2.0.0.

Flash driver status codes.

Values:

enumerator kStatus_FLASH_Success
API is executed successfully

enumerator kStatus_FLASH_InvalidArgument
Invalid argument

enumerator kStatus_FLASH_SizeError
Error size

enumerator kStatus_FLASH_AlignmentError
Parameter is not aligned with the specified baseline

enumerator kStatus_FLASH_AddressError
Address is out of range

enumerator kStatus_FLASH_AccessError
Invalid instruction codes and out-of bound addresses

enumerator kStatus_FLASH_ProtectionViolation
The program/erase operation is requested to execute on protected areas

enumerator kStatus_FLASH_CommandFailure
Run-time error during command execution.

enumerator kStatus_FLASH_UnknownProperty
Unknown property.

enumerator kStatus_FLASH_EraseKeyError
API erase key is invalid.

enumerator kStatus_FLASH_RegionExecuteOnly
The current region is execute-only.

enumerator kStatus_FLASH_ExecuteInRamFunctionNotReady
Execute-in-RAM function is not available.

enumerator kStatus_FLASH_CommandNotSupported
Flash API is not supported.

enumerator kStatus_FLASH_ReadOnlyProperty
The flash property is read-only.

enumerator kStatus_FLASH_InvalidPropertyValue
The flash property value is out of range.

enumerator kStatus_FLASH_InvalidSpeculationOption
The option of flash prefetch speculation is invalid.

enumerator kStatus_FLASH_EccError
A correctable or uncorrectable error during command execution.

enumerator kStatus_FLASH_CompareError
Destination and source memory contents do not match.

enumerator kStatus_FLASH_RegulationLoss
A loss of regulation during read.

enumerator kStatus_FLASH_InvalidWaitStateCycles
The wait state cycle set to r/w mode is invalid.

enumerator kStatus_FLASH_OutOfDateCfpaPage
CFPA page version is out of date.

enumerator kStatus_FLASH_BlankIfrPageData
Blank page cannnot be read.

enumerator kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce
Encrypted flash subregions are not erased at once.

enumerator kStatus_FLASH_ProgramVerificationNotAllowed
Program verification is not allowed when the encryption is enabled.

enumerator kStatus_FLASH_HashCheckError
Hash check of page data is failed.

enumerator kStatus_FLASH_SealedFfrRegion
The FFR region is sealed.

enumerator kStatus_FLASH_FfrRegionWriteBroken
The FFR Spec region is not allowed to be written discontinuously.

enumerator kStatus_FLASH_NmpaAccessNotAllowed
The NMPA region is not allowed to be read/written/erased.

enumerator kStatus_FLASH_CmpaCfgDirectEraseNotAllowed
The CMPA Cfg region is not allowed to be erased directly.

enumerator kStatus_FLASH_FfrBankIsLocked
The FFR bank region is locked.

enumerator kStatus_FLASH_CfpaScratchPageInvalid
CFPA Scratch Page is invalid

enumerator kStatus_FLASH_CfpaVersionRollbackDisallowed
CFPA version rollback is not allowed

enumerator kStatus_FLASH_ReadHidingAreaDisallowed
Flash hiding read is not allowed

enumerator kStatus_FLASH_ModifyProtectedAreaDisallowed
Flash firewall page locked erase and program are not allowed

enumerator kStatus_FLASH_CommandOperationInProgress
The flash state is busy, indicate that a flash command in progress.

status_t FLASH_IsFlashAreaReadable(*flash_config_t* *config, uint32_t startAddress, uint32_t lengthInBytes)

Validates the given address range is loaded in the flash hiding region.

Parameters

- config – A pointer to the storage for the driver runtime state.
- startAddress – The start address of the desired flash memory to be verified.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed.

kStatusGroupGeneric

Flash driver status group.

kStatusGroupFlashDriver

MAKE_STATUS(group, code)

Constructs a status code value from a group and a code number.

enum _flash_driver_api_keys

Enumeration for Flash driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFLASH_ApiEraseKey

Key value used to validate all flash erase APIs.

FOUR_CHAR_CODE(a, b, c, d)

Constructs the four character code for the Flash driver API key.

status_t FLASH_Init(*flash_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FLASH_Deinit(*flash_config_t* *config)

De-Initializes the global flash properties structure members.

This API De-initializes the FLASH default parameters and related FLASH clock for the FLASH and FMC. The flash_deinit API should be called after all the other FLASH APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FLASH_Success – API was executed successfully.

- kStatus_FLASH_InvalidArgument – An invalid argument is provided.

status_t FLASH_Erase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. NOTE: The start address need to be 4 Bytes-aligned.
- lengthInBytes – The length, given in bytes need be 4 Bytes-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FLASH_AddressError – The address is out of range.
- kStatus_FLASH_EraseKeyError – The API erase key is invalid.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FLASH_Program(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.

- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_AddressError – Address is out of range.
- kStatus_FLASH_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.

status_t FLASH_Read(*flash_config_t* *config, uint32_t start, uint8_t *dest, uint32_t lengthInBytes)

Reads flash at locations passed in through parameters.

This function read the flash memory from a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be read.
- dest – A pointer to the dest buffer of data that is to be read from the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_AddressError – Address is out of range.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.

status_t FLASH_VerifyErase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.

- start – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- margin – Read margin choice.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FLASH_AddressError – Address is out of range.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.

status_t FLASH_VerifyProgram(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, uint32_t *failedAddress, uint32_t *failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- expectedData – A pointer to the expected data that is to be verified against.
- margin – Read margin choice.
- failedAddress – A pointer to the returned failing address.
- failedData – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FLASH_AddressError – Address is out of range.

- kStatus_FLASH_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.

status_t FLASH_GetProperty(*flash_config_t* *config, *flash_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flash property.

Parameters

- config – A pointer to the storage for the driver runtime state.
- whichProperty – The desired property from the list of properties in enum flash_property_tag_t
- value – A pointer to the value returned for the desired flash property.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_UnknownProperty – An unknown property tag.

status_t FLASH_GetCustKeyStore(*flash_config_t* *config, uint8_t *pData, uint32_t offset, uint32_t len)

Get the customer key store data from the customer key store region .

Parameters

- config – Pointer to flash_config_t data structure in memory to store driver runtime state.
- pData – Pointer to the customer key store data buffer, which got from the customer key store region.
- offset – Point to the offset value based on the customer key store address(0x3e400) of the device.
- len – Point to the length of the expected get customer key store data, and the offset + len <= 512B.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.

MAKE_VERSION(major, minor, bugfix)

Constructs the version number for drivers.

FSL_FEATURE_SYSCON_HAS_FLASH_HIDING

enum _flash_property_tag

Enumeration for various flash properties.

Values:

enumerator kFLASH_PropertyPflashSectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflashTotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflashBlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflashBlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflashBlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflashPageSize
Pflash page size property.

enumerator kFLASH_PropertyPflashSystemFreq
System Frequency System Frequency.

enumerator kFLASH_PropertyFfrSectorSize
FFR sector size property.

enumerator kFLASH_PropertyFfrTotalSize
FFR total size property.

enumerator kFLASH_PropertyFfrBlockBaseAddr
FFR block base address property.

enumerator kFLASH_PropertyFfrPageSize
FFR page size property.

enum _flash_max_erase_page_value
Enumeration for flash max pages to erase.

Values:

enumerator kFLASH_MaxPagesToErase
The max value in pages to erase.

enum _flash_alignment_property
Enumeration for flash alignment property.

Values:

enumerator kFLASH_AlignementUnitVerifyErase
The alignment unit in bytes used for verify erase operation.

enumerator kFLASH_AlignementUnitProgram
The alignment unit in bytes used for program operation.

enumerator kFLASH_AlignementUnitSingleWordRead
The alignment unit in bytes used for verify program operation. The alignment unit in bytes used for SingleWordRead command.

enum _flash_read_ecc_option
Enumeration for flash read ecc option.

Values:

enumerator kFLASH_ReadWithEccOn

enumerator kFLASH_ReadWithEccOff
ECC is on

enum _flash_read_margin_option
Enumeration for flash read margin option.

Values:

enumerator kFLASH_ReadMarginNormal
Normal read

enumerator kFLASH_ReadMarginVsProgram
Margin vs. program

enumerator kFLASH_ReadMarginVsErase
Margin vs. erase

enumerator kFLASH_ReadMarginIllegalBitCombination
Illegal bit combination

enum _flash_read_dmacc_option
Enumeration for flash read dmacc option.

Values:

enumerator kFLASH_ReadDmaccDisabled
Memory word

enumerator kFLASH_ReadDmaccEnabled
DMACC word

enum _flash_ramp_control_option
Enumeration for flash ramp control option.

Values:

enumerator kFLASH_RampControlDivisionFactorReserved
Reserved

enumerator kFLASH_RampControlDivisionFactor256
clk48mhz / 256 = 187.5KHz

enumerator kFLASH_RampControlDivisionFactor128
clk48mhz / 128 = 375KHz

enumerator kFLASH_RampControlDivisionFactor64
clk48mhz / 64 = 750KHz

typedef enum *_flash_property_tag* flash_property_tag_t
Enumeration for various flash properties.

typedef struct *_flash_ecc_log* flash_ecc_log_t
Flash ECC log info.

typedef struct *_flash_mode_config* flash_mode_config_t
Flash controller paramter config.

typedef struct *_flash_ffr_config* flash_ffr_config_t
Flash controller paramter config.

struct _flash_ecc_log
#include <fsl_flash.h> Flash ECC log info.

struct _flash_mode_config

#include <fsl_flash.h> Flash controller paramter config.

struct _flash_ffr_config

#include <fsl_flash.h> Flash controller paramter config.

struct flash_config_t

#include <fsl_flash.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Public Members

uint32_t PFlashBlockBase

A base address of the first PFlash block

uint32_t PFlashTotalSize

The size of the combined PFlash block.

uint32_t PFlashBlockCount

A number of PFlash blocks.

uint32_t PFlashPageSize

The size in bytes of a page of PFlash.

uint32_t PFlashSectorSize

The size in bytes of a sector of PFlash.

struct readSingleWord

struct setWriteMode

struct setReadMode

2.20 Flash_ffr_driver

status_t FFR_Init(*flash_config_t* *config)

Initializes the global FFR properties structure members.

Parameters

- config – A pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success – API was executed successfully.

status_t FFR_Lock(*flash_config_t* *config)

Enable firewall for all flash banks.

CFPA, CMPA, and NMPA flash areas region will be locked, After this function executed; Unless the board is reset again.

Parameters

- config – A pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success – An invalid argument is provided.

status_t FFR_InfieldPageWrite(*flash_config_t* *config, uint8_t *page_data, uint32_t valid_len)

APIs to access CFPA pages.

This routine will erase CFPA and program the CFPA page with passed data.

Parameters

- config – A pointer to the storage for the driver runtime state.
- page_data – A pointer to the source buffer of data that is to be programmed into the CFPA.
- valid_len – The length, given in bytes, to be programmed.

Return values

- kStatus_FLASH_Success – The desire page-data were programed successfully into CFPA.
- kStatus_FLASH_SizeError – Error size
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- #kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FLASH_FfrBankIsLocked – The CFPA was locked.
- kStatus_FLASH_OutOfDateCfpaPage – It is not newest CFPA page.
- kStatus_FLASH_CommandFailure – access error.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FFR_GetCustomerInfieldData(*flash_config_t* *config, uint8_t *pData, uint32_t offset, uint32_t len)

APIs to access CFPA pages.

Generic read function, used by customer to read data stored in 'Customer In-field Page'.

Parameters

- config – A pointer to the storage for the driver runtime state.
- pData – A pointer to the dest buffer of data that is to be read from 'Customer In-field Page'.
- offset – An offset from the 'Customer In-field Page' start address.
- len – The length, given in bytes, to be read.

Return values

- kStatus_FLASH_Success – Get data from 'Customer In-field Page'.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- #kStatus_FTFx_AddressError – Address is out of range.

- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed
- kStatus_FLASH_CommandFailure – access error.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FFR_CustFactoryPageWrite(*flash_config_t* *config, uint8_t *page_data, bool seal_part)

APIs to access CMPA pages.

This routine will erase "customer factory page" and program the page with passed data. If 'seal_part' parameter is TRUE then the routine will compute SHA256 hash of the page contents and then programs the pages. 1.During development customer code uses this API with 'seal_part' set to FALSE. 2.During manufacturing this parameter should be set to TRUE to seal the part from further modifications 3.This routine checks if the page is sealed or not. A page is said to be sealed if the SHA256 value in the page has non-zero value. On boot ROM locks the firewall for the region if hash is programmed anyways. So, write/erase commands will fail eventually.

Parameters

- config – A pointer to the storage for the driver runtime state.
- page_data – A pointer to the source buffer of data that is to be programmed into the "customer factory page".
- seal_part – Set fasle for During development customer code.

Return values

- kStatus_FLASH_Success – The desire page-data were programed successfully into CMPA.
- kStatus_FLASH_InvalidArgument – Parameter is not aligned with the specified baseline.
- #kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_EraseKeyError – API erase key is invalid.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed
- kStatus_Fail – Generic status for Fail.
- kStatus_FLASH_CommandFailure – access error.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FFR_GetCustomerData(*flash_config_t* *config, uint8_t *pData, uint32_t offset, uint32_t len)

APIs to access CMPA page.

Read data stored in 'Customer Factory CFG Page'.

Parameters

- config – A pointer to the storage for the driver runtime state.
- pData – A pointer to the dest buffer of data that is to be read from the Customer Factory CFG Page.
- offset – Address offset relative to the CMPA area.
- len – The length, given in bytes to be read.

Return values

- kStatus_FLASH_Success – Get data from 'Customer Factory CFG Page'.
- kStatus_FLASH_InvalidArgument – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- #kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FLASH_CommandFailure – access error.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed

status_t FFR_GetCustKeystoreData(*flash_config_t* *config, uint8_t *pData, uint32_t offset, uint32_t len)

The API is used for getting the customer key store data from the customer key store region(0x3e400 C 0x3e600), and the API should be called after the FLASH_Init and FFR_Init.

Parameters

- config – A pointer to the storage for the driver runtime state.
- pData – A pointer to the dest buffer of data that is to be read from the Customer Factory CFG Page.
- offset – Address offset relative to the CMPA area.
- len – The length, given in bytes to be read.

Return values

- kStatus_FLASH_Success – Get data from 'Customer Factory CFG Page'.
- kStatus_FLASH_InvalidArgument – Parameter is not aligned with the specified baseline.
- #kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FLASH_AddressError – Address is out of range
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed
- kStatus_FLASH_CommandFailure – access error.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported

- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FFR_CustKeystoreWrite(*flash_config_t* *config, *ffr_key_store_t* *pKeyStore)

This routine writes the 3 pages allocated for Key store data.

Parameters

- config – A pointer to the storage for the driver runtime state.
- pKeyStore – A pointer to the source buffer of data that is to be programmed into the "Key store".

Return values

- kStatus_FLASH_Success – Get data from 'Customer Factory CFG Page'.
- kStatus_FLASH_InvalidArgument – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_CommandFailure – access error.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_SealedFfrRegion – The FFR region is sealed.
- kStatus_FLASH_FfrBankIsLocked – The FFR bank region is locked.
- kStatus_FLASH_AddressError – Address is out of range
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed

status_t FFR_GetUUID(*flash_config_t* *config, uint8_t *uuid)

APIs to access CMPA page.

1.SW should use this API routine to get the UUID of the chip. 2.Calling routine should pass a pointer to buffer which can hold 128-bit value.

Return values

- kStatus_FLASH_Success – Get data from 'Customer Factory CFG Page'.
- kStatus_FLASH_InvalidArgument – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed
- kStatus_FLASH_CommandFailure – Run-time error during command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

enum flash_ffr_page_offset

Values:

enumerator kFfrPageOffset_CFPA

Customer In-Field programmed area

enumerator kFfrPageOffset_CFPA_CfgPing
CFPA Configuration area (Ping page)

enumerator kFfrPageOffset_CFPA_CfgPong
Same as CFPA page (Pong page)

enumerator kFfrPageOffset_CMPA_Cfg
Customer Manufacturing programmed area

enumerator kFfrPageOffset_NMPA_Cfg
Customer Manufacturing programmed area

enumerator kFfrPageOffset_SBL_Cfg
SBL recovery programmed area

enumerator kFfrPageOffset_B0_IFR1_Visible
Trim programmed area

enum flash_ffr_page_num
Values:

enumerator kFfrSectorNum_CFPA
Customer In-Field programmed area

enumerator kFfrSectorNum_CMPA
Customer Manufacturing programmed area

enumerator kFfrSectorNum_NMPA
NXP Manufacturing programmed area

enumerator kFfrSectorNum_SBL
SBL Cus programmed area

enumerator kFfrSectorNum_Total

enum flash_ffr_block_size
Values:

enumerator kFfrBlockSize_Key

enumerator kFfrBlockSize_ActivationCode

enum cfpa_cfg_cmpa_prog_status
Values:

enumerator kFfrCmpaProgStatus_Idle

enumerator kFfrCmpaProgStatus_InProgress

enum cmpa_prog_process_t
Values:

enumerator kFfrCmpaProgProcess_Pre

enumerator kFfrCmpaProgProcess_Post

enum ffr_key_type_t
Values:

enumerator kFFR_KeyTypeSbkek

enumerator kFFR_KeyTypeUser

enumerator kFFR_KeyTypeUds

enumerator kFFR_KeyTypePrinceRegion0

enumerator kFFR_KeyTypePrinceRegion1

enumerator kFFR_KeyTypePrinceRegion2

enum ffr_bank_type_t

Values:

enumerator kFFR_BankTypeBank0_CFPA0

enumerator kFFR_BankTypeBank0_CFPA1

enumerator kFFR_BankTypeBank0_CMPA

enumerator kFFR_BankTypeBank0_NMPA

enumerator kFFR_BankTypeBank0_SBL

ALIGN_DOWN(x, a)

Alignment(down) utility.

ALIGN_UP(x, a)

Alignment(up) utility.

FLASH_FFR_MAX_PAGE_SIZE

FLASH_FFR_CUST_ADDRESS

FLASH_FFR_CUST_PAGE_NUMBER

FLASH_FFR_HASH_DIGEST_SIZE

FLASH_FFR_IV_CODE_SIZE

FLASH_FFR_KETBLOB_OFFSET

FLASH_FFR_KETBLOB_SIZE

CFPA_HEADER_MARKER

CMPA_HEADER_MARKER

FLASH_FFR_UUID_SIZE

FFR_BOOTCFG_USBSPEED_SHIFT

FFR_BOOTCFG_USBSPEED_MASK

FFR_BOOTCFG_USBSPEED_NMPASEL0

FFR_BOOTCFG_USBSPEED_FS

FFR_BOOTCFG_USBSPEED_HS

FFR_BOOTCFG_USBSPEED_NMPASEL3

FFR_BOOTCFG_BOOTSPEED_MASK

FFR_BOOTCFG_BOOTSPEED_SHIFT

FFR_BOOTCFG_BOOTSPEED_NMPASEL

FFR_BOOTCFG_BOOTSPEED_48MHZ

FFR_BOOTCFG_BOOTSPEED_96MHZ

FFR_USBID_VENDORID_MASK

FFR_USBID_VENDORID_SHIFT

FFR_USBID_PRODUCTID_MASK

FFR_USBID_PRODUCTID_SHIFT

FFR_IMAGE0_CMAC_UPDATE_MASK

FFR_IMAGE1_CMAC_UPDATE_MASK

FFR_IFR1_PUF_AC_CODE_ADDR

FFR_IFR1_PUF_AC_CODE_LEN

FFR_IFR1_NXP_CERT_ADDR

FFR_IFR1_NXP_CERT_LEN

FFR_IFR1_ROM_PATCH_ARRAY0_ADDR

FFR_IFR1_ROM_PATCH_ARRAY0_LEN

FFR_IFR1_ROM_PATCH_ARRAY1_ADDR

FFR_IFR1_ROM_PATCH_ARRAY1_LEN

FFR_IFR1_ROM_PATCH_ARRAY2_ADDR

FFR_IFR1_ROM_PATCH_ARRAY2_LEN

FFR_IFR1_ROM_PATCH_ARRAY3_ADDR

FFR_IFR1_ROM_PATCH_ARRAY3_LEN

FFR_IFR1_NXP_WRITEABLE_REGION0_START

FFR_IFR1_NXP_WRITEABLE_REGION0_END

FFR_IFR1_NXP_WRITEABLE_REGION1_START

FFR_IFR1_NXP_WRITEABLE_REGION1_END

FFR_IFR1_NXP_WRITEABLE_REGION2_START

FFR_IFR1_NXP_WRITEABLE_REGION2_END

FFR_SYSTEM_SPEED_CODE_MASK

FFR_SYSTEM_SPEED_CODE_SHIFT

FFR_SYSTEM_SPEED_CODE_FRO12MHZ_12MHZ

FFR_SYSTEM_SPEED_CODE_FROHF96MHZ_24MHZ

FFR_SYSTEM_SPEED_CODE_FROHF96MHZ_48MHZ

FFR_SYSTEM_SPEED_CODE_FROHF96MHZ_96MHZ

FFR_USBCFG_USBSPEED_HS

FFR_USBCFG_USBSPEED_FS

FFR_USBCFG_USBSPEED_NO

FFR_MCAN_BAUDRATE_MASK

FFR_MCAN_BAUDRATE_SHIFT

FFR_PERIPHERALCFG_PERI_MASK

FFR_PERIPHERALCFG_PERI_SHIFT

FFR_PERIPHERALCFG_COREEN_MASK

FFR_PERIPHERALCFG_COREEN_SHIFT

FFR_PUF_SRAM_CONFIG_MASK

FFR_PUF_SRAM_CONFIG_MASK_SHIFT

FFR_PUF_SRAM_VALID_MASK

FFR_PUF_SRAM_VALID_SHIFT

FFR_PUF_SRAM_MODE_MASK

FFR_PUF_SRAM_MODE_SHIFT

FFR_PUF_SRAM_CKGATING_MASK

FFR_PUF_SRAM_CKGATING_SHIFT

FFR_PUF_SRAM_SMB_MASK

FFR_PUF_SRAM_SMB_SHIFT

FFR_PUF_SRAM_RM_MASK

FFR_PUF_SRAM_RM_SHIFT

FFR_PUF_SRAM_WM_MASK

FFR_PUF_SRAM_WM_SHIFT

FFR_PUF_SRAM_WRME_MASK

FFR_PUF_SRAM_WRME_SHIFT

FFR_PUF_SRAM_RAEN_MASK

FFR_PUF_SRAM_RAEN_SHIFT

FFR_PUF_SRAM_RAM_MASK

FFR_PUF_SRAM_RAM_SHIFT

FFR_PUF_SRAM_WAEN_MASK

FFR_PUF_SRAM_WAEN_SHIFT

FFR_PUF_SRAM_WAM_MASK

FFR_PUF_SRAM_WAM_SHIFT

FFR_PUF_SRAM_STBP_MASK

FFR_PUF_SRAM_STBP_SHIFT

struct cfpa_cfg_info_t

#include <fsl_flash_ffr.h>

Public Members

struct *cfpa_cfg_info_t* header
[0x000-0x003]

uint32_t secureFwVersion
[0x008-0x00b]

uint32_t nsFwVersion
[0x00c-0x00f]

uint32_t recFwVersion
[0x010-0x013]

uint32_t secBootFlags
[0x014-0x01f]

uint32_t imageKeyRevoke
[0x018-0x01b]

uint32_t lpVectorAddr
[0x01c-0x01f]

uint32_t vendorUsage
[0x020-0x02f]

uint32_t dcfgNsPin
[0x024-0x027]

uint32_t dcfgNsDflt
[0x028-0x02b]

uint32_t reserved0
[0x02c-0x02f]

uint32_t ivPrince[4]
[0x030-0x03f]

uint32_t ivIped[8]
[0x040-0x05f]

uint32_t errCnt[8]
[0x060-0x07f]

uint32_t custCtr[8]
[0x080-0x09f]

uint32_t mflagCtr[8]
[0x0a0-0x0bf]

uint32_t flashAcl[8]
[0x0C0-0x0Df]

uint32_t sblImg0Cmac[4]
[0x0E0-0x0Ef]

uint32_t img1Cmac[4]
[0x0F0-0x0Ff]

uint32_t diceCert[36]
[0x100-0x18f]

uint32_t reserved2[23]
[0x190-0x1eb]

uint32_t cfpaCrc
[0x1ec-0x1ef]

uint32_t cfpaCmac[4]
[0x1f0-0x1ff]

struct cmpa_cfg_info_t
#include <fsl_flash_ffr.h>

Public Members

struct *cmpa_cfg_info_t* bootCfg
[0x000-0x003]

struct *cmpa_cfg_info_t* FlashCfg
[0x004-0x007]

struct *cmpa_cfg_info_t* bootLedStatus
[0x008-0x00b]

struct *cmpa_cfg_info_t* bootTimers
[0x00c-0x00f]

uint32_t resv2
[0x010-0x013]

uint32_t resv3
[0x014-0x017]

uint32_t recSpiFlashCfg0
[0x018-0x01b]

uint32_t recSpiFlashCfg1
[0x01c-0x01f]

uint32_t isp_uart_cfg
[0x020-0x023]

uint32_t isp_i2c_cfg
[0x024-0x027]

uint32_t isp_can_cfg
[0x028-0x02b]

uint32_t isp_spi_cfg0
[0x02c-0x02f]

uint32_t isp_spi_cfg1
[0x030-0x034]

struct *cmpa_cfg_info_t* usbId
[0x034-0x037]

uint32_t isp_usb_cfg
[0x038-0x038]

uint32_t isp_misc_cfg
[0x03c-0x03f]

uint32_t dcfgPin
[0x040-0x043]

uint32_t dcfgDflt
[0x044-0x047]

uint32_t dapVendorUsage
[0x048-0x04b]

uint32_t resv1
[0x04c-0x04f]

uint32_t secureBootCfg
[0x050-0x053]

uint32_t rokthUsage
[0x054-0x057]

uint32_t resv4
[0x058-0x05b]

uint32_t resv5
[0x05c-0x05f]

uint32_t rotkh[12]
[0x060-0x08f]

struct *cmpa_cfg_info_t* princeSr[4]
[0x090-0x0af]

struct *cmpa_cfg_info_t* ipedRegions[8]
[0x0b0-0x11f]

struct *cmpa_cfg_info_t* quickSetGpio[6]
[0x120-0x14f]

uint32_t resv7[4]
[0x150-0x15f]

uint32_t cust_key_blob[12]
[0x160-0x18f]

uint32_t resv8[23]
[0x190-0x1eb]

uint32_t cmpaCrc
[0x1ec-0x1ef]

uint32_t cmpaCmac[4]
[0x1f0-0x1ff]

struct cmpa_key_store_header_t
#include <fsl_flash_ffr.h>

struct nmpa_cfg_info_t
#include <fsl_flash_ffr.h>

Public Members

uint32_t fro32kCfg
[0x000-0x003]

uint32_t puf_cfg
[0x004-0x007]

uint32_t bod
[0x008-0x00b]

uint32_t trim
[0x00c-0x00f]

uint32_t deviceID
[0x010-0x03f]

uint32_t peripheralCfg
[0x014-0x017]

uint32_t dcdPowerProFileLOW[2]
[0x018-0x01f]

uint32_t deviceType
[0x020-0x023]

uint32_t ldo_ao
[0x024-0x027]

uint32_t gdetDelayCfg
[0x028-0x02b]

uint32_t gdetMargin
[0x02c-0x02f]

uint32_t gdetTrim1
[0x030-0x033]

uint32_t gdetEanble1
[0x034-0x037]

uint32_t gdetCtrl1
[0x038-0x03b]

uint32_t gdetUpdateTimer
[0x03c-0x03f]

uint32_t GpoDataChecksum[4]
[0x040-0x04f]

uint32_t finalTestBatchId[4]
[0x050-0x05f]

uint32_t ecidBackup[4]
[0x060-0x06f]

uint32_t uuid[4]
[0x070-0x07f]

uint32_t reserved1[7]
[0x080-0x09b]

struct *nmpa_cfg_info_t* usbCfg
[0x09c-0x09f]

uint32_t reserved2[80]
[0x0a0-0x1df]

uint8_t cmac[16]
[0x1e0-0x1ef]

uint32_t pageChecksum[4]
[0x1f0-0x1ff]

struct ffr_key_store_t
#include <fsl_flash_ffr.h>

struct header

struct cfpa_page_version

Public Members

uint32_t version
cfpa version

uint32_t img_upd
image cmac update

uint32_t cmpa_update
CFPA page updated through SB command.

uint32_t dice_en
Update DICE certificate during next boot.

struct bootCfg

struct FlashCfg

struct bootLedStatus

struct bootTimers

struct usbId

struct princeSr

struct ipedRegions

struct quickSetGpio

struct usbCfg

2.21 FlexCAN: Flex Controller Area Network Driver

2.22 FlexCAN Driver

bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)
Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN_Init function needs to be executed first, and then call this

API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN_FDInit to enable CAN FD mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)

Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080. For 32 bytes payload frame, the second mailbox is ((1/12)*512 + 1%12*40)/4 = 10, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – Mailbox index.

Returns
Mailbox address offset in word.

status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

uint32_t FLEXCAN_GetInstance(CAN_Type *base)

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns
FlexCAN instance.

bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

void FLEXCAN_Init(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                 = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate                = 1000000U;
flexcanConfig.maxMbNum               = 16;
flexcanConfig.enableLoopBack         = false;
flexcanConfig.enableSelfWakeup       = false;
flexcanConfig.enableIndividMask      = false;
flexcanConfig.enableDoze             = false;
flexcanConfig.disableSelfReception   = false;
flexcanConfig.enableListenOnlyMode   = false;
flexcanConfig.timingConfig           = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 40000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t bitRateFD, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.

- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

void FLEXCAN_FDInit(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz, *flexcan_mb_size_t* dataSize, bool brs)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                 = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate                = 1000000U;
flexcanConfig.bitRateFD              = 2000000U;
flexcanConfig.maxMbNum               = 16;
flexcanConfig.enableLoopBack         = false;
flexcanConfig.enableSelfWakeup       = false;
flexcanConfig.enableIndividMask      = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze             = false;
flexcanConfig.timingConfig           = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 80000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

void FLEXCAN_Deinit(CAN_Type *base)

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN_GetDefaultConfig(*flexcan_config_t* *pConfig)

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig-

>enablePretendedeNetworking = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian; flexcanConfig.timingConfig = timingConfig;

Parameters

- pConfig – Pointer to the FlexCAN configuration structure.

void FLEXCAN_SetTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetBitRate() instead.

Note: Calling FLEXCAN_SetTimingConfig() overrides the bit rate set in FLEXCAN_Init() or FLEXCAN_SetBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on FLEXCAN_CalculateImprovedTimingValues() API calculated timing values.

Note: Calling FLEXCAN_SetBitRate() overrides the bit rate set in FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRate_Bps – Bit rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only Nominal phase) successfully.

void FLEXCAN_SetFDTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetFDBitRate() to set both Nominal/Data bit Rate instead.

Note: Calling FLEXCAN_SetFDTimingConfig() overrides the data phase bit rate set in FLEXCAN_FDInit()/FLEXCAN_SetFDBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRateN_Bps, uint32_t bitRateD_Bps)

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on FLEXCAN_FDCalculateImprovedTimingValues() API calculated timing values.

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRateN_Bps – Nominal bit Rate in Bps.
- bitRateD_Bps – Data bit Rate in Bps.

Returns

kStatus_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN_SetTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbEmpty, kFLEXCAN_RxMbFull or kFLEXCAN_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pFrame)

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const *flexcan_rx_fifo_config_t* *pRxFifoConfig, bool enable)

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note: Legacy Rx FIFO only can receive classic CAN message.

Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.

- enable – Enable/disable Legacy Rx FIFO.
  - true: Enable Legacy Rx FIFO.
  - false: Disable Legacy Rx FIFO.

void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const *flexcan_enhanced_rx_fifo_config_t* *pConfig, bool enable)

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

Note: Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
  - true: Enable Enhanced Rx FIFO.
  - false: Disable Enhanced Rx FIFO.

void FLEXCAN_SetPNConfig(CAN_Type *base, const *flexcan_pn_config_t* *pConfig)

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators _flexcan_flags. To check the specific status, compare the return value with enumerators in _flexcan_flags.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the _flexcan_flags.

static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of _flexcan_flags.

static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base, *flexcan_memory_error_report_status_t* *errorStatus)

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of received wake up msessages.

static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of _flexcan_interrupt_enable.

static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to disable. Logical OR of _flexcan_interrupt_enable.

static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- enable – true to enable, false to disable.

static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN Rx FIFO Head address.

static inline *status_t* FLEXCAN_Enable(CAN_Type *base, bool enable)

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

Parameters

- base – FlexCAN base pointer.
- enable – true to enable, false to disable.

Returns

kStatus_Success Enable FlexCAN module successful kStatus_Timeout Timeout when wait for Low-Power Mode Acknowledge

status_t FLEXCAN_WriteTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pTxFrame)

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.

status_t FLEXCAN_WriteFDTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_fd_frame_t* *pTxFrame)

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadFDRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.

status_t FLEXCAN_ReadRxFifo(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadEnhancedRxFifo(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadPNWakeUpMB(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

Parameters

- base – FlexCAN peripheral base address.

- pRxFrame – Pointer to CAN message frame structure for reception.
- mbIdx – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

Return values

- kStatus_Success – - Read Message from Wake up Message Buffer successfully.
- kStatus_Fail – - Wake up Message Buffer has no valid content.

status_t FLEXCAN_TransferFDSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferFDReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferFDSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferFDReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

void FLEXCAN_TransferFDAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

void FLEXCAN_TransferFDAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

status_t FLEXCAN_TransferSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveFifoBlocking(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveEnhancedFifoBlocking(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

void FLEXCAN_TransferCreateHandle(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_transfer_callback_t* callback, void *userData)

Initializes the FlexCAN handle.

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

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t FLEXCAN_TransferSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the flexcan_fifo_transfer_t.

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

status_t FLEXCAN_TransferGetReceiveFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

status_t FLEXCAN_TransferReceiveEnhancedFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the ref flexcan_fifo_transfer_t.@

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

uint32_t FLEXCAN_GetTimeStamp(*flexcan_handle_t* *handle, uint8_t mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

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_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).

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_WakeUpInterruptEnable
Self Wake Up interrupt, use bit 26.

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_WakeUpIntFlag
Self Wake-Up 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_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).

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 enableSelfWakeup
: Enable or Disable Self Wakeup Mode.

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.

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 ___unnamed75___

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 ___unnamed77___

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 ___unnamed79___

Public Members

struct _flexcan_frame

struct _flexcan_frame

struct ___unnamed81___

Public Members

uint32_t dataWord0
CAN Frame payload word0.

uint32_t dataWord1
CAN Frame payload word1.

struct ___unnamed83___

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 ___unnamed85___

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 ___unnamed87___

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 ___unnamed89___

Public Members

struct _flexcan_fd_frame

struct _flexcan_fd_frame

struct ___unnamed91___

Public Members

uint32_t dataWord[16]

CAN FD Frame payload, 16 double word maximum.

struct ___unnamed93___

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 ___unnamed95___

Public Members

struct _flexcan_config

struct _flexcan_config

struct ___unnamed97___

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 ___unnamed99___

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 ___unnamed101___

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 ___unnamed105___

< 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 ___unnamed107___

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 ___unnamed103___

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 ___unnamed109___

< 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 ___unnamed111___

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.

2.23 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.

2.24 FlexIO: FlexIO Driver

2.25 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,
```

(continues on next page)

(continued from previous page)

```
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const *flexio_timer_config_t* *timerConfig)

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_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

2.26 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

2.27 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

2.28 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

2.29 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)
Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
```

(continues on next page)

(continued from previous page)

```
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND _flexio_i2c_master_status_flags can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_I2C_RxFullFlag
  - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t* direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
  - kFLEXIO_I2C_Write: Transmit
  - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns
data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns
status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

status_t FLEXIO_I2C_MasterTransferGetCount(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.

- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_I2C_MasterTransferAbort(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak

NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout

Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable

Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable

Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag
Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag
Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag
Receive NAK flag.

enum _flexio_i2c_direction
Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write
Master send to slave.

enumerator kFLEXIO_I2C_Read
Master receive from slave.

typedef enum *_flexio_i2c_direction* flexio_i2c_direction_t
Direction of master transfer.

typedef struct *_flexio_i2c_type* FLEXIO_I2C_Type
Define FlexIO I2C master access structure typedef.

typedef struct *_flexio_i2c_master_config* flexio_i2c_master_config_t
Define FlexIO I2C master user configuration structure.

typedef struct *_flexio_i2c_master_transfer* flexio_i2c_master_transfer_t
Define FlexIO I2C master transfer structure.

typedef struct *_flexio_i2c_master_handle* flexio_i2c_master_handle_t
FlexIO I2C master handle typedef.

typedef void (*flexio_i2c_master_transfer_callback_t)(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *status_t* status, void *userData)
FlexIO I2C master transfer callback typedef.

I2C_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_i2c_type
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t SDAPinIndex
Pin select for I2C SDA.

uint8_t SCLPinIndex
Pin select for I2C SCL.

uint8_t shifterIndex[2]
Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]
Timer index used in FlexIO I2C.

uint32_t baudrate
Master transfer baudrate, used to calculate delay time.

struct _flexio_i2c_master_config
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster
Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze
Enable/disable FlexIO operation in doze mode.

bool enableInDebug
Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
Baud rate in Bps.

struct _flexio_i2c_master_transfer
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags
Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress
7-bit slave address.

flexio_i2c_direction_t direction
Transfer direction, read or write.

uint32_t subaddress
Sub address. Transferred MSB first.

uint8_t subaddressSize
Size of sub address.

uint8_t volatile *data
Transfer buffer.

volatile size_t dataSize
Transfer size.

struct _flexio_i2c_master_handle
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer
FlexIO I2C master transfer copy.

size_t transferSize
Total bytes to be transferred.

uint8_t state
Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback
Callback function called at transfer event. Callback function called at transfer event.

void *userData
Callback parameter passed to callback function.

bool needRestart
Whether master needs to send re-start signal.

2.30 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

2.31 FlexIO SPI Driver

void FLEXIO_SPI_MasterInit(*FLEXIO_SPI_Type* *base, *flexio_spi_master_config_t* *masterConfig, uint32_t srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
```

(continues on next page)

(continued from previous page)

```
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 3*2=6. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
```

(continues on next page)

(continued from previous page)

```
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- mask – status flag The parameter can be any combination of the following values:
  - kFLEXIO_SPI_TxEmptyFlag
  - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_DisableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_EnableDMA(*FLEXIO_SPI_Type* *base, uint32_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

static inline void FLEXIO_SPI_Enable(*FLEXIO_SPI_Type* *base, bool enable)

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.
- enable – True to enable, false does not have any effect.

void FLEXIO_SPI_MasterSetBaudRate(*FLEXIO_SPI_Type* *base, uint32_t baudRate_Bps, uint32_t srcClockHz)

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

static inline void FLEXIO_SPI_WriteData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint32_t data)

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

static inline uint32_t FLEXIO_SPI_ReadData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- direction – Shift direction of MSB first or LSB first.

Returns
8 bit/16 bit data received.

status_t FLEXIO_SPI_WriteBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, const uint8_t *buffer, size_t size)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint8_t *buffer, size_t size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully create the handle.

- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle)

Aborts the master data transfer, which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_SlaveTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *flexio_spi_slave_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_SlaveTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle)

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI slave IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

FSL_FLEXIO_SPI_DRIVER_VERSION

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator kStatus_FLEXIO_SPI_Busy

FlexIO SPI is busy.

enumerator kStatus_FLEXIO_SPI_Idle

SPI is idle

enumerator kStatus_FLEXIO_SPI_Error

FlexIO SPI error.

enumerator kStatus_FLEXIO_SPI_Timeout

FlexIO SPI timeout polling status flags.

enum _flexio_spi_clock_phase

FlexIO SPI clock phase configuration.

Values:

enumerator kFLEXIO_SPI_ClockPhaseFirstEdge

First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge

First edge on SPSCK occurs at the start of the first cycle of a data transfer.

enum _flexio_spi_shift_direction

FlexIO SPI data shifter direction options.

Values:

enumerator kFLEXIO_SPI_MsbFirst

Data transfers start with most significant bit.

enumerator kFLEXIO_SPI_LsbFirst

Data transfers start with least significant bit.

enum _flexio_spi_data_bitcount_mode

FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode

8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode

16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode

32-bit data transmission mode.

enum _flexio_spi_interrupt_enable

Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable

Receive buffer full interrupt enable.

enum _flexio_spi_status_flags

Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag

Receive buffer full flag.

enum _flexio_spi_dma_enable

Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable

Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable
Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable
All DMA request source

enum _flexio_spi_transfer_flags
Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb
FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb
FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb
FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb
FlexIO SPI 16-bit LSB first

enumerator kFLEXIO_SPI_32bitMsb
FlexIO SPI 32-bit MSB first

enumerator kFLEXIO_SPI_32bitLsb
FlexIO SPI 32-bit LSB first

enumerator kFLEXIO_SPI_csContinuous
Enable the CS signal continuous mode

typedef enum *_flexio_spi_clock_phase* flexio_spi_clock_phase_t
FlexIO SPI clock phase configuration.

typedef enum *_flexio_spi_shift_direction* flexio_spi_shift_direction_t
FlexIO SPI data shifter direction options.

typedef enum *_flexio_spi_data_bitcount_mode* flexio_spi_data_bitcount_mode_t
FlexIO SPI data length mode options.

typedef struct *_flexio_spi_type* FLEXIO_SPI_Type
Define FlexIO SPI access structure typedef.

typedef struct *_flexio_spi_master_config* flexio_spi_master_config_t
Define FlexIO SPI master configuration structure.

typedef struct *_flexio_spi_slave_config* flexio_spi_slave_config_t
Define FlexIO SPI slave configuration structure.

typedef struct *_flexio_spi_transfer* flexio_spi_transfer_t
Define FlexIO SPI transfer structure.

typedef struct *_flexio_spi_master_handle* flexio_spi_master_handle_t
typedef for flexio_spi_master_handle_t in advance.

typedef *flexio_spi_master_handle_t* flexio_spi_slave_handle_t
Slave handle is the same with master handle.

typedef void (*flexio_spi_master_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI slave callback for finished transmit.

FLEXIO_SPI_DUMMYDATA

FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

SPI_RETRY_TIMES

Retry times for waiting flag.

FLEXIO_SPI_XFER_DATA_FORMAT(flag)

Get the transfer data format of width and bit order.

struct _flexio_spi_type

#include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

uint8_t SDOPinIndex

Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex

Pin select for data input.

uint8_t SCKPinIndex

Pin select for clock.

uint8_t CSnPinIndex

Pin select for enable.

uint8_t shifterIndex[2]

Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]

Timer index used in FlexIO SPI.

struct _flexio_spi_master_config

#include <fsl_flexio_spi.h> Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster

Enable/disable FlexIO SPI master after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct _flexio_spi_slave_config

#include <fsl_flexio_spi.h> Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave

Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct _flexio_spi_transfer

#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

size_t dataSize

Transfer bytes.

uint8_t flags

FlexIO SPI control flag, MSB first or LSB first.

struct _flexio_spi_master_handle

#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData
Transfer buffer.

uint8_t *rxData
Receive buffer.

size_t transferSize
Total bytes to be transferred.

volatile size_t txRemainingBytes
Send data remaining in bytes.

volatile size_t rxRemainingBytes
Receive data remaining in bytes.

volatile uint32_t state
FlexIO SPI internal state.

uint8_t bytePerFrame
SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction
Shift direction.

flexio_spi_master_transfer_callback_t callback
FlexIO SPI callback.

void *userData
Callback parameter.

bool isCsContinuous
Is current transfer using CS continuous mode.

uint32_t timer1Cfg
TIMER1 TIMCFG regiser value backup.

2.32 FlexIO UART Driver

status_t FLEXIO_UART_Init(*FLEXIO_UART_Type* *base, const *flexio_uart_config_t* *userConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_GetDefaultConfig().

Example

```
FLEXIO_UART_Type base = {
.flexioBase = FLEXIO,
.TxPinIndex = 0,
.RxPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_uart_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
```

(continues on next page)

(continued from previous page)

```
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_UART_TxDataRegEmptyFlag
  - kFLEXIO_UART_RxEmptyFlag

– kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UARt transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART receive data register address.

static inline void FLEXIO_UART_EnableTxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_EnableRxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_Enable(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

static inline void FLEXIO_UART_WriteByte(*FLEXIO_UART_Type* *base, const uint8_t *buffer)

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

static inline void FLEXIO_UART_ReadByte(*FLEXIO_UART_Type* *base, uint8_t *buffer)

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

void FLEXIO_UART_TransferStartRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void FLEXIO_UART_TransferStopRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferSendNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See flexio_uart_transfer_t.

Return values

- kStatus_Success – Successfully starts the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished, data not written to the TX register.

void FLEXIO_UART_TransferAbortSend(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetSendCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferReceiveNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to xfer->data. This function returns with the parameter receivedBytes set to 5. For the last 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See flexio_uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

void FLEXIO_UART_TransferAbortReceive(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetReceiveCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy

Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy

Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle

UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR

ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun
UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout
UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport
Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char
FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar
7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar
8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar
9-bit data characters

enum _flexio_uart_interrupt_enable
Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
Define FlexIO UART status mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag
Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag
Receive buffer over run flag.

typedef enum *_flexio_uart_bit_count_per_char* flexio_uart_bit_count_per_char_t
FlexIO UART bit count per char.

typedef struct *_flexio_uart_type* FLEXIO_UART_Type
Define FlexIO UART access structure typedef.

typedef struct *_flexio_uart_config* flexio_uart_config_t
Define FlexIO UART user configuration structure.

typedef struct *_flexio_uart_transfer* flexio_uart_transfer_t
Define FlexIO UART transfer structure.

typedef struct *_flexio_uart_handle* flexio_uart_handle_t

typedef void (*flexio_uart_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *status_t* status, void *userData)

FlexIO UART transfer callback function.

UART_RETRY_TIMES

Retry times for waiting flag.

struct _flexio_uart_type

#include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

uint8_t TxPinIndex

Pin select for UART_Tx.

uint8_t RxPinIndex

Pin select for UART_Rx.

uint8_t shifterIndex[2]

Shifter index used in FlexIO UART.

uint8_t timerIndex[2]

Timer index used in FlexIO UART.

struct _flexio_uart_config

#include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

Public Members

bool enableUart

Enable/disable FlexIO UART TX & RX.

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

flexio_uart_bit_count_per_char_t bitCountPerChar

number of bits, 7/8/9 -bit

struct _flexio_uart_transfer

#include <fsl_flexio_uart.h> Define FlexIO UART transfer structure.

Public Members

size_t dataSize

Transfer size

struct __flexio_uart_handle

#include <fsl_flexio_uart.h> Define FLEXIO UART handle structure.

Public Members

const uint8_t *volatile txData
: Address of remaining data to send.

volatile size_t txDataSize
: Size of the remaining data to send.

uint8_t *volatile rxData
: Address of remaining data to receive.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t txDataSizeAll
: Total bytes to be sent.

size_t rxDataSizeAll
: Total bytes to be received.

uint8_t *rxRingBuffer
: Start address of the receiver ring buffer.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
: Callback function.

void *userData
: UART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state

union ___unnamed159___

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.

2.33 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 void FLEXSPI_GetDataLearningPhase(FLEXSPI_Type *base, uint8_t *portAPhase, uint8_t *portBPhase)

Gets the sampling clock phase selection after Data Learning.

Parameters

- base – FLEXSPI peripheral base address.
- portAPhase – Pointer to a uint8_t type variable to receive the selected clock phase on PORTA.
- portBPhase – Pointer to a uint8_t type variable to receive the selected clock phase on PORTB.

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

static inline void FLEXSPI_EnableIPParallelMode(FLEXSPI_Type *base, bool enable)

Enables/disables the FLEXSPI IP command parallel mode.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable parallel mode, false means disable parallel mode.

static inline void FLEXSPI_EnableAHBParallelMode(FLEXSPI_Type *base, bool enable)

Enables/disables the FLEXSPI AHB command parallel mode.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable parallel mode, false means disable parallel mode.

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_AhbBusTimeoutFlag
AHB Bus timeout.

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_DataLearningFailedFlag
Data learning failed.

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 enableCombination

Enable/disable combining PORT A and B Data Pins (SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.

bool enableDoze

Enable/disable doze mode support.

bool enableHalfSpeedAccess

Enable/disable divide by 2 of the clock for half speed commands.

bool enableSckBDiffOpt

Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.

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.

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.

2.34 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.

FLEXSPI EDMA driver.

FSL_FLEXSPI_EDMA_DRIVER_VERSION

FLEXSPI EDMA driver.

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

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

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

2.35 Flexspi_nor_flash_driver

Flash Pad Definitions.

Values:

enumerator kSerialFlash_1Pad

enumerator kSerialFlash_2Pads

enumerator kSerialFlash_4Pads

enumerator kSerialFlash_8Pads

FLEXSPI clock configuration type.

Values:

enumerator kFLEXSPIClk_SDR
Clock configure for SDR mode

enumerator kFLEXSPIClk_DDR
Clock configurat for DDR mode

enum _flexspi_read_sample_clk
FLEXSPI Read Sample Clock Source definition.

Values:

enumerator kFLEXSPIReadSampleClk_LoopbackInternally

enumerator kFLEXSPIReadSampleClk_LoopbackFromDqsPad

enumerator kFLEXSPIReadSampleClk_LoopbackFromSckPad

enumerator kFLEXSPIReadSampleClk_ExternalInputFromDqsPad

Flash Type Definition.

Values:

enumerator kFLEXSPIDeviceType_SerialNOR
Flash device is Serial NOR

Flash Configuration Command Type.

Values:

enumerator kDeviceConfigCmdType_Generic
Generic command, for example: configure dummy cycles, drive strength, etc

enumerator kDeviceConfigCmdType_QuadEnable
Quad Enable command

enumerator kDeviceConfigCmdType_Spi2Xpi
Switch from SPI to DPI/QPI/OPI mode

enumerator kDeviceConfigCmdType_Xpi2Spi
Switch from DPI/QPI/OPI to SPI mode

enumerator kDeviceConfigCmdType_Spi2NoCmd
Switch to 0-4-4/0-8-8 mode

enumerator kDeviceConfigCmdType_Reset
Reset device command

enum _flexspi_serial_clk_freq
Defintions for FLEXSPI Serial Clock Frequency.

Values:

enumerator kFLEXSPISerialClk_NoChange

enumerator kFLEXSPISerialClk_30MHz

enumerator kFLEXSPISerialClk_50MHz

enumerator kFLEXSPISerialClk_60MHz

enumerator kFLEXSPISerialClk_75MHz

enumerator kFLEXSPISerialClk_100MHz

Misc feature bit definitions.

Values:

enumerator kFLEXSPIMiscOffset_DiffClkEnable
Bit for Differential clock enable

enumerator kFLEXSPIMiscOffset_Ck2Enable
Bit for CK2 enable

enumerator kFLEXSPIMiscOffset_ParallelEnable
Bit for Parallel mode enable

enumerator kFLEXSPIMiscOffset_WordAddressableEnable
Bit for Word Addressable enable

enumerator kFLEXSPIMiscOffset_SafeConfigFreqEnable
Bit for Safe Configuration Frequency enable

enumerator kFLEXSPIMiscOffset_PadSettingOverrideEnable
Bit for Pad setting override enable

enumerator kFLEXSPIMiscOffset_DdrModeEnable
Bit for DDR clock confiuration indication.

enumerator kFLEXSPIMiscOffset_UseValidTimeForAllFreq
Bit for DLLCR settings under all modes

enum _flexspi_status_groups
FLEXSPI status group numbers.

Values:

enumerator kStatusROMGroup_FLEXSPI
Group number for ROM FLEXSPI status codes.

enumerator kStatusROMGroup_FLEXSPINOR
ROM FLEXSPI NOR status group number.

enum _flexspi_nor_status
FLEXSPI NOR status.

Values:

enumerator kStatus_FLEXSPINOR_ProgramFail
Status for Page programming failure

enumerator kStatus_FLEXSPINOR_EraseSectorFail
Status for Sector Erase failure

enumerator kStatus_FLEXSPINOR_EraseAllFail
Status for Chip Erase failure

enumerator kStatus_FLEXSPINOR_WaitTimeout
Status for timeout

enumerator kStatus_FlexSPINOR_NotSupported

enumerator kStatus_FlexSPINOR_WriteAlignmentError
Status for Alignement error

enumerator kStatus_FlexSPINOR_CommandFailure
Status for Erase/Program Verify Error

enumerator kStatus_FlexSPINOR_SFDP_NotFound
Status for SFDP read failure

enumerator kStatus_FLEXSPINOR_Unsupported_SFDP_Version
Status for Unrecognized SFDP version

enumerator kStatus_FLEXSPINOR_Flash_NotFound
Status for Flash detection failure

enumerator kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed
Status for DDR Read dummy probe failure

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout
Status for Sequence Execution timeout

enumerator kStatus_FLEXSPI_InvalidSequence
Status for Invalid Sequence

enumerator kStatus_FLEXSPI_DeviceTimeout
Status for Device timeout

Configure the device_type of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorCfgOption_Tag

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_SDR

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_DDR

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH1V8

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH3V0

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalSDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalSDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalDDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalSDR

Configure the quad_mode_setting of "serial_nor_config_option_t" structure.

Values:

enumerator kSerialNorQuadMode_NotConfig

enumerator kSerialNorQuadMode_StatusReg1_Bit6

enumerator kSerialNorQuadMode_StatusReg2_Bit1

enumerator kSerialNorQuadMode_StatusReg2_Bit7

enumerator kSerialNorQuadMode_StatusReg2_Bit1_0x31

FLEXSPI NOR Octal mode.

Values:

enumerator kSerialNorOctaldMode_NoOctalEnableBit

enumerator kSerialNorOctaldMode_HasOctalEnableBit

miscellaneous mode

Values:

enumerator kSerialNorEnhanceMode_Disabled

enumerator kSerialNorEnhanceMode_0_4_4_Mode

enumerator kSerialNorEnhanceMode_0_8_8_Mode

enumerator kSerialNorEnhanceMode_DataOrderSwapped

enumerator kSerialNorEnhanceMode_2ndPinMux

enumerator kSerialNorEnhanceMode_InternalLoopback

enumerator kSerialNorEnhanceMode_SpiMode

enumerator kSerialNorEnhanceMode_ExtDqs

FLEXSPI NOR reset logic options.

Values:

enumerator kFlashResetLogic_Disabled

enumerator kFlashResetLogic_ResetPin

enumerator kFlashResetLogic_JedecHwReset

Configure the flash_connection of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorConnection_SinglePortA

enumerator kSerialNorConnection_Parallel

enumerator kSerialNorConnection_SinglePortB

enumerator kSerialNorConnection_BothPorts

FLEXSPI ROOT clock soruce related definitions.

Values:

enumerator kFLEXSPIClkSrc_MainClk

enumerator kFLEXSPIClkSrc_Pll0

enumerator kFLEXSPIClkSrc_FroHf

enumerator kFLEXSPIClkSrc_Pll1

Restore sequence options Configure the restore_sequence of “flash_run_context_t” structure.

Values:

enumerator kRestoreSequence_None

enumerator kRestoreSequence_HW_Reset

enumerator kRestoreSequence_QPI_4_0xFFs

enumerator kRestoreSequence_QPI_Mode_0x00

enumerator kRestoreSequence_8QPI_FF

enumerator kRestoreSequence_Send_F0

enumerator kRestoreSequence_Send_66_99

enumerator kRestoreSequence_Send_6699_9966

enumerator kRestoreSequence_Send_06_FF
Adesto EcoXIP

enumerator kRestoreSequence_QPI_5_0xFFs

enumerator kRestoreSequence_Send_QPI_8_0xFFs

enumerator kRestoreSequence_Wakeup_0xAB

enumerator kRestoreSequence_Wakeup_0xAB_54

Port mode options.

Values:

enumerator kFlashInstMode_ExtendedSpi

enumerator kFlashInstMode_0_4_4_SDR

enumerator kFlashInstMode_0_4_4_DDR

enumerator kFlashInstMode_DPI_SDR

enumerator kFlashInstMode_DPI_DDR

enumerator kFlashInstMode_QPI_SDR

enumerator kFlashInstMode_QPI_DDR

enumerator kFlashInstMode_OPI_SDR

enumerator kFlashInstMode_OPI_DDR

Manufacturer ID.

Values:

enumerator kSerialFlash_ISSI_ManufacturerID
Manufacturer ID of the ISSI serial flash

enumerator kSerialFlash_Adesto_ManufacturerID
Manufacturer ID of the Adesto Technologies serial flash

enumerator kSerialFlash_Winbond_ManufacturerID
Manufacturer ID of the Winbond serial flash

enumerator kSerialFlash_Cypress_ManufacturerID
Manufacturer ID for Cypress

enum _flexspi_operation

Values:

enumerator kFLEXSPIOperation_Command
FLEXSPI operation: Only command, both TX and RX buffer are ignored.

enumerator kFLEXSPIOperation_Config
FLEXSPI operation: Configure device mode, the TX FIFO size is fixed in LUT.

enumerator kFLEXSPIOperation_Write
FLEXSPI operation: Write, only TX buffer is effective

enumerator kFLEXSPIOperation_Read
FLEXSPI operation: Read, only Rx Buffer is effective.

enumerator kFLEXSPIOperation_End

enum flexspi_clock_type_t

FLEXSPI Clock Type.

Values:

enumerator kFlexSpiClock_CoreClock
ARM Core Clock

enumerator kFlexSpiClock_AhbClock
AHB clock

enumerator kFlexSpiClock_SerialRootClock
Serial Root Clock

enumerator kFlexSpiClock_IpgClock
IPG clock

typedef struct *_serial_nor_config_option* serial_nor_config_option_t
Serial NOR configuration option.

typedef struct *_lut_sequence* flexspi_lut_seq_t
FLEXSPI LUT Sequence structure.

typedef struct *_FlexSPIConfig* flexspi_mem_config_t
FLEXSPI Memory Configuration Block.

typedef struct *_flexspi_nor_config* flexspi_nor_config_t
Serial NOR configuration block.

typedef enum *_flexspi_operation* flexspi_operation_t

typedef struct *_flexspi_xfer* flexspi_xfer_t
FLEXSPI Transfer Context.

uint32_t max_freq
Maximum supported Frequency

uint32_t misc_mode
miscellaneous mode

uint32_t quad_mode_setting
Quad mode setting

uint32_t cmd_pads
Command pads

uint32_t query_pads
SFDP read pads

uint32_t device_type
Device type

uint32_t option_size
Option size, in terms of uint32_t, size = (option_size + 1) * 4

uint32_t tag
Tag, must be 0x0E

struct *_serial_nor_config_option* B

uint32_t U

union *_serial_nor_config_option* option0

uint32_t dummy_cycles
Dummy cycles before read

uint32_t status_override
Override status register value during device mode configuration

uint32_t pinmux_group
The pinmux group selection

uint32_t dqs_pinmux_group
The DQS Pinmux Group Selection

uint32_t drive_strength
: The Drive Strength of FLEXSPI Pads

uint32_t flash_connection
: Flash connection option: 0 - Single Flash connected to port A, 1 - Parallel mode, 2 - Single Flash connected to Port B

struct *_serial_nor_config_option* B

uint32_t U

union *_serial_nor_config_option* option1

uint8_t por_mode

uint8_t current_mode

uint8_t exit_no_cmd_sequence

uint8_t restore_sequence

struct *flash_run_context_t* B

uint32_t U

uint8_t seqNum
: Sequence Number, valid number: 1-16

uint8_t seqId
: Sequence Index, valid number: 0-15

uint16_t reserved

uint8_t time_100ps
: Data valid time, in terms of 100ps

uint8_t delay_cells
: Data valid time, in terms of delay cells

uint32_t tag
: [0x000-0x003] Tag, fixed value 0x42464346UL

uint32_t version
: [0x004-0x007] Version,[31:24] -'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix

uint32_t reserved0
: [0x008-0x00b] Reserved for future use

uint8_t readSampleClkSrc
: [0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3

uint8_t csHoldTime
: [0x00d-0x00d] CS hold time, default value: 3

uint8_t csSetupTime
: [0x00e-0x00e] CS setup time, default value: 3

uint8_t columnAddressWidth
: [0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For Serial NAND, need to refer to datasheet

uint8_t deviceModeCfgEnable
: [0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

uint8_t deviceModeType

[0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch, Generic configuration, etc.

uint16_t waitTimeCfgCommands

[0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for DPI/QPI/OPI switch or reset command

flexspi_lut_seq_t deviceModeSeq

[0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUt sequence number, [31:16] Reserved

uint32_t deviceModeArg

[0x018-0x01b] Argument/Parameter for device configuration

uint8_t configCmdEnable

[0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable

uint8_t configModeType[3]

[0x01d-0x01f] Configure Mode Type, similar as deviceModeTpe

flexspi_lut_seq_t configCmdSeqs[3]

[0x020-0x02b] Sequence info for Device Configuration command, similar as deviceModeSeq

uint32_t reserved1

[0x02c-0x02f] Reserved for future use

uint32_t configCmdArgs[3]

[0x030-0x03b] Arguments/Parameters for device Configuration commands

uint32_t reserved2

[0x03c-0x03f] Reserved for future use

uint32_t controllerMiscOption

[0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more details

uint8_t deviceType

[0x044-0x044] Device Type: See Flash Type Definition for more details

uint8_t sflashPadType

[0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal

uint8_t serialClkFreq

[0x046-0x046] Serial Flash Frequencey, device specific definitions, See System Boot Chapter for more details

uint8_t lutCustomSeqEnable

[0x047-0x047] LUT customization Enable, it is required if the program/erase cannot be done using 1 LUT sequence, currently, only applicable to HyperFLASH

uint32_t reserved3[2]

[0x048-0x04f] Reserved for future use

uint32_t sflashA1Size

[0x050-0x053] Size of Flash connected to A1

uint32_t sflashA2Size

[0x054-0x057] Size of Flash connected to A2

uint32_t sflashB1Size

[0x058-0x05b] Size of Flash connected to B1

uint32_t sflashB2Size
: [0x05c-0x05f] Size of Flash connected to B2

uint32_t csPadSettingOverride
: [0x060-0x063] CS pad setting override value

uint32_t sclkPadSettingOverride
: [0x064-0x067] SCK pad setting override value

uint32_t dataPadSettingOverride
: [0x068-0x06b] data pad setting override value

uint32_t dqsPadSettingOverride
: [0x06c-0x06f] DQS pad setting override value

uint32_t timeoutInMs
: [0x070-0x073] Timeout threshold for read status command

uint32_t commandInterval
: [0x074-0x077] CS deselect interval between two commands

flexspi_dll_time_t dataValidTime[2]
: [0x078-0x07b] CLK edge to data valid time for PORT A and PORT B

uint16_t busyOffset
: [0x07c-0x07d] Busy offset, valid value: 0-31

uint16_t busyBitPolarity
: [0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 - busy flag is 0 when flash device is busy

uint32_t lookupTable[64]
: [0x080-0x17f] Lookup table holds Flash command sequences

flexspi_lut_seq_t lutCustomSeq[12]
: [0x180-0x1af] Customizable LUT Sequences

uint32_t dll0CrVal

uint32_t dll1CrVal

: [0x1b0-0x1b3] Customizable DLL0CR setting */

uint32_t reserved4[2]

: [0x1b4-0x1b7] Customizable DLL1CR setting */

: [0x1b8-0x1bf] Reserved for future use

flexspi_mem_config_t memConfig
: Common memory configuration info via FLEXSPI

uint32_t pageSize
: Page size of Serial NOR

uint32_t sectorSize
: Sector size of Serial NOR

uint8_t ipcmdSerialClkFreq
: Clock frequency for IP command

uint8_t isUniformBlockSize
: Sector/Block size is the same

uint8_t isDataOrderSwapped
Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)

uint8_t reserved0[1]
Reserved for future use

uint8_t serialNorType
Serial NOR Flash type: 0/1/2/3

uint8_t needExitNoCmdMode
Need to exit NoCmd mode before other IP command

uint8_t halfClkForNonReadCmd
Half the Serial Clock for non-read command: true/false

uint8_t needRestoreNoCmdMode
Need to Restore NoCmd mode after IP commmand execution

uint32_t blockSize
Block size

uint32_t flashStateCtx
Flash State Context

uint32_t reserve2[10]
Reserved for future use

flexspi_operation_t operation
FLEXSPI operation

uint32_t baseAddress
FLEXSPI operation base address

uint32_t seqId
Sequence Id

uint32_t seqNum
Sequence Number

bool isParallelModeEnable
Is a parallel transfer

uint32_t *txBuffer
Tx buffer

uint32_t txSize
Tx size in bytes

uint32_t *rxBuffer
Rx buffer

uint32_t rxSize
Rx size in bytes

uint32_t FLEXSPI_NorFlash_GetVersion(void)

status_t FLEXSPI_NorFlash_Init(uint32_t instance, *flexspi_nor_config_t* *config)
Initialize Serial NOR devices via FLEXSPI.

This function checks and initializes the FLEXSPI module for the other FLEXSPI APIs.

Parameters

- instance – storage the instance of FLEXSPI.

- config – A pointer to the storage for the driver runtime state.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_ProgramPage(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t dstAddr, const uint32_t *src)

Program data to Serial NOR via FLEXSPI.

This function programs the NOR flash memory with the dest address for a given flash area as determined by the dst address and the length.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- dst_addr – A pointer to the desired flash memory to be programmed. NOTE: It is recommended that use page aligned access; If the dst_addr is not aligned to page,the driver automatically aligns address down with the page address.
- src – A pointer to the source buffer of data that is to be programmed into the NOR flash.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_EraseAll(uint32_t instance, *flexspi_nor_config_t* *config)

Erase all the Serial NOR devices connected on FLEXSPI.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_EraseSector(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t address)

Erase one sector specified by address.

This function erases one of NOR flash sectors based on the desired address.

Parameters

- instance – storage the index of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- address – The start address of the desired NOR flash memory to be erased. NOTE: It is recommended that use sector-aligned access nor device; If dstAddr is not aligned with the sector,The driver automatically aligns address down with the sector address.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_EraseBlock(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t address)

Erase one block specified by address.

This function erases one block of NOR flash based on the desired address.

Parameters

- instance – storage the index of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired NOR flash memory to be erased. NOTE: It is recommended that use block-aligned access nor device; If dstAddr is not aligned with the block,The driver automatically aligns address down with the block address.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_GetConfig(uint32_t instance, *flexspi_nor_config_t* *config, *serial_nor_config_option_t* *option)

Get FLEXSPI NOR Configuration Block based on specified option.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- option – A pointer to the storage Serial NOR Configuration Option Context.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_Erase(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t start, uint32_t length)

Erase Flash Region specified by address and length.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- instance – storage the index of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired NOR flash memory to be erased. NOTE: It is recommended that use sector-aligned access nor device; If dstAddr is not aligned with the sector,the driver automatically aligns address down with the sector address.
- length – The length, given in bytes to be erased. NOTE: It is recommended that use sector-aligned access nor device; If length is not aligned with the sector,the driver automatically aligns up with the sector.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_Read(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t *dst, uint32_t start, uint32_t bytes)

Read data from Serial NOR via FLEXSPI.

This function read the NOR flash memory with the start address for a given flash area as determined by the dst address and the length.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- dst – A pointer to the dest buffer of data that is to be read from the NOR flash. NOTE: It is recommended that use page aligned access; If the dstAddr is not aligned to page,the driver automatically aligns address down with the page address.
- start – The start address of the desired NOR flash memory to be read.
- lengthInBytes – The length, given in bytes to be read.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_CommandXfer(uint32_t instance, *flexspi_xfer_t* *xfer)

FLEXSPI command.

This function is used to perform the command write sequence to the NOR device.

Parameters

- instance – storage the index of FLEXSPI.
- xfer – A pointer to the storage FLEXSPI Transfer Context.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.

status_t FLEXSPI_NorFlash_UpdateLut(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t seqNumber)

Configure FLEXSPI Lookup table.

Parameters

- instance – storage the index of FLEXSPI.
- seqIndex – storage the sequence Id.
- lutBase – A pointer to the look-up-table for command sequences.
- seqNumber – storage sequence number.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.

status_t FLEXSPI_NorFlash_SetClockSource(uint32_t clockSource)

Set the clock source for FLEXSPI NOR.

Parameters

- clockSource – Clock source for FLEXSPI NOR. See to "_flexspi_nor_clock_source".

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.

void FLEXSPI_NorFlash_ConfigClock(uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode)

Configure the FlexSPI clock.

The API is used for configuring the FlexSPI clock.

Parameters

- instance – storage the index of FLEXSPI.
- freqOption – storage FlexSPIFlexSPI flash serial clock frequency.
- sampleClkMode – storage the FlexSPI clock configuration type.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.

FLEXSPI_FEATURE_HAS_PARALLEL_MODE

FLEXSPI Feature related definitions

FSL_ROM_FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

CMD_SDR

CMD_DDR

RADDR_SDR

RADDR_DDR

CADDR_SDR

CADDR_DDR

MODE1_SDR

MODE1_DDR

MODE2_SDR

MODE2_DDR

MODE4_SDR

MODE4_DDR

MODE8_SDR

MODE8_DDR

WRITE_SDR

WRITE_DDR

READ_SDR

READ_DDR

LEARN_SDR

LEARN_DDR

DATSZ_SDR

DATSZ_DDR

DUMMY_SDR

DUMMY_DDR

DUMMY_RWDS_SDR

DUMMY_RWDS_DDR

JMP_ON_CS

FLEXSPI_STOP

FLEXSPI_1PAD

FLEXSPI_2PAD

FLEXSPI_4PAD

FLEXSPI_8PAD

NOR_CMD_LUT_SEQ_IDX_READ
: NOR LUT sequence index used for default LUT assignment NOTE: The will take effect if the lut sequences are not customized.

  READ LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS
: Read Status LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI
: Read status DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE
: Write Enable sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI
: Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_ERASESECTOR
: Erase Sector sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READID

NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK
: Erase Block sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM
: Program sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_CHIPERASE
: Chip Erase sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_READ_SFDP
: Read SFDP sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD
: Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD
: Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config blobk

struct _serial_nor_config_option
#include <fsl_flexspi_nor_flash.h> Serial NOR configuration option.

union flash_run_context_t
#include <fsl_flexspi_nor_flash.h>

struct _lut_sequence
#include <fsl_flexspi_nor_flash.h> FLEXSPI LUT Sequence structure.

struct flexspi_dll_time_t
#include <fsl_flexspi_nor_flash.h>

struct _FlexSPIConfig
#include <fsl_flexspi_nor_flash.h> FLEXSPI Memory Configuration Block.

struct _flexspi_nor_config
#include <fsl_flexspi_nor_flash.h> Serial NOR configuration block.

struct _flexspi_xfer
#include <fsl_flexspi_nor_flash.h> FLEXSPI Transfer Context.

union option0

Public Members

struct *_serial_nor_config_option* B

uint32_t U

struct B

Public Members

uint32_t max_freq
Maximum supported Frequency

uint32_t misc_mode
miscellaneous mode

uint32_t quad_mode_setting
Quad mode setting

uint32_t cmd_pads
Command pads

uint32_t query_pads
SFDP read pads

uint32_t device_type
Device type

uint32_t option_size
Option size, in terms of uint32_t, size = (option_size + 1) * 4

uint32_t tag
Tag, must be 0x0E

union option1

Public Members

struct *_serial_nor_config_option* B

uint32_t U

struct B

Public Members

uint32_t dummy_cycles
Dummy cycles before read

uint32_t status_override
Override status register value during device mode configuration

uint32_t pinmux_group
The pinmux group selection

uint32_t dqs_pinmux_group
The DQS Pinmux Group Selection

uint32_t drive_strength
The Drive Strength of FLEXSPI Pads

uint32_t flash_connection
Flash connection option: 0 - Single Flash connected to port A, 1 - Parallel mode, 2 - Single Flash connected to Port B

struct B

2.36 FREQME: Frequency Measurement

2.37 GDET

status_t GDET_Init(GDET_Type *base)
Initialize GDET.

This function initializes GDET block and setting.

Parameters

- base – GDET peripheral base address

Returns
Status of the init operation

void GDET_Deinit(GDET_Type *base)
Deinitialize GDET.

This function deinitializes GDET secure counter.

Parameters

- base – GDET peripheral base address

status_t GDET_Enable(GDET_Type *base)

Enable GDET.

This function enables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns

Status of the enable operation

status_t GDET_Disable(GDET_Type *base)

Disable GDET.

This function disables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns

Status of the disable operation

status_t GDET_IsolateOn(GDET_Type *base)

Turn on GDET isolation.

This function turns on isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns

Status of the operation

status_t GDET_IsolateOff(GDET_Type *base)

Turn off GDET isolation.

This function turns off isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns

Status of the operation

status_t GDET_ReconfigureVoltageMode(GDET_Type *base, *gdet_core_voltage_t* voltage)

Change expected core voltage.

This function changes core voltage which Glitch detector expect.

Parameters

- base – GDET peripheral base address
- voltage – Expected core voltage

Returns

Status of the GDET reconfiguration operation

FSL_GDET_DRIVER_VERSION

Defines GDET driver version 2.1.0.

Change log:

- Version 2.1.0
  - Update for multiple instances

- Fix bug in isolation off API
- Add enable and disable APIs

- 2.0.1
  - Fix MISRA in GDET_ReconfigureVoltageMode()
- Version 2.0.0
  - initial version

typedef uint32_t gdet_core_voltage_t

GDET Core Voltage.

These constants are used to define core voltage argument to be used with GDET_ReconfigureVoltageMode(). Different SoC may support various volatages, refer to documentation.

void GDET_DriverIRQHandler(void)

kGDET_0_9v

Voltage (0.9V)

kGDET_1_0v

Voltage (1.0V)

kGDET_1_1v

Voltage (1.1V)

2.38 Glitch Detect

2.39 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION

GPIO driver version.

enum _gpio_pin_direction

GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput

Set current pin as digital input

enumerator kGPIO_DigitalOutput

Set current pin as digital output

enum _gpio_checker_attribute

GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW

User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW

User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW

User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO_IgnoreAttributeCheck
Ignores the attribute check

enum _gpio_interrupt_config
Configures the interrupt generation condition.

Values:

enumerator kGPIO_InterruptStatusFlagDisabled
Interrupt status flag is disabled.

enumerator kGPIO_DMARisingEdge
ISF flag and DMA request on rising edge.

enumerator kGPIO_DMAFallingEdge
ISF flag and DMA request on falling edge.

enumerator kGPIO_DMAEitherEdge
ISF flag and DMA request on either edge.

enumerator kGPIO_FlagRisingEdge
Flag sets on rising edge.

enumerator kGPIO_FlagFallingEdge
Flag sets on falling edge.

enumerator kGPIO_FlagEitherEdge
Flag sets on either edge.

enumerator kGPIO_InterruptLogicZero
Interrupt when logic zero.

enumerator kGPIO_InterruptRisingEdge
Interrupt on rising edge.

enumerator kGPIO_InterruptFallingEdge
Interrupt on falling edge.

enumerator kGPIO_InterruptEitherEdge
Interrupt on either edge.

enumerator kGPIO_InterruptLogicOne
Interrupt when logic one.

enumerator kGPIO_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kGPIO_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

enum _gpio_interrupt_selection
Configures the selection of interrupt/DMA request/trigger output.

Values:

enumerator kGPIO_InterruptOutput0
Interrupt/DMA request/trigger output 0.

enumerator kGPIO_InterruptOutput1
Interrupt/DMA request/trigger output 1.

enum gpio_pin_interrupt_control_t
GPIO pin and interrupt control.

Values:

enumerator kGPIO_PinControlNonSecure
Pin Control Non-Secure.

enumerator kGPIO_InterruptControlNonSecure
Interrupt Control Non-Secure.

enumerator kGPIO_PinControlNonPrivilege
Pin Control Non-Privilege.

enumerator kGPIO_InterruptControlNonPrivilege
Interrupt Control Non-Privilege.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_checker_attribute* gpio_checker_attribute_t
GPIO checker attribute.

typedef struct *_gpio_pin_config* gpio_pin_config_t
The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

typedef enum *_gpio_interrupt_config* gpio_interrupt_config_t
Configures the interrupt generation condition.

typedef enum *_gpio_interrupt_selection* gpio_interrupt_selection_t
Configures the selection of interrupt/DMA request/trigger output.

typedef struct *_gpio_version_info* gpio_version_info_t
GPIO version information.

GPIO_FIT_REG(value)

struct _gpio_pin_config
#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Public Members

gpio_pin_direction_t pinDirection
GPIO direction, input or output

uint8_t outputLogic
Set a default output logic, which has no use in input

struct _gpio_version_info
#include <fsl_gpio.h> GPIO version information.

Public Members

uint16_t feature
Feature Specification Number.

uint8_t minor
Minor Version Number.

uint8_t major
Major Version Number.

2.40 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)
Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PortDenit(GPIO_Type *base)
Denitializes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)
Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalOutput,
  0,
}
```

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

void GPIO_GetVersionInfo(GPIO_Type *base, *gpio_version_info_t* *info)

Get GPIO version information.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- info – GPIO version information

static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, *gpio_pin_interrupt_control_t* mask)

lock or unlock secure privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Disable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Disable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)

Enable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputEnable(GPIO_Type *base, uint32_t mask)
Enable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputDisable(GPIO_Type *base, uint32_t mask)
Disable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)
Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)
Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_config_t* config)

Configures the gpio pin interrupt/DMA request.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- config – GPIO pin interrupt configuration.
  - kGPIO_InterruptStatusFlagDisabled: Interrupt/DMA request disabled.
  - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kGPIO_InterruptLogicZero : Interrupt when logic zero.
  - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
  - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
  - kGPIO_InterruptEitherEdge : Interrupt on either edge.
  - kGPIO_InterruptLogicOne : Interrupt when logic one.

- kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
- kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

static inline void GPIO_SetPinInterruptChannel(GPIO_Type *base, uint32_t pin, *gpio_interrupt_selection_t* selection)

Configures the gpio pin interrupt/DMA request/trigger output channel selection.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- selection – GPIO pin interrupt output selection.
  - kGPIO_InterruptOutput0: Interrupt/DMA request/trigger output 0.
  - kGPIO_InterruptOutput1 : Interrupt/DMA request/trigger output 1.

uint32_t GPIO_GpioGetInterruptFlags(GPIO_Type *base)

Read the GPIO interrupt status flags.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)

Returns

The current GPIO's interrupt status flag. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint32_t GPIO_GpioGetInterruptChannelFlags(GPIO_Type *base, uint32_t channel)

Read the GPIO interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

Returns

The current GPIO's interrupt status flag based on the selected interrupt channel. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint8_t GPIO_PinGetInterruptFlag(GPIO_Type *base, uint32_t pin)

Read individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on)
- pin – GPIO specific pin number.

Returns

The current selected pin's interrupt status flag.

void GPIO_GpioClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

void GPIO_GpioClearInterruptChannelFlags(GPIO_Type *base, uint32_t mask, uint32_t channel)

Clears GPIO pin interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

void GPIO_PinClearInterruptFlag(GPIO_Type *base, uint32_t pin)

Clear GPIO individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on).
- pin – GPIO specific pin number.

static inline void GPIO_SetMultipleInterruptPinsConfig(GPIO_Type *base, uint32_t mask, *gpio_interrupt_config_t* config)

Sets the GPIO interrupt configuration in PCR register for multiple pins.

Parameters

- base – GPIO peripheral base pointer.
- mask – GPIO pin number macro.
- config – GPIO pin interrupt configuration.
  - kGPIO_InterruptStatusFlagDisabled: Interrupt disabled.
  - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kGPIO_InterruptLogicZero : Interrupt when logic zero.
  - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
  - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
  - kGPIO_InterruptEitherEdge : Interrupt on either edge.
  - kGPIO_InterruptLogicOne : Interrupt when logic one.
  - kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
  - kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

void GPIO_CheckAttributeBytes(GPIO_Type *base, *gpio_checker_attribute_t* attribute)

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If

the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

2.41 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.

2.42 I3C Common Driver

typedef struct *_i3c_config* i3c_config_t
: Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

  This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

  The configuration structure can be made constant so it resides in flash.

uint32_t I3C_GetInstance(I3C_Type *base)
: Get which instance current I3C is used.

  Parameters

  - base – The I3C peripheral base address.

void I3C_GetDefaultConfig(*i3c_config_t* *config)
: Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

  This function provides the following default configuration for I3C:

```
config->enableMaster                = kI3C_MasterCapable;
config->disableTimeout              = false;
config->hKeep                       = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop         = true;
config->enableOpenDrainHigh         = true;
config->baudRate_Hz.i2cBaud         = 400000U;
config->baudRate_Hz.i3cPushPullBaud  = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress        = 0x0AU;
config->slowClock_Hz                = 1000000U;
config->enableSlave                 = true;
config->vendorID                    = 0x11BU;
config->enableRandomPart            = false;
config->partNumber                  = 0;
config->dcr                         = 0;
config->bcr = 0;
config->hdrMode             = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest       = false;
config->ignoreS0S1Error     = false;
config->offline             = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C_Init().

Parameters

- config – **[out]** User provided configuration structure for default values. Refer to i3c_config_t.

void I3C_Init(I3C_Type *base, const *i3c_config_t* *config, uint32_t sourceClock_Hz)

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- config – User provided peripheral configuration. Use I3C_GetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

struct _i3c_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

uint8_t masterDynamicAddress

Main master dynamic address configuration.

uint32_t maxWriteLength

Maximum write length.

uint32_t maxReadLength

Maximum read length.

bool enableSlave
: Whether to enable slave.

bool isHotJoin
: Whether to enable slave hotjoin before 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.

2.43 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.

void I3C_SlaveRequestEvent(I3C_Type *base, *i3c_slave_event_t* event)

I3C slave request event.

Parameters

- base – The I3C peripheral base address.
- event – I3C slave event of type i3c_slave_event_t

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

void I3C_MasterTransferCreateHandle(I3C_Type *base, *i3c_master_handle_t* *handle, const *i3c_master_transfer_callback_t* *callback, void *userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_MasterTransferNonBlocking(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I3C_MasterTransferGetCount(I3C_Type *base, *i3c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void I3C_MasterTransferAbort(I3C_Type *base, *i3c_master_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to the I3C master driver handle.

enum _i3c_master_flags

I3C master peripheral flags.

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag
- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterBetweenFlag

Between messages/DAAs flag

enumerator kI3C_MasterNackDetectFlag

NACK detected flag

enumerator kI3C_MasterSlaveStartFlag

Slave request start flag

enumerator kI3C_MasterControlDoneFlag

Master request complete flag

enumerator kI3C_MasterCompleteFlag

Transfer complete flag

enumerator kI3C_MasterRxReadyFlag

Rx data ready in Rx buffer flag

enumerator kI3C_MasterTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_MasterArbitrationWonFlag
Header address won arbitration flag

enumerator kI3C_MasterErrorFlag
Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag
Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum _i3c_master_error_flags
I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag
Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag
Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag
Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag
CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag
Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag
Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag
Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag
The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags
All error flags

enum _i3c_master_state
I3C working master state.

Values:

enumerator kI3C_MasterStateIdle
Bus stopped.

enumerator kI3C_MasterStateSlvReq
Bus stopped but slave holding SDA low.

enumerator kI3C_MasterStateMsgSdr
In SDR Message mode from using MWMSG_SDR.

enumerator kI3C_MasterStateNormAct
In normal active SDR mode.

enumerator kI3C_MasterStateDdr
In DDR Message mode.

enumerator kI3C_MasterStateDaa
In ENTDAA mode.

enumerator kI3C_MasterStateIbiAck
Waiting on IBI ACK/NACK decision.

enumerator kI3C_MasterStateIbiRcv
Receiving IBI.

enum _i3c_master_enable
I3C master enable configuration.

Values:

enumerator kI3C_MasterOff
Master off.

enumerator kI3C_MasterOn
Master on.

enumerator kI3C_MasterCapable
Master capable.

enum _i3c_master_hkeep
I3C high keeper configuration.

Values:

enumerator kI3C_MasterHighKeeperNone
Use PUR to hold SCL high.

enumerator kI3C_MasterHighKeeperWiredIn
Use pin_HK controls.

enumerator kI3C_MasterPassiveSDA
Hi-Z for Bus Free and hold SDA.

enumerator kI3C_MasterPassiveSDASCL
Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum _i3c_bus_request
Emits the requested operation when doing in pieces vs. by message.

Values:

enumerator kI3C_RequestNone
No request.

enumerator kI3C_RequestEmitStartAddr
Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop
Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack
Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA
Process DAA.

enumerator kI3C_RequestForceExit
Request to force exit.

enumerator kI3C_RequestAutoIbi
Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type
Bus type with EmitStartAddr.

Values:

enumerator kI3C_TypeI3CSdr
SDR mode of I3C.

enumerator kI3C_TypeI2C
Standard i2c protocol.

enumerator kI3C_TypeI3CDdr
HDR-DDR mode of I3C.

enum _i3c_ibi_response
IBI response.

Values:

enumerator kI3C_IbiRespAck
ACK with no mandatory byte.

enumerator kI3C_IbiRespNack
NACK.

enumerator kI3C_IbiRespAckMandatory
ACK with mandatory byte.

enumerator kI3C_IbiRespManual
Reserved.

enum _i3c_ibi_type
IBI type.

Values:

enumerator kI3C_IbiNormal
In-band interrupt.

enumerator kI3C_IbiHotJoin
slave hot join.

enumerator kI3C_IbiMasterRequest
slave master ship request.

enum _i3c_ibi_state
IBI state.

Values:

enumerator kI3C_IbiReady
In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed
In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending
In-band interrupt Ack/Nack pending for decision.

enum _i3c_direction
Direction of master and slave transfers.

Values:

enumerator kI3C_Write
Master transmit.

enumerator kI3C_Read
Master receive.

enum _i3c_tx_trigger_level
Watermark of TX int/dma trigger level.

Values:

enumerator kI3C_TxTriggerOnEmpty
Trigger on empty.

enumerator kI3C_TxTriggerUntilOneQuarterOrLess
Trigger on 1/4 full or less.

enumerator kI3C_TxTriggerUntilOneHalfOrLess
Trigger on 1/2 full or less.

enumerator kI3C_TxTriggerUntilOneLessThanFull
Trigger on 1 less than full or less.

enum _i3c_rx_trigger_level
Watermark of RX int/dma trigger level.

Values:

enumerator kI3C_RxTriggerOnNotEmpty
Trigger on not empty.

enumerator kI3C_RxTriggerUntilOneQuarterOrMore
Trigger on 1/4 full or more.

enumerator kI3C_RxTriggerUntilOneHalfOrMore
Trigger on 1/2 full or more.

enumerator kI3C_RxTriggerUntilThreeQuarterOrMore
Trigger on 3/4 full or more.

enum _i3c_rx_term_ops
I3C master read termination operations.

Values:

enumerator kI3C_RxTermDisable
Master doesn't terminate read, used for CCC transfer.

enumerator kI3C_RxAutoTerm
Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C_RxTermLastByte
Master terminates read at any time after START, no length limitation.

enum _i3c_start_scl_delay

I3C start SCL delay options.

Values:

enumerator kI3C_NoDelay

No delay.

enumerator kI3C_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum _i3c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _i3c_master_transfer::flags field.

Values:

enumerator kI3C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI3C_TransferNoStopFlag

Don't send a stop condition.

enumerator kI3C_TransferWordsFlag

Transfer in words, else transfer in bytes.

enumerator kI3C_TransferDisableRxTermFlag

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C_TransferRxAutoTermFlag

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C_MasterReceive.

enumerator kI3C_TransferStartWithBroadcastAddr

Start transfer with 0x7E, then read/write data with device address.

typedef enum *_i3c_master_state* i3c_master_state_t

I3C working master state.

typedef enum *_i3c_master_enable* i3c_master_enable_t

I3C master enable configuration.

typedef enum *_i3c_master_hkeep* i3c_master_hkeep_t

I3C high keeper configuration.

typedef enum *_i3c_bus_request* i3c_bus_request_t

Emits the requested operation when doing in pieces vs. by message.

typedef enum *_i3c_bus_type* i3c_bus_type_t
Bus type with EmitStartAddr.

typedef enum *_i3c_ibi_response* i3c_ibi_response_t
IBI response.

typedef enum *_i3c_ibi_type* i3c_ibi_type_t
IBI type.

typedef enum *_i3c_ibi_state* i3c_ibi_state_t
IBI state.

typedef enum *_i3c_direction* i3c_direction_t
Direction of master and slave transfers.

typedef enum *_i3c_tx_trigger_level* i3c_tx_trigger_level_t
Watermark of TX int/dma trigger level.

typedef enum *_i3c_rx_trigger_level* i3c_rx_trigger_level_t
Watermark of RX int/dma trigger level.

typedef enum *_i3c_rx_term_ops* i3c_rx_term_ops_t
I3C master read termination operations.

typedef enum *_i3c_start_scl_delay* i3c_start_scl_delay_t
I3C start SCL delay options.

typedef struct *_i3c_register_ibi_addr* i3c_register_ibi_addr_t
Structure with setting master IBI rules and slave registry.

typedef struct *_i3c_baudrate* i3c_baudrate_hz_t
Structure with I3C baudrate settings.

typedef struct *_i3c_master_daa_baudrate* i3c_master_daa_baudrate_t
I3C DAA baud rate configuration.

typedef struct *_i3c_master_config* i3c_master_config_t
Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef struct *_i3c_master_transfer* i3c_master_transfer_t

typedef struct *_i3c_master_handle* i3c_master_handle_t

typedef struct *_i3c_master_transfer_callback* i3c_master_transfer_callback_t
i3c master callback functions.

typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
Typedef for master interrupt handler.

struct __i3c_register_ibi_addr
#include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.

Public Members

uint8_t address[5]
Address array for registry.

bool i3cFastStart

Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct _i3c_baudrate

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

uint32_t i2cBaud

Desired I2C baud rate in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct _i3c_master_daa_baudrate

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

uint32_t sourceClock_Hz

FCLK, function clock in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct _i3c_master_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

struct __i3c_master_transfer_callback

#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)

Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)

IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Transfer complete callback

struct __i3c_master_transfer

#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration _i3c_master_transfer_flags for available options. Set to 0 or kI3C_TransferDefaultFlag for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i3c_direction_t direction

Either kI3C_Read or kI3C_Write.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

i3c_bus_type_t busType

bus type.

i3c_ibi_response_t ibiResponse

ibi response during transfer.

struct _i3c_master_handle

#include <fsl_i3c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint32_t remainingBytes

Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps

Read termination operation.

i3c_master_transfer_t transfer

Copy of the current transfer info.

uint8_t ibiAddress

Slave address which request IBI.

uint8_t *ibiBuff

Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize

IBI payload size.

i3c_ibi_type_t ibiType

IBI type.

i3c_master_transfer_callback_t callback

Callback functions pointer.

void *userData

Application data passed to callback.

2.44 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.

2.45 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.

void I3C_SlaveRequestEvent(I3C_Type *base, *i3c_slave_event_t* event)

I3C slave request event.

Parameters

- base – The I3C peripheral base address.
- event – I3C slave event of type i3c_slave_event_t

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.

void I3C_SlaveRequestIBIWithData(I3C_Type *base, uint8_t *data, size_t dataSize)

I3C slave request IBI event with data payload(mandatory and extended).

Parameters

- base – The I3C peripheral base address.
- data – Pointer to IBI data to be sent in the request.
- dataSize – IBI data size.

void I3C_SlaveRequestIBIWithSingleData(I3C_Type *base, uint8_t data, size_t dataSize)

I3C slave request IBI event with single data.

Deprecated:

Do not use this function. It has been superseded by I3C_SlaveRequestIBIWithData.

Parameters

- base – The I3C peripheral base address.
- data – IBI data to be sent in the request.
- dataSize – IBI data size.

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.

bool isHotJoin
: Whether to enable slave hotjoin before 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

2.46 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.

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.

2.47 INPUTMUX: Input Multiplexing Driver

enum _inputmux_connection_t
: INPUTMUX connections type.

Values:

enumerator kINPUTMUX_Sct0In0ToSct0
: SCT0 INMUX.

enumerator kINPUTMUX_Sct0In1ToSct0

enumerator kINPUTMUX_Sct0In2ToSct0

enumerator kINPUTMUX_Sct0In3ToSct0

enumerator kINPUTMUX_Sct0In4ToSct0

enumerator kINPUTMUX_Sct0In5ToSct0

enumerator kINPUTMUX_Sct0In6ToSct0

enumerator kINPUTMUX_Sct0In7ToSct0

enumerator kINPUTMUX_Ctimer0M0ToSct0

enumerator kINPUTMUX_Ctimer1M0ToSct0

enumerator kINPUTMUX_Ctimer2M0ToSct0

enumerator kINPUTMUX_Ctimer3M0ToSct0

enumerator kINPUTMUX_Ctimer4M0ToSct0

enumerator kINPUTMUX_Adc0IrqToSct0

enumerator kINPUTMUX_GpioIntBmatToSct0

enumerator kINPUTMUX_Usb0StartOfFrameToggleToSct0

enumerator kINPUTMUX_Usb1StartOfFrameToggleToSct0

enumerator kINPUTMUX_SincFilterCh0ToSct0

enumerator kINPUTMUX_SincFilterCh1ToSct0

enumerator kINPUTMUX_SincFilterCh2ToSct0

enumerator kINPUTMUX_SincFilterCh3ToSct0

enumerator kINPUTMUX_SincFilterCh4ToSct0

enumerator kINPUTMUX_ArmTxevToSct0

enumerator kINPUTMUX_DebugHaltedToSct0

enumerator kINPUTMUX_Adc1IrqToSct0

enumerator kINPUTMUX_Adc0tcomp0ToSct0

enumerator kINPUTMUX_Adc0tcomp1ToSct0

enumerator kINPUTMUX_Adc0tcomp2ToSct0

enumerator kINPUTMUX_Adc0tcomp3ToSct0

enumerator kINPUTMUX_Adc1tcomp0ToSct0

enumerator kINPUTMUX_Adc1tcomp1ToSct0

enumerator kINPUTMUX_Adc1tcomp2ToSct0

enumerator kINPUTMUX_Adc1tcomp3ToSct0

enumerator kINPUTMUX_Cmp0OutToSct0

enumerator kINPUTMUX_Cmp1OutToSct0

enumerator kINPUTMUX_Cmp2OutToSct0

enumerator kINPUTMUX_Pwm0A0Trig01ToSct0

enumerator kINPUTMUX_Pwm0A1Trig01ToSct0

enumerator kINPUTMUX_Pwm0A2Trig01ToSct0

enumerator kINPUTMUX_Pwm0A3Trig01ToSct0

enumerator kINPUTMUX_Pwm1A0Trig01ToSct0

enumerator kINPUTMUX_Pwm1A1Trig01ToSct0

enumerator kINPUTMUX_Pwm1A2Trig01ToSct0

enumerator kINPUTMUX_Pwm1A3Trig01ToSct0

enumerator kINPUTMUX_Qdc0CmpPosMatchToSct0

enumerator kINPUTMUX_Qdc1CmpPosMatchToSct0

enumerator kINPUTMUX_EvtgOut0AToSct0

enumerator kINPUTMUX_EvtgOut0BToSct0

enumerator kINPUTMUX_EvtgOut1AToSct0

enumerator kINPUTMUX_EvtgOut1BToSct0

enumerator kINPUTMUX_EvtgOut2AToSct0

enumerator kINPUTMUX_EvtgOut2BToSct0

enumerator kINPUTMUX_EvtgOut3AToSct0

enumerator kINPUTMUX_EvtgOut3BToSct0

enumerator kINPUTMUX_Fc3P0ToSct0

enumerator kINPUTMUX_Fc3P1ToSct0

enumerator kINPUTMUX_Fc3P2ToSct0

enumerator kINPUTMUX_Fc3P3ToSct0

enumerator kINPUTMUX_Lpflexcomm0Trig0ToSct0

enumerator kINPUTMUX_Lpflexcomm0Trig1ToSct0

enumerator kINPUTMUX_Lpflexcomm0Trig2ToSct0

enumerator kINPUTMUX_Lpflexcomm1Trig0ToSct0

enumerator kINPUTMUX_Lpflexcomm1Trig1ToSct0

enumerator kINPUTMUX_Lpflexcomm1Trig2ToSct0

enumerator kINPUTMUX_Lpflexcomm2Trig0ToSct0

enumerator kINPUTMUX_Lpflexcomm2Trig1ToSct0

enumerator kINPUTMUX_Lpflexcomm2Trig2ToSct0

enumerator kINPUTMUX_Lpflexcomm3Trig0ToSct0

enumerator kINPUTMUX_Lpflexcomm3Trig1ToSct0

enumerator kINPUTMUX_Lpflexcomm3Trig2ToSct0

enumerator kINPUTMUX_Lpflexcomm3Trig3ToSct0

enumerator kINPUTMUX_Sai0TxBclkToSct0

enumerator kINPUTMUX_Sai0RxBclkToSct0

enumerator kINPUTMUX_Sai1TxBclkToSct0

enumerator kINPUTMUX_Sai1RxBclkToSct0
TIMER0 CAPTSEL.

enumerator kINPUTMUX_CtimerInp0ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp1ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp2ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp3ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp4ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp5ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp6ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp7ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp8ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp9ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp10ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp11ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp12ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp13ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp14ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp15ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp16ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp17ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp18ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp19ToTimer0Captsel

enumerator kINPUTMUX_Usb0StartOfFrameToTimer0Captsel

enumerator kINPUTMUX_Usb1StartOfFrameToTimer0Captsel

enumerator kINPUTMUX_DcdcBurstActiveToTimer0Captsel

enumerator kINPUTMUX_Sai0TxSyncOutToTimer0Captsel

enumerator kINPUTMUX_Sai0RxSyncOutToTimer0Captsel

enumerator kINPUTMUX_Adc0IrqToTimer0Captsel

enumerator kINPUTMUX_Adc1IrqToTimer0Captsel

enumerator kINPUTMUX_Cmp0OutToTimer0Captsel

enumerator kINPUTMUX_Cmp1OutToTimer0Captsel

enumerator kINPUTMUX_Cmp2OutToTimer0Captsel

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer0Captsel

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer0Captsel

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer0Captsel

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer0Captsel

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer0Captsel

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer0Captsel

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer0Captsel

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer0Captsel

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer0Captsel

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer0Captsel

enumerator kINPUTMUX_EvtgOut0AToTimer0Captsel

enumerator kINPUTMUX_EvtgOut0BToTimer0Captsel

enumerator kINPUTMUX_EvtgOut1AToTimer0Captsel

enumerator kINPUTMUX_EvtgOut1BToTimer0Captsel

enumerator kINPUTMUX_EvtgOut2AToTimer0Captsel

enumerator kINPUTMUX_EvtgOut2BToTimer0Captsel

enumerator kINPUTMUX_EvtgOut3AToTimer0Captsel

enumerator kINPUTMUX_EvtgOut3BToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer0Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer0Captsel

enumerator kINPUTMUX_Sai1TxSyncOutToTimer0Captsel

enumerator kINPUTMUX_Sai1RxSyncOutToTimer0Captsel

TIMER1 CAPTSEL.

enumerator kINPUTMUX_CtimerInp0ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp1ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp2ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp3ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp4ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp5ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp6ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp7ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp8ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp9ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp10ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp11ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp12ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp13ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp14ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp15ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp16ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp17ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp18ToTimer1Captsel

enumerator kINPUTMUX_CtimerInp19ToTimer1Captsel

enumerator kINPUTMUX_Usb0StartOfFrameToTimer1Captsel

enumerator kINPUTMUX_Usb1StartOfFrameToTimer1Captsel

enumerator kINPUTMUX_DcdcBurstActiveToTimer1Captsel

enumerator kINPUTMUX_Sai0TxSyncOutToTimer1Captsel

enumerator kINPUTMUX_Sai0RxSyncOutToTimer1Captsel

enumerator kINPUTMUX_Adc0IrqToTimer1Captsel

enumerator kINPUTMUX_Adc1IrqToTimer1Captsel

enumerator kINPUTMUX_Cmp0OutToTimer1Captsel

enumerator kINPUTMUX_Cmp1OutToTimer1Captsel

enumerator kINPUTMUX_Cmp2OutToTimer1Captsel

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer1Captsel

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer1Captsel

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer1Captsel

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer1Captsel

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer1Captsel

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer1Captsel

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer1Captsel

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer1Captsel

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer1Captsel

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer1Captsel

enumerator kINPUTMUX_EvtgOut0AToTimer1Captsel

enumerator kINPUTMUX_EvtgOut0BToTimer1Captsel

enumerator kINPUTMUX_EvtgOut1AToTimer1Captsel

enumerator kINPUTMUX_EvtgOut1BToTimer1Captsel

enumerator kINPUTMUX_EvtgOut2AToTimer1Captsel

enumerator kINPUTMUX_EvtgOut2BToTimer1Captsel

enumerator kINPUTMUX_EvtgOut3AToTimer1Captsel

enumerator kINPUTMUX_EvtgOut3BToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer1Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer1Captsel

enumerator kINPUTMUX_Sai1TxSyncOutToTimer1Captsel

enumerator kINPUTMUX_Sai1RxSyncOutToTimer1Captsel

TIMER2 CAPTSEL.

enumerator kINPUTMUX_CtimerInp0ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp1ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp2ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp3ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp4ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp5ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp6ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp7ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp8ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp9ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp10ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp11ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp12ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp13ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp14ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp15ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp16ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp17ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp18ToTimer2Captsel

enumerator kINPUTMUX_CtimerInp19ToTimer2Captsel

enumerator kINPUTMUX_Usb0StartOfFrameToTimer2Captsel

enumerator kINPUTMUX_Usb1StartOfFrameToTimer2Captsel

enumerator kINPUTMUX_DcdcBurstActiveToTimer2Captsel

enumerator kINPUTMUX_Sai0TxSyncOutToTimer2Captsel

enumerator kINPUTMUX_Sai0RxSyncOutToTimer2Captsel

enumerator kINPUTMUX_Adc0IrqToTimer2Captsel

enumerator kINPUTMUX_Adc1IrqToTimer2Captsel

enumerator kINPUTMUX_Cmp0OutToTimer2Captsel

enumerator kINPUTMUX_Cmp1OutToTimer2Captsel

enumerator kINPUTMUX_Cmp2OutToTimer2Captsel

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer2Captsel

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer2Captsel

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer2Captsel

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer2Captsel

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer2Captsel

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer2Captsel

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer2Captsel

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer2Captsel

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer2Captsel

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer2Captsel

enumerator kINPUTMUX_EvtgOut0AToTimer2Captsel

enumerator kINPUTMUX_EvtgOut0BToTimer2Captsel

enumerator kINPUTMUX_EvtgOut1AToTimer2Captsel

enumerator kINPUTMUX_EvtgOut1BToTimer2Captsel

enumerator kINPUTMUX_EvtgOut2AToTimer2Captsel

enumerator kINPUTMUX_EvtgOut2BToTimer2Captsel

enumerator kINPUTMUX_EvtgOut3AToTimer2Captsel

enumerator kINPUTMUX_EvtgOut3BToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer2Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer2Captsel

enumerator kINPUTMUX_Sai1TxSyncOutToTimer2Captsel

enumerator kINPUTMUX_Sai1RxSyncOutToTimer2Captsel
TIMER3 CAPTSEL.

enumerator kINPUTMUX_CtimerInp0ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp1ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp2ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp3ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp4ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp5ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp6ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp7ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp8ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp9ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp10ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp11ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp12ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp13ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp14ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp15ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp16ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp17ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp18ToTimer3Captsel

enumerator kINPUTMUX_CtimerInp19ToTimer3Captsel

enumerator kINPUTMUX_Usb0StartOfFrameToTimer3Captsel

enumerator kINPUTMUX_Usb1StartOfFrameToTimer3Captsel

enumerator kINPUTMUX_DcdcBurstActiveToTimer3Captsel

enumerator kINPUTMUX_Sai0TxSyncOutToTimer3Captsel

enumerator kINPUTMUX_Sai0RxSyncOutToTimer3Captsel

enumerator kINPUTMUX_Adc0IrqToTimer3Captsel

enumerator kINPUTMUX_Adc1IrqToTimer3Captsel

enumerator kINPUTMUX_Cmp0OutToTimer3Captsel

enumerator kINPUTMUX_Cmp1OutToTimer3Captsel

enumerator kINPUTMUX_Cmp2OutToTimer3Captsel

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer3Captsel

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer3Captsel

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer3Captsel

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer3Captsel

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer3Captsel

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer3Captsel

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer3Captsel

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer3Captsel

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer3Captsel

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer3Captsel

enumerator kINPUTMUX_EvtgOut0AToTimer3Captsel

enumerator kINPUTMUX_EvtgOut0BToTimer3Captsel

enumerator kINPUTMUX_EvtgOut1AToTimer3Captsel

enumerator kINPUTMUX_EvtgOut1BToTimer3Captsel

enumerator kINPUTMUX_EvtgOut2AToTimer3Captsel

enumerator kINPUTMUX_EvtgOut2BToTimer3Captsel

enumerator kINPUTMUX_EvtgOut3AToTimer3Captsel

enumerator kINPUTMUX_EvtgOut3BToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer3Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer3Captsel

enumerator kINPUTMUX_Sai1TxSyncOutToTimer3Captsel

enumerator kINPUTMUX_Sai1RxSyncOutToTimer3Captsel

Timer4 CAPTSEL.

enumerator kINPUTMUX_CtimerInp0ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp1ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp2ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp3ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp4ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp5ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp6ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp7ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp8ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp9ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp10ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp11ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp12ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp13ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp14ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp15ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp16ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp17ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp18ToTimer4Captsel

enumerator kINPUTMUX_CtimerInp19ToTimer4Captsel

enumerator kINPUTMUX_Usb0StartOfFrameToTimer4Captsel

enumerator kINPUTMUX_Usb1StartOfFrameToTimer4Captsel

enumerator kINPUTMUX_DcdcBurstActiveToTimer4Captsel

enumerator kINPUTMUX_Sai0TxSyncOutToTimer4Captsel

enumerator kINPUTMUX_Sai0RxSyncOutToTimer4Captsel

enumerator kINPUTMUX_Adc0IrqToTimer4Captsel

enumerator kINPUTMUX_Adc1IrqToTimer4Captsel

enumerator kINPUTMUX_Cmp0OutToTimer4Captsel

enumerator kINPUTMUX_Cmp1OutToTimer4Captsel

enumerator kINPUTMUX_Cmp2OutToTimer4Captsel

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer4Captsel

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer4Captsel

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer4Captsel

enumerator kINPUTMUX_EvtgOut0AToTimer4Captsel

enumerator kINPUTMUX_EvtgOut0BToTimer4Captsel

enumerator kINPUTMUX_EvtgOut1AToTimer4Captsel

enumerator kINPUTMUX_EvtgOut1BToTimer4Captsel

enumerator kINPUTMUX_EvtgOut2AToTimer4Captsel

enumerator kINPUTMUX_EvtgOut2BToTimer4Captsel

enumerator kINPUTMUX_EvtgOut3AToTimer4Captsel

enumerator kINPUTMUX_EvtgOut3BToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer4Captsel

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer4Captsel

enumerator kINPUTMUX_Sai1TxSyncOutToTimer4Captsel

enumerator kINPUTMUX_Sai1RxSyncOutToTimer4Captsel

TIMER0 Trigger.

enumerator kINPUTMUX_CtimerInp0ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp1ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp2ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp3ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp4ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp5ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp6ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp7ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp8ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp9ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp10ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp11ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp12ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp13ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp14ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp15ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp16ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp17ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp18ToTimer0Trigger

enumerator kINPUTMUX_CtimerInp19ToTimer0Trigger

enumerator kINPUTMUX_Usb0StartOfFrameToTimer0Trigger

enumerator kINPUTMUX_Usb1StartOfFrameToTimer0Trigger

enumerator kINPUTMUX_DcdcBurstActiveToTimer0Trigger

enumerator kINPUTMUX_Sai0TxSyncOutToTimer0Trigger

enumerator kINPUTMUX_Sai0RxSyncOutToTimer0Trigger

enumerator kINPUTMUX_Adc0IrqToTimer0Trigger

enumerator kINPUTMUX_Adc1IrqToTimer0Trigger

enumerator kINPUTMUX_Cmp0OutToTimer0Trigger

enumerator kINPUTMUX_Cmp1OutToTimer0Trigger

enumerator kINPUTMUX_Cmp2OutToTimer0Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer0Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer0Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer0Trigger

enumerator kINPUTMUX_EvtgOut0AToTimer0Trigger

enumerator kINPUTMUX_EvtgOut0BToTimer0Trigger

enumerator kINPUTMUX_EvtgOut1AToTimer0Trigger

enumerator kINPUTMUX_EvtgOut1BToTimer0Trigger

enumerator kINPUTMUX_EvtgOut2AToTimer0Trigger

enumerator kINPUTMUX_EvtgOut2BToTimer0Trigger

enumerator kINPUTMUX_EvtgOut3AToTimer0Trigger

enumerator kINPUTMUX_EvtgOut3BToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer0Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer0Trigger

enumerator kINPUTMUX_Sai1TxSyncOutToTimer0Trigger

enumerator kINPUTMUX_Sai1RxSyncOutToTimer0Trigger

TIMER1 Trigger.

enumerator kINPUTMUX_CtimerInp0ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp1ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp2ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp3ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp4ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp5ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp6ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp7ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp8ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp9ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp10ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp11ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp12ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp13ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp14ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp15ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp16ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp17ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp18ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp19ToTimer1Trigger

enumerator kINPUTMUX_Usb0StartOfFrameToTimer1Trigger

enumerator kINPUTMUX_Usb1StartOfFrameToTimer1Trigger

enumerator kINPUTMUX_DcdcBurstActiveToTimer1Trigger

enumerator kINPUTMUX_Sai0TxSyncOutToTimer1Trigger

enumerator kINPUTMUX_Sai0RxSyncOutToTimer1Trigger

enumerator kINPUTMUX_Adc0IrqToTimer1Trigger

enumerator kINPUTMUX_Adc1IrqToTimer1Trigger

enumerator kINPUTMUX_Cmp0OutToTimer1Trigger

enumerator kINPUTMUX_Cmp1OutToTimer1Trigger

enumerator kINPUTMUX_Cmp2OutToTimer1Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer1Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer1Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer1Trigger

enumerator kINPUTMUX_EvtgOut0AToTimer1Trigger

enumerator kINPUTMUX_EvtgOut0BToTimer1Trigger

enumerator kINPUTMUX_EvtgOut1AToTimer1Trigger

enumerator kINPUTMUX_EvtgOut1BToTimer1Trigger

enumerator kINPUTMUX_EvtgOut2AToTimer1Trigger

enumerator kINPUTMUX_EvtgOut2BToTimer1Trigger

enumerator kINPUTMUX_EvtgOut3AToTimer1Trigger

enumerator kINPUTMUX_EvtgOut3BToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer1Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer1Trigger

enumerator kINPUTMUX_Sai1TxSyncOutToTimer1Trigger

enumerator kINPUTMUX_Sai1RxSyncOutToTimer1Trigger

TIMER2 Trigger.

enumerator kINPUTMUX_CtimerInp0ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp1ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp2ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp3ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp4ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp5ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp6ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp7ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp8ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp9ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp10ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp11ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp12ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp13ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp14ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp15ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp16ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp17ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp18ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp19ToTimer2Trigger

enumerator kINPUTMUX_Usb0StartOfFrameToTimer2Trigger

enumerator kINPUTMUX_Usb1StartOfFrameToTimer2Trigger

enumerator kINPUTMUX_DcdcBurstActiveToTimer2Trigger

enumerator kINPUTMUX_Sai0TxSyncOutToTimer2Trigger

enumerator kINPUTMUX_Sai0RxSyncOutToTimer2Trigger

enumerator kINPUTMUX_Adc0IrqToTimer2Trigger

enumerator kINPUTMUX_Adc1IrqToTimer2Trigger

enumerator kINPUTMUX_Cmp0OutToTimer2Trigger

enumerator kINPUTMUX_Cmp1OutToTimer2Trigger

enumerator kINPUTMUX_Cmp2OutToTimer2Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer2Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer2Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer2Trigger

enumerator kINPUTMUX_EvtgOut0AToTimer2Trigger

enumerator kINPUTMUX_EvtgOut0BToTimer2Trigger

enumerator kINPUTMUX_EvtgOut1AToTimer2Trigger

enumerator kINPUTMUX_EvtgOut1BToTimer2Trigger

enumerator kINPUTMUX_EvtgOut2AToTimer2Trigger

enumerator kINPUTMUX_EvtgOut2BToTimer2Trigger

enumerator kINPUTMUX_EvtgOut3AToTimer2Trigger

enumerator kINPUTMUX_EvtgOut3BToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer2Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer2Trigger

enumerator kINPUTMUX_Sai1TxSyncOutToTimer2Trigger

enumerator kINPUTMUX_Sai1RxSyncOutToTimer2Trigger

TIMER3 Trigger.

enumerator kINPUTMUX_CtimerInp0ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp1ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp2ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp3ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp4ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp5ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp6ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp7ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp8ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp9ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp10ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp11ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp12ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp13ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp14ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp15ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp16ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp17ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp18ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp19ToTimer3Trigger

enumerator kINPUTMUX_Usb0StartOfFrameToTimer3Trigger

enumerator kINPUTMUX_Usb1StartOfFrameToTimer3Trigger

enumerator kINPUTMUX_DcdcBurstActiveToTimer3Trigger

enumerator kINPUTMUX_Sai0TxSyncOutToTimer3Trigger

enumerator kINPUTMUX_Sai0RxSyncOutToTimer3Trigger

enumerator kINPUTMUX_Adc0IrqToTimer3Trigger

enumerator kINPUTMUX_Adc1IrqToTimer3Trigger

enumerator kINPUTMUX_Cmp0OutToTimer3Trigger

enumerator kINPUTMUX_Cmp1OutToTimer3Trigger

enumerator kINPUTMUX_Cmp2OutToTimer3Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer3Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer3Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer3Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer3Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer3Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer3Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer3Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer3Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer3Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer3Trigger

enumerator kINPUTMUX_EvtgOut0AToTimer3Trigger

enumerator kINPUTMUX_EvtgOut0BToTimer3Trigger

enumerator kINPUTMUX_EvtgOut1AToTimer3Trigger

enumerator kINPUTMUX_EvtgOut1BToTimer3Trigger

enumerator kINPUTMUX_EvtgOut2AToTimer3Trigger

enumerator kINPUTMUX_EvtgOut2BToTimer3Trigger

enumerator kINPUTMUX_EvtgOut3AToTimer3Trigger

enumerator kINPUTMUX_EvtgOut3BToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer3Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer3Trigger

enumerator kINPUTMUX_Sai1TxSyncOutToTimer3Trigger

enumerator kINPUTMUX_Sai1RxSyncOutToTimer3Trigger

TIMER4 Trigger.

enumerator kINPUTMUX_CtimerInp0ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp1ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp2ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp3ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp4ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp5ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp6ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp7ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp8ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp9ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp10ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp11ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp12ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp13ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp14ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp15ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp16ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp17ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp18ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp19ToTimer4Trigger

enumerator kINPUTMUX_Usb0StartOfFrameToTimer4Trigger

enumerator kINPUTMUX_Usb1StartOfFrameToTimer4Trigger

enumerator kINPUTMUX_DcdcBurstActiveToTimer4Trigger

enumerator kINPUTMUX_Sai0TxSyncOutToTimer4Trigger

enumerator kINPUTMUX_Sai0RxSyncOutToTimer4Trigger

enumerator kINPUTMUX_Adc0IrqToTimer4Trigger

enumerator kINPUTMUX_Adc1IrqToTimer4Trigger

enumerator kINPUTMUX_Cmp0OutToTimer4Trigger

enumerator kINPUTMUX_Cmp1OutToTimer4Trigger

enumerator kINPUTMUX_Cmp2OutToTimer4Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToTimer4Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer4Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToTimer4Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToTimer4Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer4Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToTimer4Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToTimer4Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer4Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer4Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer4Trigger

enumerator kINPUTMUX_EvtgOut0AToTimer4Trigger

enumerator kINPUTMUX_EvtgOut0BToTimer4Trigger

enumerator kINPUTMUX_EvtgOut1AToTimer4Trigger

enumerator kINPUTMUX_EvtgOut1BToTimer4Trigger

enumerator kINPUTMUX_EvtgOut2AToTimer4Trigger

enumerator kINPUTMUX_EvtgOut2BToTimer4Trigger

enumerator kINPUTMUX_EvtgOut3AToTimer4Trigger

enumerator kINPUTMUX_EvtgOut3BToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer4Trigger

enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer4Trigger

enumerator kINPUTMUX_Sai1TxSyncOutToTimer4Trigger

enumerator kINPUTMUX_Sai1RxSyncOutToTimer4Trigger

SMARTDMA arch B inputs.

enumerator kINPUTMUX_FlexioToSmartDma

enumerator kINPUTMUX_GpioPort0Pin0ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin1ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin2ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin3ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin4ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin5ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin6ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin7ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin8ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin9ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin10ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin11ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin12ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin13ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin14ToSmartDma

enumerator kINPUTMUX_GpioPort0Pin15ToSmartDma

enumerator kINPUTMUX_SctOut8ToSmartDma

enumerator kINPUTMUX_SctOut9ToSmartDma

enumerator kINPUTMUX_MrtCh0IrqToSmartDma

enumerator kINPUTMUX_MrtCh1IrqToSmartDma

enumerator kINPUTMUX_Ctimer4M3ToSmartDma

enumerator kINPUTMUX_Ctimer4M2ToSmartDma

enumerator kINPUTMUX_Ctimer3M3ToSmartDma

enumerator kINPUTMUX_Ctimer3M2ToSmartDma

enumerator kINPUTMUX_Ctimer1M3ToSmartDma

enumerator kINPUTMUX_Ctimer1M2ToSmartDma

enumerator kINPUTMUX_UtickIrqToSmartDma

enumerator kINPUTMUX_Wdt0IrqToSmartDma

enumerator kINPUTMUX_Adc0IrqToSmartDma

enumerator kINPUTMUX_Cmp0IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm7IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm6IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm5IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm4IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm3IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm2IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm1IrqToSmartDma

enumerator kINPUTMUX_LpFlexcomm0IrqToSmartDma

enumerator kINPUTMUX_Dma0IrqToSmartDma

enumerator kINPUTMUX_Dma1IrqToSmartDma

enumerator kINPUTMUX_SysIrqToSmartDma

enumerator kINPUTMUX_RtcComboIrqToSmartDma

enumerator kINPUTMUX_ArmTxevToSmartDma

enumerator kINPUTMUX_GpioIntBmatchToSmartDma

enumerator kINPUTMUX_Cmp0OutToSmartDma

enumerator kINPUTMUX_Usb0StartOfFrameIrqToSmartDma

enumerator kINPUTMUX_Usb1StartOfFrameIrqToSmartDma

enumerator kINPUTMUX_OsEventTimerIrqToSmartDma

enumerator kINPUTMUX_Adc1IrqToSmartDma

enumerator kINPUTMUX_Cmp012IrqToSmartDma

enumerator kINPUTMUX_Dac0IrqToSmartDma

enumerator kINPUTMUX_Dac12IrqToSmartDma

enumerator kINPUTMUX_Pwm0IrqToSmartDma

enumerator kINPUTMUX_Pwm1IrqToSmartDma

enumerator kINPUTMUX_Qdc0IrqToSmartDma

enumerator kINPUTMUX_Qdc1IrqToSmartDma

enumerator kINPUTMUX_EvtgOut0AToSmartDma

enumerator kINPUTMUX_EvtgOut1AToSmartDma

enumerator kINPUTMUX_Gpio1PinEventTrig0ToSmartDma

enumerator kINPUTMUX_Gpio1PinEventTrig1ToSmartDma

enumerator kINPUTMUX_Gpio2PinEventTrig0ToSmartDma

enumerator kINPUTMUX_Gpio2PinEventTrig1ToSmartDma

enumerator kINPUTMUX_Gpio3PinEventTrig0ToSmartDma

enumerator kINPUTMUX_Gpio3PinEventTrig1ToSmartDma

Pin interrupt select.

enumerator kINPUTMUX_GpioPort0Pin0ToPintsel

enumerator kINPUTMUX_GpioPort0Pin1ToPintsel

enumerator kINPUTMUX_GpioPort0Pin2ToPintsel

enumerator kINPUTMUX_GpioPort0Pin3ToPintsel

enumerator kINPUTMUX_GpioPort0Pin4ToPintsel

enumerator kINPUTMUX_GpioPort0Pin5ToPintsel

enumerator kINPUTMUX_GpioPort0Pin6ToPintsel

enumerator kINPUTMUX_GpioPort0Pin7ToPintsel

enumerator kINPUTMUX_GpioPort0Pin8ToPintsel

enumerator kINPUTMUX_GpioPort0Pin9ToPintsel

enumerator kINPUTMUX_GpioPort0Pin10ToPintsel

enumerator kINPUTMUX_GpioPort0Pin11ToPintsel

enumerator kINPUTMUX_GpioPort0Pin12ToPintsel

enumerator kINPUTMUX_GpioPort0Pin13ToPintsel

enumerator kINPUTMUX_GpioPort0Pin14ToPintsel

enumerator kINPUTMUX_GpioPort0Pin15ToPintsel

enumerator kINPUTMUX_GpioPort0Pin16ToPintsel

enumerator kINPUTMUX_GpioPort0Pin17ToPintsel

enumerator kINPUTMUX_GpioPort0Pin18ToPintsel

enumerator kINPUTMUX_GpioPort0Pin19ToPintsel

enumerator kINPUTMUX_GpioPort0Pin20ToPintsel

enumerator kINPUTMUX_GpioPort0Pin21ToPintsel

enumerator kINPUTMUX_GpioPort0Pin22ToPintsel

enumerator kINPUTMUX_GpioPort0Pin23ToPintsel

enumerator kINPUTMUX_GpioPort0Pin24ToPintsel

enumerator kINPUTMUX_GpioPort0Pin25ToPintsel

enumerator kINPUTMUX_GpioPort0Pin26ToPintsel

enumerator kINPUTMUX_GpioPort0Pin27ToPintsel

enumerator kINPUTMUX_GpioPort0Pin28ToPintsel

enumerator kINPUTMUX_GpioPort0Pin29ToPintsel

enumerator kINPUTMUX_GpioPort0Pin30ToPintsel

enumerator kINPUTMUX_GpioPort0Pin31ToPintsel

enumerator kINPUTMUX_GpioPort1Pin0ToPintsel

enumerator kINPUTMUX_GpioPort1Pin1ToPintsel

enumerator kINPUTMUX_GpioPort1Pin2ToPintsel

enumerator kINPUTMUX_GpioPort1Pin3ToPintsel

enumerator kINPUTMUX_GpioPort1Pin4ToPintsel

enumerator kINPUTMUX_GpioPort1Pin5ToPintsel

enumerator kINPUTMUX_GpioPort1Pin6ToPintsel

enumerator kINPUTMUX_GpioPort1Pin7ToPintsel

enumerator kINPUTMUX_GpioPort1Pin8ToPintsel

enumerator kINPUTMUX_GpioPort1Pin9ToPintsel

enumerator kINPUTMUX_GpioPort1Pin10ToPintsel

enumerator kINPUTMUX_GpioPort1Pin11ToPintsel

enumerator kINPUTMUX_GpioPort1Pin12ToPintsel

enumerator kINPUTMUX_GpioPort1Pin13ToPintsel

enumerator kINPUTMUX_GpioPort1Pin14ToPintsel

enumerator kINPUTMUX_GpioPort1Pin15ToPintsel

enumerator kINPUTMUX_GpioPort1Pin16ToPintsel

enumerator kINPUTMUX_GpioPort1Pin17ToPintsel

enumerator kINPUTMUX_GpioPort1Pin18ToPintsel

enumerator kINPUTMUX_GpioPort1Pin19ToPintsel

enumerator kINPUTMUX_GpioPort1Pin20ToPintsel

enumerator kINPUTMUX_GpioPort1Pin21ToPintsel

enumerator kINPUTMUX_GpioPort1Pin22ToPintsel

enumerator kINPUTMUX_GpioPort1Pin23ToPintsel

enumerator kINPUTMUX_GpioPort1Pin30ToPintsel

enumerator kINPUTMUX_GpioPort1Pin31ToPintsel

Selection for frequency measurement reference clock.

enumerator kINPUTMUX_ClkInToFreqmeasRef

enumerator kINPUTMUX_Fro12MToFreqmeasRef

enumerator kINPUTMUX_Fro144MToFreqmeasRef

enumerator kINPUTMUX_Osc32KToFreqmeasRef

enumerator kINPUTMUX_CpuAhbClkToFreqmeasRef

enumerator kINPUTMUX_FreqmeClkIn0ToFreqmeasRef

enumerator kINPUTMUX_FreqmeClkIn1ToFreqmeasRef

enumerator kINPUTMUX_EvtgOut0AToFreqmeasRef

enumerator kINPUTMUX_EvtgOut1AToFreqmeasRef

Selection for frequency measurement target clock.

enumerator kINPUTMUX_ClkInToFreqmeasTar

enumerator kINPUTMUX_Fro12MToFreqmeasTar

enumerator kINPUTMUX_Fro144MToFreqmeasTar

enumerator kINPUTMUX_Osc32KToFreqmeasTar

enumerator kINPUTMUX_CpuAhbClkToFreqmeasTar

enumerator kINPUTMUX_FreqmeClkIn0ToFreqmeasTar

enumerator kINPUTMUX_FreqmeClkIn1ToFreqmeasTar

enumerator kINPUTMUX_EvtgOut0AToFreqmeasTar

enumerator kINPUTMUX_EvtgOut1AToFreqmeasTar

Cmp0 Trigger.

enumerator kINPUTMUX_PinInt0ToCmp0Trigger

enumerator kINPUTMUX_PinInt6ToCmp0Trigger

enumerator kINPUTMUX_SctOut4ToCmp0Trigger

enumerator kINPUTMUX_SctOut5ToCmp0Trigger

enumerator kINPUTMUX_SctOut6ToCmp0Trigger

enumerator kINPUTMUX_Ctimer0M3ToCmp0Trigger

enumerator kINPUTMUX_Ctimer1M3ToCmp0Trigger

enumerator kINPUTMUX_Ctimer2M3ToCmp0Trigger

enumerator kINPUTMUX_Ctimer0M0ToCmp0Trigger

enumerator kINPUTMUX_Ctimer4M0ToCmp0Trigger

enumerator kINPUTMUX_ArmTxevToCmp0Trigger

enumerator kINPUTMUX_GpioIntBmatchToCmp0Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToCmp0Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToCmp0Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToCmp0Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToCmp0Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToCmp0Trigger

enumerator kINPUTMUX_EvtgOut0AToCmp0Trigger

enumerator kINPUTMUX_EvtgOut0BToCmp0Trigger

enumerator kINPUTMUX_EvtgOut1AToCmp0Trigger

enumerator kINPUTMUX_EvtgOut1BToCmp0Trigger

enumerator kINPUTMUX_EvtgOut2AToCmp0Trigger

enumerator kINPUTMUX_EvtgOut2BToCmp0Trigger

enumerator kINPUTMUX_EvtgOut3AToCmp0Trigger

enumerator kINPUTMUX_EvtgOut3BToCmp0Trigger

enumerator kINPUTMUX_Lptmr0ToCmp0Trigger

enumerator kINPUTMUX_Lptmr1ToCmp0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToCmp0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToCmp0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToCmp0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToCmp0Trigger
Cmp1 Trigger.

enumerator kINPUTMUX_PinInt0ToCmp1Trigger

enumerator kINPUTMUX_PinInt7ToCmp1Trigger

enumerator kINPUTMUX_SctOut4ToCmp1Trigger

enumerator kINPUTMUX_SctOut5ToCmp1Trigger

enumerator kINPUTMUX_SctOut7ToCmp1Trigger

enumerator kINPUTMUX_Ctimer0M3ToCmp1Trigger

enumerator kINPUTMUX_Ctimer1M3ToCmp1Trigger

enumerator kINPUTMUX_Ctimer2M3ToCmp1Trigger

enumerator kINPUTMUX_Ctimer3M1ToCmp1Trigger

enumerator kINPUTMUX_Ctimer4M1ToCmp1Trigger

enumerator kINPUTMUX_ArmTxevToCmp1Trigger

enumerator kINPUTMUX_GpioIntBmatchToCmp1Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToCmp1Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToCmp1Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToCmp1Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToCmp1Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToCmp1Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToCmp1Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToCmp1Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToCmp1Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToCmp1Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToCmp1Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToCmp1Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToCmp1Trigger

enumerator kINPUTMUX_EvtgOut0AToCmp1Trigger

enumerator kINPUTMUX_EvtgOut0BToCmp1Trigger

enumerator kINPUTMUX_EvtgOut1AToCmp1Trigger

enumerator kINPUTMUX_EvtgOut1BToCmp1Trigger

enumerator kINPUTMUX_EvtgOut2AToCmp1Trigger

enumerator kINPUTMUX_EvtgOut2BToCmp1Trigger

enumerator kINPUTMUX_EvtgOut3AToCmp1Trigger

enumerator kINPUTMUX_EvtgOut3BToCmp1Trigger

enumerator kINPUTMUX_Lptmr0ToCmp1Trigger

enumerator kINPUTMUX_Lptmr1ToCmp1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToCmp1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToCmp1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToCmp1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToCmp1Trigger
Cmp2 Trigger.

enumerator kINPUTMUX_PinInt0ToCmp2Trigger

enumerator kINPUTMUX_PinInt4ToCmp2Trigger

enumerator kINPUTMUX_SctOut4ToCmp2Trigger

enumerator kINPUTMUX_SctOut5ToCmp2Trigger

enumerator kINPUTMUX_SctOut8ToCmp2Trigger

enumerator kINPUTMUX_Ctimer0M3ToCmp2Trigger

enumerator kINPUTMUX_Ctimer1M3ToCmp2Trigger

enumerator kINPUTMUX_Ctimer2M3ToCmp2Trigger

enumerator kINPUTMUX_Ctimer3M2ToCmp2Trigger

enumerator kINPUTMUX_Ctimer4M2ToCmp2Trigger

enumerator kINPUTMUX_ArmTxevToCmp2Trigger

enumerator kINPUTMUX_GpioIntBmatchToCmp2Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToCmp2Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToCmp2Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToCmp2Trigger

enumerator kINPUTMUX_Pwm0A1Trig01ToCmp2Trigger

enumerator kINPUTMUX_Pwm0A2Trig01ToCmp2Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToCmp2Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToCmp2Trigger

enumerator kINPUTMUX_Pwm1A1Trig01ToCmp2Trigger

enumerator kINPUTMUX_Pwm1A2Trig01ToCmp2Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToCmp2Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToCmp2Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToCmp2Trigger

enumerator kINPUTMUX_EvtgOut0AToCmp2Trigger

enumerator kINPUTMUX_EvtgOut0BToCmp2Trigger

enumerator kINPUTMUX_EvtgOut1AToCmp2Trigger

enumerator kINPUTMUX_EvtgOut1BToCmp2Trigger

enumerator kINPUTMUX_EvtgOut2AToCmp2Trigger

enumerator kINPUTMUX_EvtgOut2BToCmp2Trigger

enumerator kINPUTMUX_EvtgOut3AToCmp2Trigger

enumerator kINPUTMUX_EvtgOut3BToCmp2Trigger

enumerator kINPUTMUX_Lptmr0ToCmp2Trigger

enumerator kINPUTMUX_Lptmr1ToCmp2Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToCmp2Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToCmp2Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToCmp2Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToCmp2Trigger

Adc0 Trigger.

enumerator kINPUTMUX_PinInt0ToAdc0Trigger

enumerator kINPUTMUX_PinInt1ToAdc0Trigger

enumerator kINPUTMUX_SctOut4ToAdc0Trigger

enumerator kINPUTMUX_SctOut5ToAdc0Trigger

enumerator kINPUTMUX_SctOut9ToAdc0Trigger

enumerator kINPUTMUX_Ctimer0M3ToAdc0Trigger

enumerator kINPUTMUX_Ctimer1M3ToAdc0Trigger

enumerator kINPUTMUX_Ctimer2M3ToAdc0Trigger

enumerator kINPUTMUX_Ctimer3M3ToAdc0Trigger

enumerator kINPUTMUX_Ctimer4M3ToAdc0Trigger

enumerator kINPUTMUX_DcdcBurstDoneTrigToAdc0Trigger

enumerator kINPUTMUX_ArmTxevToAdc0Trigger

enumerator kINPUTMUX_GpioIntBmatchToAdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToAdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToAdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToAdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToAdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToAdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToAdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToAdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToAdc0Trigger

enumerator kINPUTMUX_Cmp0OutToAdc0Trigger

enumerator kINPUTMUX_Cmp1OutToAdc0Trigger

enumerator kINPUTMUX_Cmp2OutToAdc0Trigger

enumerator kINPUTMUX_Pwm0A0Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm0A0Trig1ToAdc0Trigger

enumerator kINPUTMUX_Pwm0A1Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm0A1Trig1ToAdc0Trigger

enumerator kINPUTMUX_Pwm0A2Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm0A2Trig1ToAdc0Trigger

enumerator kINPUTMUX_Pwm0A3Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm0A3Trig1ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A0Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A0Trig1ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A1Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A1Trig1ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A2Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A3Trig0ToAdc0Trigger

enumerator kINPUTMUX_Pwm1A3Trig1ToAdc0Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToAdc0Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToAdc0Trigger

enumerator kINPUTMUX_EvtgOut0AToAdc0Trigger

enumerator kINPUTMUX_EvtgOut0BToAdc0Trigger

enumerator kINPUTMUX_EvtgOut1AToAdc0Trigger

enumerator kINPUTMUX_EvtgOut1BToAdc0Trigger

enumerator kINPUTMUX_EvtgOut2AToAdc0Trigger

enumerator kINPUTMUX_EvtgOut2BToAdc0Trigger

enumerator kINPUTMUX_EvtgOut3AToAdc0Trigger

enumerator kINPUTMUX_EvtgOut3BToAdc0Trigger

enumerator kINPUTMUX_Lptmr0ToAdc0Trigger

enumerator kINPUTMUX_Lptmr1ToAdc0Trigger

enumerator kINPUTMUX_FlexioCh0ToAdc0Trigger

enumerator kINPUTMUX_FlexioCh1ToAdc0Trigger

enumerator kINPUTMUX_FlexioCh2ToAdc0Trigger

enumerator kINPUTMUX_FlexioCh3ToAdc0Trigger

enumerator kINPUTMUX_SincFilterCh0ToAdc0Trigger

enumerator kINPUTMUX_SincFilterCh1ToAdc0Trigger

enumerator kINPUTMUX_SincFilterCh2ToAdc0Trigger

enumerator kINPUTMUX_SincFilterCh3ToAdc0Trigger

enumerator kINPUTMUX_SincFilterCh4ToAdc0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToAdc0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToAdc0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToAdc0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToAdc0Trigger

enumerator kINPUTMUX_WuuToAdc0Trigger

Adc1 Trigger.

enumerator kINPUTMUX_PinInt0ToAdc1Trigger

enumerator kINPUTMUX_PinInt2ToAdc1Trigger

enumerator kINPUTMUX_SctOut4ToAdc1Trigger

enumerator kINPUTMUX_SctOut5ToAdc1Trigger

enumerator kINPUTMUX_SctOut3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer0M3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer1M3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer2M3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer3M2ToAdc1Trigger

enumerator kINPUTMUX_Ctimer4M1ToAdc1Trigger

enumerator kINPUTMUX_DcdcBurstDoneTrigToAdc1Trigger

enumerator kINPUTMUX_ArmTxevToAdc1Trigger

enumerator kINPUTMUX_GpioIntBmatchToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToAdc1Trigger

enumerator kINPUTMUX_Cmp0OutToAdc1Trigger

enumerator kINPUTMUX_Cmp1OutToAdc1Trigger

enumerator kINPUTMUX_Cmp2OutToAdc1Trigger

enumerator kINPUTMUX_Pwm0A0Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0A0Trig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm0A1Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0A1Trig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm0A2Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0A2Trig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm0A3Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0A3Trig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A0Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A0Trig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A1Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A1Trig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A2Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A3Trig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm1A3Trig1ToAdc1Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToAdc1Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToAdc1Trigger

enumerator kINPUTMUX_EvtgOut0AToAdc1Trigger

enumerator kINPUTMUX_EvtgOut0BToAdc1Trigger

enumerator kINPUTMUX_EvtgOut1AToAdc1Trigger

enumerator kINPUTMUX_EvtgOut1BToAdc1Trigger

enumerator kINPUTMUX_EvtgOut2AToAdc1Trigger

enumerator kINPUTMUX_EvtgOut2BToAdc1Trigger

enumerator kINPUTMUX_EvtgOut3AToAdc1Trigger

enumerator kINPUTMUX_EvtgOut3BToAdc1Trigger

enumerator kINPUTMUX_Lptmr0ToAdc1Trigger

enumerator kINPUTMUX_Lptmr1ToAdc1Trigger

enumerator kINPUTMUX_FlexioCh0ToAdc1Trigger

enumerator kINPUTMUX_FlexioCh1ToAdc1Trigger

enumerator kINPUTMUX_FlexioCh2ToAdc1Trigger

enumerator kINPUTMUX_FlexioCh3ToAdc1Trigger

enumerator kINPUTMUX_SincFilterCh0ToAdc1Trigger

enumerator kINPUTMUX_SincFilterCh1ToAdc1Trigger

enumerator kINPUTMUX_SincFilterCh2ToAdc1Trigger

enumerator kINPUTMUX_SincFilterCh3ToAdc1Trigger

enumerator kINPUTMUX_SincFilterCh4ToAdc1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToAdc1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToAdc1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToAdc1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToAdc1Trigger

enumerator kINPUTMUX_WuuToAdc1Trigger

Dac0 Trigger.

enumerator kINPUTMUX_PinInt0ToDac0Trigger

enumerator kINPUTMUX_PinInt3ToDac0Trigger

enumerator kINPUTMUX_SctOut4ToDac0Trigger

enumerator kINPUTMUX_SctOut5ToDac0Trigger

enumerator kINPUTMUX_SctOut0ToDac0Trigger

enumerator kINPUTMUX_Ctimer0M3ToDac0Trigger

enumerator kINPUTMUX_Ctimer1M3ToDac0Trigger

enumerator kINPUTMUX_Ctimer2M3ToDac0Trigger

enumerator kINPUTMUX_Ctimer2M0ToDac0Trigger

enumerator kINPUTMUX_Ctimer3M0ToDac0Trigger

enumerator kINPUTMUX_ArmTxevToDac0Trigger

enumerator kINPUTMUX_GpioIntBmatchToDac0Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToDac0Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToDac0Trigger

enumerator kINPUTMUX_Cmp0OutToDac0Trigger

enumerator kINPUTMUX_Cmp1OutToDac0Trigger

enumerator kINPUTMUX_Cmp2OutToDac0Trigger

enumerator kINPUTMUX_EvtgOut0AToDac0Trigger

enumerator kINPUTMUX_EvtgOut0BToDac0Trigger

enumerator kINPUTMUX_EvtgOut1AToDac0Trigger

enumerator kINPUTMUX_EvtgOut1BToDac0Trigger

enumerator kINPUTMUX_EvtgOut2AToDac0Trigger

enumerator kINPUTMUX_EvtgOut2BToDac0Trigger

enumerator kINPUTMUX_EvtgOut3AToDac0Trigger

enumerator kINPUTMUX_EvtgOut3BToDac0Trigger

enumerator kINPUTMUX_Lptmr0ToDac0Trigger

enumerator kINPUTMUX_Lptmr1ToDac0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToDac0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToDac0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToDac0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToDac0Trigger

Dac1 Trigger.

enumerator kINPUTMUX_PinInt0ToDac1Trigger

enumerator kINPUTMUX_PinInt4ToDac1Trigger

enumerator kINPUTMUX_SctOut4ToDac1Trigger

enumerator kINPUTMUX_SctOut5ToDac1Trigger

enumerator kINPUTMUX_SctOut1ToDac1Trigger

enumerator kINPUTMUX_Ctimer0M3ToDac1Trigger

enumerator kINPUTMUX_Ctimer1M3ToDac1Trigger

enumerator kINPUTMUX_Ctimer2M3ToDac1Trigger

enumerator kINPUTMUX_Ctimer2M1ToDac1Trigger

enumerator kINPUTMUX_Ctimer3M1ToDac1Trigger

enumerator kINPUTMUX_ArmTxevToDac1Trigger

enumerator kINPUTMUX_GpioIntBmatchToDac1Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToDac1Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToDac1Trigger

enumerator kINPUTMUX_Cmp0OutToDac1Trigger

enumerator kINPUTMUX_Cmp1OutToDac1Trigger

enumerator kINPUTMUX_Cmp2OutToDac1Trigger

enumerator kINPUTMUX_EvtgOut0AToDac1Trigger

enumerator kINPUTMUX_EvtgOut0BToDac1Trigger

enumerator kINPUTMUX_EvtgOut1AToDac1Trigger

enumerator kINPUTMUX_EvtgOut1BToDac1Trigger

enumerator kINPUTMUX_EvtgOut2AToDac1Trigger

enumerator kINPUTMUX_EvtgOut2BToDac1Trigger

enumerator kINPUTMUX_EvtgOut3AToDac1Trigger

enumerator kINPUTMUX_EvtgOut3BToDac1Trigger

enumerator kINPUTMUX_Lptmr0ToDac1Trigger

enumerator kINPUTMUX_Lptmr1ToDac1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToDac1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToDac1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToDac1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToDac1Trigger

Dac2 Trigger.

enumerator kINPUTMUX_PinInt0ToDac2Trigger

enumerator kINPUTMUX_PinInt5ToDac2Trigger

enumerator kINPUTMUX_SctOut4ToDac2Trigger

enumerator kINPUTMUX_SctOut5ToDac2Trigger

enumerator kINPUTMUX_SctOut2ToDac2Trigger

enumerator kINPUTMUX_Ctimer0M3ToDac2Trigger

enumerator kINPUTMUX_Ctimer1M3ToDac2Trigger

enumerator kINPUTMUX_Ctimer2M3ToDac2Trigger

enumerator kINPUTMUX_Ctimer2M2ToDac2Trigger

enumerator kINPUTMUX_Ctimer3M2ToDac2Trigger

enumerator kINPUTMUX_ArmTxevToDac2Trigger

enumerator kINPUTMUX_GpioIntBmatchToDac2Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToDac2Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToDac2Trigger

enumerator kINPUTMUX_Cmp0OutToDac2Trigger

enumerator kINPUTMUX_Cmp1OutToDac2Trigger

enumerator kINPUTMUX_Cmp2OutToDac2Trigger

enumerator kINPUTMUX_EvtgOut0AToDac2Trigger

enumerator kINPUTMUX_EvtgOut0BToDac2Trigger

enumerator kINPUTMUX_EvtgOut1AToDac2Trigger

enumerator kINPUTMUX_EvtgOut1BToDac2Trigger

enumerator kINPUTMUX_EvtgOut2AToDac2Trigger

enumerator kINPUTMUX_EvtgOut2BToDac2Trigger

enumerator kINPUTMUX_EvtgOut3AToDac2Trigger

enumerator kINPUTMUX_EvtgOut3BToDac2Trigger

enumerator kINPUTMUX_Lptmr0ToDac2Trigger

enumerator kINPUTMUX_Lptmr1ToDac2Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToDac2Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToDac2Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToDac2Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToDac2Trigger

QDC0 Trigger Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc0Trigger

enumerator kINPUTMUX_PinInt4ToQdc0Trigger

enumerator kINPUTMUX_SctOut4ToQdc0Trigger

enumerator kINPUTMUX_SctOut5ToQdc0Trigger

enumerator kINPUTMUX_SctOut1ToQdc0Trigger

enumerator kINPUTMUX_Ctimer0M3ToQdc0Trigger

enumerator kINPUTMUX_Ctimer1M3ToQdc0Trigger

enumerator kINPUTMUX_Ctimer2M3ToQdc0Trigger

enumerator kINPUTMUX_Ctimer1M0ToQdc0Trigger

enumerator kINPUTMUX_Ctimer3M0ToQdc0Trigger

enumerator kINPUTMUX_ArmTxevToQdc0Trigger

enumerator kINPUTMUX_GpioIntBmatchToQdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToQdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToQdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToQdc0Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToQdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToQdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToQdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToQdc0Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToQdc0Trigger

enumerator kINPUTMUX_Cmp0OutToQdc0Trigger

enumerator kINPUTMUX_Cmp1OutToQdc0Trigger

enumerator kINPUTMUX_Cmp2OutToQdc0Trigger

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc0Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc0Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc0Trigger

enumerator kINPUTMUX_EvtgOut0AToQdc0Trigger

enumerator kINPUTMUX_EvtgOut0BToQdc0Trigger

enumerator kINPUTMUX_EvtgOut1AToQdc0Trigger

enumerator kINPUTMUX_EvtgOut1BToQdc0Trigger

enumerator kINPUTMUX_EvtgOut2AToQdc0Trigger

enumerator kINPUTMUX_EvtgOut2BToQdc0Trigger

enumerator kINPUTMUX_EvtgOut3AToQdc0Trigger

enumerator kINPUTMUX_EvtgOut3BToQdc0Trigger

enumerator kINPUTMUX_TrigIn0ToQdc0Trigger

enumerator kINPUTMUX_TrigIn1ToQdc0Trigger

enumerator kINPUTMUX_TrigIn2ToQdc0Trigger

enumerator kINPUTMUX_TrigIn3ToQdc0Trigger

enumerator kINPUTMUX_TrigIn4ToQdc0Trigger

enumerator kINPUTMUX_TrigIn5ToQdc0Trigger

enumerator kINPUTMUX_TrigIn6ToQdc0Trigger

enumerator kINPUTMUX_TrigIn7ToQdc0Trigger

enumerator kINPUTMUX_TrigIn8ToQdc0Trigger

enumerator kINPUTMUX_TrigIn9ToQdc0Trigger

QDC0 Home Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc0Home

enumerator kINPUTMUX_PinInt4ToQdc0Home

enumerator kINPUTMUX_SctOut4ToQdc0Home

enumerator kINPUTMUX_SctOut5ToQdc0Home

enumerator kINPUTMUX_SctOut1ToQdc0Home

enumerator kINPUTMUX_Ctimer0M3ToQdc0Home

enumerator kINPUTMUX_Ctimer1M3ToQdc0Home

enumerator kINPUTMUX_Ctimer2M3ToQdc0Home

enumerator kINPUTMUX_Ctimer1M0ToQdc0Home

enumerator kINPUTMUX_Ctimer3M0ToQdc0Home

enumerator kINPUTMUX_ArmTxevToQdc0Home

enumerator kINPUTMUX_GpioIntBmatchToQdc0Home

enumerator kINPUTMUX_Adc0Tcomp0ToQdc0Home

enumerator kINPUTMUX_Adc0Tcomp1ToQdc0Home

enumerator kINPUTMUX_Adc0Tcomp2ToQdc0Home

enumerator kINPUTMUX_Adc0Tcomp3ToQdc0Home

enumerator kINPUTMUX_Adc1Tcomp0ToQdc0Home

enumerator kINPUTMUX_Adc1Tcomp1ToQdc0Home

enumerator kINPUTMUX_Adc1Tcomp2ToQdc0Home

enumerator kINPUTMUX_Adc1Tcomp3ToQdc0Home

enumerator kINPUTMUX_Cmp0OutToQdc0Home

enumerator kINPUTMUX_Cmp1OutToQdc0Home

enumerator kINPUTMUX_Cmp2OutToQdc0Home

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc0Home

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc0Home

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc0Home

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc0Home

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc0Home

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc0Home

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc0Home

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc0Home

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc0Home

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc0Home

enumerator kINPUTMUX_EvtgOut0AToQdc0Home

enumerator kINPUTMUX_EvtgOut0BToQdc0Home

enumerator kINPUTMUX_EvtgOut1AToQdc0Home

enumerator kINPUTMUX_EvtgOut1BToQdc0Home

enumerator kINPUTMUX_EvtgOut2AToQdc0Home

enumerator kINPUTMUX_EvtgOut2BToQdc0Home

enumerator kINPUTMUX_EvtgOut3AToQdc0Home

enumerator kINPUTMUX_EvtgOut3BToQdc0Home

enumerator kINPUTMUX_TrigIn0ToQdc0Home

enumerator kINPUTMUX_TrigIn1ToQdc0Home

enumerator kINPUTMUX_TrigIn2ToQdc0Home

enumerator kINPUTMUX_TrigIn3ToQdc0Home

enumerator kINPUTMUX_TrigIn4ToQdc0Home

enumerator kINPUTMUX_TrigIn5ToQdc0Home

enumerator kINPUTMUX_TrigIn6ToQdc0Home

enumerator kINPUTMUX_TrigIn7ToQdc0Home

enumerator kINPUTMUX_TrigIn8ToQdc0Home

enumerator kINPUTMUX_TrigIn9ToQdc0Home

QDC0 Index Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc0Index

enumerator kINPUTMUX_PinInt4ToQdc0Index

enumerator kINPUTMUX_SctOut4ToQdc0Index

enumerator kINPUTMUX_SctOut5ToQdc0Index

enumerator kINPUTMUX_SctOut1ToQdc0Index

enumerator kINPUTMUX_Ctimer0M3ToQdc0Index

enumerator kINPUTMUX_Ctimer1M3ToQdc0Index

enumerator kINPUTMUX_Ctimer2M3ToQdc0Index

enumerator kINPUTMUX_Ctimer1M0ToQdc0Index

enumerator kINPUTMUX_Ctimer3M0ToQdc0Index

enumerator kINPUTMUX_ArmTxevToQdc0Index

enumerator kINPUTMUX_GpioIntBmatchToQdc0Index

enumerator kINPUTMUX_Adc0Tcomp0ToQdc0Index

enumerator kINPUTMUX_Adc0Tcomp1ToQdc0Index

enumerator kINPUTMUX_Adc0Tcomp2ToQdc0Index

enumerator kINPUTMUX_Adc0Tcomp3ToQdc0Index

enumerator kINPUTMUX_Adc1Tcomp0ToQdc0Index

enumerator kINPUTMUX_Adc1Tcomp1ToQdc0Index

enumerator kINPUTMUX_Adc1Tcomp2ToQdc0Index

enumerator kINPUTMUX_Adc1Tcomp3ToQdc0Index

enumerator kINPUTMUX_Cmp0OutToQdc0Index

enumerator kINPUTMUX_Cmp1OutToQdc0Index

enumerator kINPUTMUX_Cmp2OutToQdc0Index

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc0Index

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc0Index

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc0Index

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc0Index

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc0Index

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc0Index

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc0Index

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc0Index

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc0Index

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc0Index

enumerator kINPUTMUX_EvtgOut0AToQdc0Index

enumerator kINPUTMUX_EvtgOut0BToQdc0Index

enumerator kINPUTMUX_EvtgOut1AToQdc0Index

enumerator kINPUTMUX_EvtgOut1BToQdc0Index

enumerator kINPUTMUX_EvtgOut2AToQdc0Index

enumerator kINPUTMUX_EvtgOut2BToQdc0Index

enumerator kINPUTMUX_EvtgOut3AToQdc0Index

enumerator kINPUTMUX_EvtgOut3BToQdc0Index

enumerator kINPUTMUX_TrigIn0ToQdc0Index

enumerator kINPUTMUX_TrigIn1ToQdc0Index

enumerator kINPUTMUX_TrigIn2ToQdc0Index

enumerator kINPUTMUX_TrigIn3ToQdc0Index

enumerator kINPUTMUX_TrigIn4ToQdc0Index

enumerator kINPUTMUX_TrigIn5ToQdc0Index

enumerator kINPUTMUX_TrigIn6ToQdc0Index

enumerator kINPUTMUX_TrigIn7ToQdc0Index

enumerator kINPUTMUX_TrigIn8ToQdc0Index

enumerator kINPUTMUX_TrigIn9ToQdc0Index

QDC0 Phaseb Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc0Phaseb

enumerator kINPUTMUX_PinInt4ToQdc0Phaseb

enumerator kINPUTMUX_SctOut4ToQdc0Phaseb

enumerator kINPUTMUX_SctOut5ToQdc0Phaseb

enumerator kINPUTMUX_SctOut1ToQdc0Phaseb

enumerator kINPUTMUX_Ctimer0M3ToQdc0Phaseb

enumerator kINPUTMUX_Ctimer1M3ToQdc0Phaseb

enumerator kINPUTMUX_Ctimer2M3ToQdc0Phaseb

enumerator kINPUTMUX_Ctimer1M0ToQdc0Phaseb

enumerator kINPUTMUX_Ctimer3M0ToQdc0Phaseb

enumerator kINPUTMUX_ArmTxevToQdc0Phaseb

enumerator kINPUTMUX_GpioIntBmatchToQdc0Phaseb

enumerator kINPUTMUX_Adc0Tcomp0ToQdc0Phaseb

enumerator kINPUTMUX_Adc0Tcomp1ToQdc0Phaseb

enumerator kINPUTMUX_Adc0Tcomp2ToQdc0Phaseb

enumerator kINPUTMUX_Adc0Tcomp3ToQdc0Phaseb

enumerator kINPUTMUX_Adc1Tcomp0ToQdc0Phaseb

enumerator kINPUTMUX_Adc1Tcomp1ToQdc0Phaseb

enumerator kINPUTMUX_Adc1Tcomp2ToQdc0Phaseb

enumerator kINPUTMUX_Adc1Tcomp3ToQdc0Phaseb

enumerator kINPUTMUX_Cmp0OutToQdc0Phaseb

enumerator kINPUTMUX_Cmp1OutToQdc0Phaseb

enumerator kINPUTMUX_Cmp2OutToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc0Phaseb

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc0Phaseb

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc0Phaseb

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut0AToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut0BToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut1AToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut1BToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut2AToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut2BToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut3AToQdc0Phaseb

enumerator kINPUTMUX_EvtgOut3BToQdc0Phaseb

enumerator kINPUTMUX_TrigIn0ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn1ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn2ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn3ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn4ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn5ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn6ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn7ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn8ToQdc0Phaseb

enumerator kINPUTMUX_TrigIn9ToQdc0Phaseb

QDC0 Phasea Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc0Phasea

enumerator kINPUTMUX_PinInt4ToQdc0Phasea

enumerator kINPUTMUX_SctOut4ToQdc0Phasea

enumerator kINPUTMUX_SctOut5ToQdc0Phasea

enumerator kINPUTMUX_SctOut1ToQdc0Phasea

enumerator kINPUTMUX_Ctimer0M3ToQdc0Phasea

enumerator kINPUTMUX_Ctimer1M3ToQdc0Phasea

enumerator kINPUTMUX_Ctimer2M3ToQdc0Phasea

enumerator kINPUTMUX_Ctimer1M0ToQdc0Phasea

enumerator kINPUTMUX_Ctimer3M0ToQdc0Phasea

enumerator kINPUTMUX_ArmTxevToQdc0Phasea

enumerator kINPUTMUX_GpioIntBmatchToQdc0Phasea

enumerator kINPUTMUX_Adc0Tcomp0ToQdc0Phasea

enumerator kINPUTMUX_Adc0Tcomp1ToQdc0Phasea

enumerator kINPUTMUX_Adc0Tcomp2ToQdc0Phasea

enumerator kINPUTMUX_Adc0Tcomp3ToQdc0Phasea

enumerator kINPUTMUX_Adc1Tcomp0ToQdc0Phasea

enumerator kINPUTMUX_Adc1Tcomp1ToQdc0Phasea

enumerator kINPUTMUX_Adc1Tcomp2ToQdc0Phasea

enumerator kINPUTMUX_Adc1Tcomp3ToQdc0Phasea

enumerator kINPUTMUX_Cmp0OutToQdc0Phasea

enumerator kINPUTMUX_Cmp1OutToQdc0Phasea

enumerator kINPUTMUX_Cmp2OutToQdc0Phasea

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc0Phasea

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc0Phasea

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc0Phasea

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc0Phasea

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc0Phasea

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc0Phasea

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc0Phasea

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc0Phasea

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc0Phasea

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc0Phasea

enumerator kINPUTMUX_EvtgOut0AToQdc0Phasea

enumerator kINPUTMUX_EvtgOut0BToQdc0Phasea

enumerator kINPUTMUX_EvtgOut1AToQdc0Phasea

enumerator kINPUTMUX_EvtgOut1BToQdc0Phasea

enumerator kINPUTMUX_EvtgOut2AToQdc0Phasea

enumerator kINPUTMUX_EvtgOut2BToQdc0Phasea

enumerator kINPUTMUX_EvtgOut3AToQdc0Phasea

enumerator kINPUTMUX_EvtgOut3BToQdc0Phasea

enumerator kINPUTMUX_TrigIn0ToQdc0Phasea

enumerator kINPUTMUX_TrigIn1ToQdc0Phasea

enumerator kINPUTMUX_TrigIn2ToQdc0Phasea

enumerator kINPUTMUX_TrigIn3ToQdc0Phasea

enumerator kINPUTMUX_TrigIn4ToQdc0Phasea

enumerator kINPUTMUX_TrigIn5ToQdc0Phasea

enumerator kINPUTMUX_TrigIn6ToQdc0Phasea

enumerator kINPUTMUX_TrigIn7ToQdc0Phasea

enumerator kINPUTMUX_TrigIn8ToQdc0Phasea

enumerator kINPUTMUX_TrigIn9ToQdc0Phasea

QDC1 Trigger Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc1Trigger

enumerator kINPUTMUX_PinInt4ToQdc1Trigger

enumerator kINPUTMUX_SctOut4ToQdc1Trigger

enumerator kINPUTMUX_SctOut5ToQdc1Trigger

enumerator kINPUTMUX_SctOut1ToQdc1Trigger

enumerator kINPUTMUX_Ctimer0M3ToQdc1Trigger

enumerator kINPUTMUX_Ctimer1M3ToQdc1Trigger

enumerator kINPUTMUX_Ctimer2M3ToQdc1Trigger

enumerator kINPUTMUX_Ctimer1M0ToQdc1Trigger

enumerator kINPUTMUX_Ctimer3M0ToQdc1Trigger

enumerator kINPUTMUX_ArmTxevToQdc1Trigger

enumerator kINPUTMUX_GpioIntBmatchToQdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToQdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToQdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToQdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToQdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToQdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToQdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToQdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToQdc1Trigger

enumerator kINPUTMUX_Cmp0OutToQdc1Trigger

enumerator kINPUTMUX_Cmp1OutToQdc1Trigger

enumerator kINPUTMUX_Cmp2OutToQdc1Trigger

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc1Trigger

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc1Trigger

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Trigger

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Trigger

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Trigger

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Trigger

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Trigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Trigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Trigger

enumerator kINPUTMUX_EvtgOut0AToQdc1Trigger

enumerator kINPUTMUX_EvtgOut0BToQdc1Trigger

enumerator kINPUTMUX_EvtgOut1AToQdc1Trigger

enumerator kINPUTMUX_EvtgOut1BToQdc1Trigger

enumerator kINPUTMUX_EvtgOut2AToQdc1Trigger

enumerator kINPUTMUX_EvtgOut2BToQdc1Trigger

enumerator kINPUTMUX_EvtgOut3AToQdc1Trigger

enumerator kINPUTMUX_EvtgOut3BToQdc1Trigger

enumerator kINPUTMUX_TrigIn0ToQdc1Trigger

enumerator kINPUTMUX_TrigIn1ToQdc1Trigger

enumerator kINPUTMUX_TrigIn2ToQdc1Trigger

enumerator kINPUTMUX_TrigIn3ToQdc1Trigger

enumerator kINPUTMUX_TrigIn4ToQdc1Trigger

enumerator kINPUTMUX_TrigIn5ToQdc1Trigger

enumerator kINPUTMUX_TrigIn6ToQdc1Trigger

enumerator kINPUTMUX_TrigIn7ToQdc1Trigger

enumerator kINPUTMUX_TrigIn8ToQdc1Trigger

enumerator kINPUTMUX_TrigIn9ToQdc1Trigger

QDC1 Home Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc1Home

enumerator kINPUTMUX_PinInt4ToQdc1Home

enumerator kINPUTMUX_SctOut4ToQdc1Home

enumerator kINPUTMUX_SctOut5ToQdc1Home

enumerator kINPUTMUX_SctOut1ToQdc1Home

enumerator kINPUTMUX_Ctimer0M3ToQdc1Home

enumerator kINPUTMUX_Ctimer1M3ToQdc1Home

enumerator kINPUTMUX_Ctimer2M3ToQdc1Home

enumerator kINPUTMUX_Ctimer1M0ToQdc1Home

enumerator kINPUTMUX_Ctimer3M0ToQdc1Home

enumerator kINPUTMUX_ArmTxevToQdc1Home

enumerator kINPUTMUX_GpioIntBmatchToQdc1Home

enumerator kINPUTMUX_Adc0Tcomp0ToQdc1Home

enumerator kINPUTMUX_Adc0Tcomp1ToQdc1Home

enumerator kINPUTMUX_Adc0Tcomp2ToQdc1Home

enumerator kINPUTMUX_Adc0Tcomp3ToQdc1Home

enumerator kINPUTMUX_Adc1Tcomp0ToQdc1Home

enumerator kINPUTMUX_Adc1Tcomp1ToQdc1Home

enumerator kINPUTMUX_Adc1Tcomp2ToQdc1Home

enumerator kINPUTMUX_Adc1Tcomp3ToQdc1Home

enumerator kINPUTMUX_Cmp0OutToQdc1Home

enumerator kINPUTMUX_Cmp1OutToQdc1Home

enumerator kINPUTMUX_Cmp2OutToQdc1Home

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc1Home

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc1Home

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Home

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Home

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Home

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Home

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Home

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Home

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Home

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Home

enumerator kINPUTMUX_EvtgOut0AToQdc1Home

enumerator kINPUTMUX_EvtgOut0BToQdc1Home

enumerator kINPUTMUX_EvtgOut1AToQdc1Home

enumerator kINPUTMUX_EvtgOut1BToQdc1Home

enumerator kINPUTMUX_EvtgOut2AToQdc1Home

enumerator kINPUTMUX_EvtgOut2BToQdc1Home

enumerator kINPUTMUX_EvtgOut3AToQdc1Home

enumerator kINPUTMUX_EvtgOut3BToQdc1Home

enumerator kINPUTMUX_TrigIn0ToQdc1Home

enumerator kINPUTMUX_TrigIn1ToQdc1Home

enumerator kINPUTMUX_TrigIn2ToQdc1Home

enumerator kINPUTMUX_TrigIn3ToQdc1Home

enumerator kINPUTMUX_TrigIn4ToQdc1Home

enumerator kINPUTMUX_TrigIn5ToQdc1Home

enumerator kINPUTMUX_TrigIn6ToQdc1Home

enumerator kINPUTMUX_TrigIn7ToQdc1Home

enumerator kINPUTMUX_TrigIn8ToQdc1Home

enumerator kINPUTMUX_TrigIn9ToQdc1Home

QDC1 Index Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc1Index

enumerator kINPUTMUX_PinInt4ToQdc1Index

enumerator kINPUTMUX_SctOut4ToQdc1Index

enumerator kINPUTMUX_SctOut5ToQdc1Index

enumerator kINPUTMUX_SctOut1ToQdc1Index

enumerator kINPUTMUX_Ctimer0M3ToQdc1Index

enumerator kINPUTMUX_Ctimer1M3ToQdc1Index

enumerator kINPUTMUX_Ctimer2M3ToQdc1Index

enumerator kINPUTMUX_Ctimer1M0ToQdc1Index

enumerator kINPUTMUX_Ctimer3M0ToQdc1Index

enumerator kINPUTMUX_ArmTxevToQdc1Index

enumerator kINPUTMUX_GpioIntBmatchToQdc1Index

enumerator kINPUTMUX_Adc0Tcomp0ToQdc1Index

enumerator kINPUTMUX_Adc0Tcomp1ToQdc1Index

enumerator kINPUTMUX_Adc0Tcomp2ToQdc1Index

enumerator kINPUTMUX_Adc0Tcomp3ToQdc1Index

enumerator kINPUTMUX_Adc1Tcomp0ToQdc1Index

enumerator kINPUTMUX_Adc1Tcomp1ToQdc1Index

enumerator kINPUTMUX_Adc1Tcomp2ToQdc1Index

enumerator kINPUTMUX_Adc1Tcomp3ToQdc1Index

enumerator kINPUTMUX_Cmp0OutToQdc1Index

enumerator kINPUTMUX_Cmp1OutToQdc1Index

enumerator kINPUTMUX_Cmp2OutToQdc1Index

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc1Index

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc1Index

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Index

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Index

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Index

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Index

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Index

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Index

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Index

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Index

enumerator kINPUTMUX_EvtgOut0AToQdc1Index

enumerator kINPUTMUX_EvtgOut0BToQdc1Index

enumerator kINPUTMUX_EvtgOut1AToQdc1Index

enumerator kINPUTMUX_EvtgOut1BToQdc1Index

enumerator kINPUTMUX_EvtgOut2AToQdc1Index

enumerator kINPUTMUX_EvtgOut2BToQdc1Index

enumerator kINPUTMUX_EvtgOut3AToQdc1Index

enumerator kINPUTMUX_EvtgOut3BToQdc1Index

enumerator kINPUTMUX_TrigIn0ToQdc1Index

enumerator kINPUTMUX_TrigIn1ToQdc1Index

enumerator kINPUTMUX_TrigIn2ToQdc1Index

enumerator kINPUTMUX_TrigIn3ToQdc1Index

enumerator kINPUTMUX_TrigIn4ToQdc1Index

enumerator kINPUTMUX_TrigIn5ToQdc1Index

enumerator kINPUTMUX_TrigIn6ToQdc1Index

enumerator kINPUTMUX_TrigIn7ToQdc1Index

enumerator kINPUTMUX_TrigIn8ToQdc1Index

enumerator kINPUTMUX_TrigIn9ToQdc1Index

QDC1 Phaseb Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc1Phaseb

enumerator kINPUTMUX_PinInt4ToQdc1Phaseb

enumerator kINPUTMUX_SctOut4ToQdc1Phaseb

enumerator kINPUTMUX_SctOut5ToQdc1Phaseb

enumerator kINPUTMUX_SctOut1ToQdc1Phaseb

enumerator kINPUTMUX_Ctimer0M3ToQdc1Phaseb

enumerator kINPUTMUX_Ctimer1M3ToQdc1Phaseb

enumerator kINPUTMUX_Ctimer2M3ToQdc1Phaseb

enumerator kINPUTMUX_Ctimer1M0ToQdc1Phaseb

enumerator kINPUTMUX_Ctimer3M0ToQdc1Phaseb

enumerator kINPUTMUX_ArmTxevToQdc1Phaseb

enumerator kINPUTMUX_GpioIntBmatchToQdc1Phaseb

enumerator kINPUTMUX_Adc0Tcomp0ToQdc1Phaseb

enumerator kINPUTMUX_Adc0Tcomp1ToQdc1Phaseb

enumerator kINPUTMUX_Adc0Tcomp2ToQdc1Phaseb

enumerator kINPUTMUX_Adc0Tcomp3ToQdc1Phaseb

enumerator kINPUTMUX_Adc1Tcomp0ToQdc1Phaseb

enumerator kINPUTMUX_Adc1Tcomp1ToQdc1Phaseb

enumerator kINPUTMUX_Adc1Tcomp2ToQdc1Phaseb

enumerator kINPUTMUX_Adc1Tcomp3ToQdc1Phaseb

enumerator kINPUTMUX_Cmp0OutToQdc1Phaseb

enumerator kINPUTMUX_Cmp1OutToQdc1Phaseb

enumerator kINPUTMUX_Cmp2OutToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Phaseb

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Phaseb

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Phaseb

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut0AToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut0BToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut1AToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut1BToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut2AToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut2BToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut3AToQdc1Phaseb

enumerator kINPUTMUX_EvtgOut3BToQdc1Phaseb

enumerator kINPUTMUX_TrigIn0ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn1ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn2ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn3ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn4ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn5ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn6ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn7ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn8ToQdc1Phaseb

enumerator kINPUTMUX_TrigIn9ToQdc1Phaseb

QDC1 Phasea Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc1Phasea

enumerator kINPUTMUX_PinInt4ToQdc1Phasea

enumerator kINPUTMUX_SctOut4ToQdc1Phasea

enumerator kINPUTMUX_SctOut5ToQdc1Phasea

enumerator kINPUTMUX_SctOut1ToQdc1Phasea

enumerator kINPUTMUX_Ctimer0M3ToQdc1Phasea

enumerator kINPUTMUX_Ctimer1M3ToQdc1Phasea

enumerator kINPUTMUX_Ctimer2M3ToQdc1Phasea

enumerator kINPUTMUX_Ctimer1M0ToQdc1Phasea

enumerator kINPUTMUX_Ctimer3M0ToQdc1Phasea

enumerator kINPUTMUX_ArmTxevToQdc1Phasea

enumerator kINPUTMUX_GpioIntBmatchToQdc1Phasea

enumerator kINPUTMUX_Adc0Tcomp0ToQdc1Phasea

enumerator kINPUTMUX_Adc0Tcomp1ToQdc1Phasea

enumerator kINPUTMUX_Adc0Tcomp2ToQdc1Phasea

enumerator kINPUTMUX_Adc0Tcomp3ToQdc1Phasea

enumerator kINPUTMUX_Adc1Tcomp0ToQdc1Phasea

enumerator kINPUTMUX_Adc1Tcomp1ToQdc1Phasea

enumerator kINPUTMUX_Adc1Tcomp2ToQdc1Phasea

enumerator kINPUTMUX_Adc1Tcomp3ToQdc1Phasea

enumerator kINPUTMUX_Cmp0OutToQdc1Phasea

enumerator kINPUTMUX_Cmp1OutToQdc1Phasea

enumerator kINPUTMUX_Cmp2OutToQdc1Phasea

enumerator kINPUTMUX_Pwm1A0Trig0ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A0Trig1ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Phasea

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Phasea

enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Phasea

enumerator kINPUTMUX_EvtgOut0AToQdc1Phasea

enumerator kINPUTMUX_EvtgOut0BToQdc1Phasea

enumerator kINPUTMUX_EvtgOut1AToQdc1Phasea

enumerator kINPUTMUX_EvtgOut1BToQdc1Phasea

enumerator kINPUTMUX_EvtgOut2AToQdc1Phasea

enumerator kINPUTMUX_EvtgOut2BToQdc1Phasea

enumerator kINPUTMUX_EvtgOut3AToQdc1Phasea

enumerator kINPUTMUX_EvtgOut3BToQdc1Phasea

enumerator kINPUTMUX_TrigIn0ToQdc1Phasea

enumerator kINPUTMUX_TrigIn1ToQdc1Phasea

enumerator kINPUTMUX_TrigIn2ToQdc1Phasea

enumerator kINPUTMUX_TrigIn3ToQdc1Phasea

enumerator kINPUTMUX_TrigIn4ToQdc1Phasea

enumerator kINPUTMUX_TrigIn5ToQdc1Phasea

enumerator kINPUTMUX_TrigIn6ToQdc1Phasea

enumerator kINPUTMUX_TrigIn7ToQdc1Phasea

enumerator kINPUTMUX_TrigIn8ToQdc1Phasea

enumerator kINPUTMUX_TrigIn9ToQdc1Phasea

FlexPWM0_SM0_EXTSYNC input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm0ExtSync

FlexPWM0_SM1_EXTSYNC input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm1ExtSync

FlexPWM0_SM2_EXTSYNC2 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm2ExtSync

FlexPWM0_SM3_EXTSYNC input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm3ExtSync

FlexPWM0_SM0_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm0Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm0Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm0Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm0Exta

FlexPWM0_SM1_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm1Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm1Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm1Exta

FlexPWM0_SM2_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm2Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm2Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm2Exta

FlexPWM0_SM3_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SctOut2ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm3Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm3Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm3Exta

FlexPWM0_EXTFORCE input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0ExtForce

enumerator kINPUTMUX_PinInt5ToFlexPwm0ExtForce

enumerator kINPUTMUX_SctOut4ToFlexPwm0ExtForce

enumerator kINPUTMUX_SctOut5ToFlexPwm0ExtForce

enumerator kINPUTMUX_SctOut2ToFlexPwm0ExtForce

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0ExtForce

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0ExtForce

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0ExtForce

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0ExtForce

enumerator kINPUTMUX_ArmTxevToFlexPwm0ExtForce

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0ExtForce

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0ExtForce

enumerator kINPUTMUX_Cmp0OutToFlexPwm0ExtForce

enumerator kINPUTMUX_Cmp1OutToFlexPwm0ExtForce

enumerator kINPUTMUX_Cmp2OutToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0ExtForce

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0ExtForce

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0ExtForce

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn0ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn1ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn2ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn3ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn4ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn5ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn6ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn7ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn8ToFlexPwm0ExtForce

enumerator kINPUTMUX_TrigIn9ToFlexPwm0ExtForce

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0ExtForce

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0ExtForce

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0ExtForce

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0ExtForce

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0ExtForce

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0ExtForce

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0ExtForce

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0ExtForce

FlexPWM0_FAULT0 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Fault0

enumerator kINPUTMUX_PinInt5ToFlexPwm0Fault0

enumerator kINPUTMUX_SctOut4ToFlexPwm0Fault0

enumerator kINPUTMUX_SctOut5ToFlexPwm0Fault0

enumerator kINPUTMUX_SctOut2ToFlexPwm0Fault0

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Fault0

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Fault0

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Fault0

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Fault0

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Fault0

enumerator kINPUTMUX_ArmTxevToFlexPwm0Fault0

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Fault0

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Fault0

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Fault0

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Fault0

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Fault0

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Fault0

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Fault0

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Fault0

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Fault0

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Fault0

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Fault0

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Fault0

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Fault0

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Fault0

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Fault0

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Fault0

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Fault0

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Fault0

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Fault0

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Fault0

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Fault0

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Fault0

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Fault0

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Fault0

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Fault0

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault0

FlexPWM0_FAULT1 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Fault1

enumerator kINPUTMUX_PinInt5ToFlexPwm0Fault1

enumerator kINPUTMUX_SctOut4ToFlexPwm0Fault1

enumerator kINPUTMUX_SctOut5ToFlexPwm0Fault1

enumerator kINPUTMUX_SctOut2ToFlexPwm0Fault1

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Fault1

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Fault1

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Fault1

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Fault1

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Fault1

enumerator kINPUTMUX_ArmTxevToFlexPwm0Fault1

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Fault1

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Fault1

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Fault1

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Fault1

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Fault1

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Fault1

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Fault1

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Fault1

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Fault1

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Fault1

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Fault1

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Fault1

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Fault1

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Fault1

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Fault1

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Fault1

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Fault1

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Fault1

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Fault1

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Fault1

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Fault1

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Fault1

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Fault1

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Fault1

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Fault1

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault1

FlexPWM0_FAULT2 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Fault2

enumerator kINPUTMUX_PinInt5ToFlexPwm0Fault2

enumerator kINPUTMUX_SctOut4ToFlexPwm0Fault2

enumerator kINPUTMUX_SctOut5ToFlexPwm0Fault2

enumerator kINPUTMUX_SctOut2ToFlexPwm0Fault2

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Fault2

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Fault2

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Fault2

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Fault2

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Fault2

enumerator kINPUTMUX_ArmTxevToFlexPwm0Fault2

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Fault2

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Fault2

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Fault2

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Fault2

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Fault2

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Fault2

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Fault2

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Fault2

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Fault2

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Fault2

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Fault2

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Fault2

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Fault2

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Fault2

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Fault2

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Fault2

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Fault2

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Fault2

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Fault2

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Fault2

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Fault2

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Fault2

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Fault2

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Fault2

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Fault2

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault2

FlexPWM0_FAULT3 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Fault3

enumerator kINPUTMUX_PinInt5ToFlexPwm0Fault3

enumerator kINPUTMUX_SctOut4ToFlexPwm0Fault3

enumerator kINPUTMUX_SctOut5ToFlexPwm0Fault3

enumerator kINPUTMUX_SctOut2ToFlexPwm0Fault3

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Fault3

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Fault3

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Fault3

enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Fault3

enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Fault3

enumerator kINPUTMUX_ArmTxevToFlexPwm0Fault3

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Fault3

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Fault3

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Fault3

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Fault3

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Fault3

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Fault3

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Fault3

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Fault3

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Fault3

enumerator kINPUTMUX_Cmp0OutToFlexPwm0Fault3

enumerator kINPUTMUX_Cmp1OutToFlexPwm0Fault3

enumerator kINPUTMUX_Cmp2OutToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Fault3

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Fault3

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Fault3

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Fault3

enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn0ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn1ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn2ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn3ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn4ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn5ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn6ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn7ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn8ToFlexPwm0Fault3

enumerator kINPUTMUX_TrigIn9ToFlexPwm0Fault3

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm0Fault3

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm0Fault3

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm0Fault3

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm0Fault3

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm0Fault3

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Fault3

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Fault3

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Fault3

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault3

FlexPWM1_SM0_EXTSYNC input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm0ExtSync

FlexPWM1_SM1_EXTSYNC input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm1ExtSync

FlexPWM1_SM2_EXTSYNC2 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm2ExtSync

FlexPWM1_SM3_EXTSYNC input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm3ExtSync

FlexPWM1_SM0_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm0Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm0Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm0Exta

FlexPWM1_SM1_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm1Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm1Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm1Exta

FlexPWM1_SM2_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm2Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm2Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm2Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm2Exta

FlexPWM1_SM3_EXTA input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SctOut4ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SctOut5ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SctOut3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm3Exta

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm3Exta

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm3Exta

FlexPWM1_EXTFORCE input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1ExtForce

enumerator kINPUTMUX_PinInt2ToFlexPwm1ExtForce

enumerator kINPUTMUX_SctOut4ToFlexPwm1ExtForce

enumerator kINPUTMUX_SctOut5ToFlexPwm1ExtForce

enumerator kINPUTMUX_SctOut3ToFlexPwm1ExtForce

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1ExtForce

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1ExtForce

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1ExtForce

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1ExtForce

enumerator kINPUTMUX_ArmTxevToFlexPwm1ExtForce

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1ExtForce

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1ExtForce

enumerator kINPUTMUX_Cmp0OutToFlexPwm1ExtForce

enumerator kINPUTMUX_Cmp1OutToFlexPwm1ExtForce

enumerator kINPUTMUX_Cmp2OutToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1ExtForce

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1ExtForce

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn0ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn1ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn2ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn3ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn4ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn5ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn6ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn7ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn8ToFlexPwm1ExtForce

enumerator kINPUTMUX_TrigIn9ToFlexPwm1ExtForce

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1ExtForce

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1ExtForce

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1ExtForce

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1ExtForce

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1ExtForce

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1ExtForce

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1ExtForce

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1ExtForce

FlexPWM1_FAULT0 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Fault0

enumerator kINPUTMUX_PinInt2ToFlexPwm1Fault0

enumerator kINPUTMUX_SctOut4ToFlexPwm1Fault0

enumerator kINPUTMUX_SctOut5ToFlexPwm1Fault0

enumerator kINPUTMUX_SctOut3ToFlexPwm1Fault0

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Fault0

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Fault0

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Fault0

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Fault0

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Fault0

enumerator kINPUTMUX_ArmTxevToFlexPwm1Fault0

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Fault0

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Fault0

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Fault0

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Fault0

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Fault0

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Fault0

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Fault0

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Fault0

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Fault0

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Fault0

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Fault0

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Fault0

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Fault0

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Fault0

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Fault0

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Fault0

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Fault0

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Fault0

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Fault0

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Fault0

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Fault0

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Fault0

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Fault0

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Fault0

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Fault0

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault0

FlexPWM1_FAULT1 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Fault1

enumerator kINPUTMUX_PinInt2ToFlexPwm1Fault1

enumerator kINPUTMUX_SctOut4ToFlexPwm1Fault1

enumerator kINPUTMUX_SctOut5ToFlexPwm1Fault1

enumerator kINPUTMUX_SctOut3ToFlexPwm1Fault1

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Fault1

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Fault1

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Fault1

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Fault1

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Fault1

enumerator kINPUTMUX_ArmTxevToFlexPwm1Fault1

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Fault1

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Fault1

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Fault1

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Fault1

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Fault1

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Fault1

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Fault1

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Fault1

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Fault1

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Fault1

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Fault1

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Fault1

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Fault1

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Fault1

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Fault1

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Fault1

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Fault1

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Fault1

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Fault1

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Fault1

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Fault1

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Fault1

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Fault1

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Fault1

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Fault1

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault1

FlexPWM1_FAULT2 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Fault2

enumerator kINPUTMUX_PinInt2ToFlexPwm1Fault2

enumerator kINPUTMUX_SctOut4ToFlexPwm1Fault2

enumerator kINPUTMUX_SctOut5ToFlexPwm1Fault2

enumerator kINPUTMUX_SctOut3ToFlexPwm1Fault2

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Fault2

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Fault2

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Fault2

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Fault2

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Fault2

enumerator kINPUTMUX_ArmTxevToFlexPwm1Fault2

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Fault2

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Fault2

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Fault2

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Fault2

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Fault2

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Fault2

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Fault2

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Fault2

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Fault2

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Fault2

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Fault2

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Fault2

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Fault2

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Fault2

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Fault2

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Fault2

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Fault2

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Fault2

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Fault2

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Fault2

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Fault2

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Fault2

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Fault2

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Fault2

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Fault2

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault2

FlexPWM1_FAULT3 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Fault3

enumerator kINPUTMUX_PinInt2ToFlexPwm1Fault3

enumerator kINPUTMUX_SctOut4ToFlexPwm1Fault3

enumerator kINPUTMUX_SctOut5ToFlexPwm1Fault3

enumerator kINPUTMUX_SctOut3ToFlexPwm1Fault3

enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Fault3

enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Fault3

enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Fault3

enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Fault3

enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Fault3

enumerator kINPUTMUX_ArmTxevToFlexPwm1Fault3

enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Fault3

enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Fault3

enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Fault3

enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Fault3

enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Fault3

enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Fault3

enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Fault3

enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Fault3

enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Fault3

enumerator kINPUTMUX_Cmp0OutToFlexPwm1Fault3

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Fault3

enumerator kINPUTMUX_Cmp2OutToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Fault3

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Fault3

enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Fault3

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Fault3

enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn0ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn1ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn2ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn3ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn4ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn5ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn6ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn7ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn8ToFlexPwm1Fault3

enumerator kINPUTMUX_TrigIn9ToFlexPwm1Fault3

enumerator kINPUTMUX_SincFilterCh0ToFlexPwm1Fault3

enumerator kINPUTMUX_SincFilterCh1ToFlexPwm1Fault3

enumerator kINPUTMUX_SincFilterCh2ToFlexPwm1Fault3

enumerator kINPUTMUX_SincFilterCh3ToFlexPwm1Fault3

enumerator kINPUTMUX_SincFilterCh4ToFlexPwm1Fault3

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Fault3

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Fault3

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Fault3

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault3

PWM0 external clock trigger.

enumerator kINPUTMUX_Fro16KToPwm0ExtClk

enumerator kINPUTMUX_Osc32KToPwm0ExtClk

enumerator kINPUTMUX_EvtgOut0AToPwm0ExtClk

enumerator kINPUTMUX_EvtgOut1AToPwm0ExtClk

enumerator kINPUTMUX_ExttrigIn0ToPwm0ExtClk

enumerator kINPUTMUX_ExttrigIn1ToPwm0ExtClk
PWM1 external clock trigger.

enumerator kINPUTMUX_Fro16KToPwm1ExtClk

enumerator kINPUTMUX_Osc32KToPwm1ExtClk

enumerator kINPUTMUX_EvtgOut0AToPwm1ExtClk

enumerator kINPUTMUX_EvtgOut1AToPwm1ExtClk

enumerator kINPUTMUX_ExttrigIn0ToPwm1ExtClk

enumerator kINPUTMUX_ExttrigIn1ToPwm1ExtClk
EVTG trigger input connections.

enumerator kINPUTMUX_PinInt0ToEvtgTrigger

enumerator kINPUTMUX_PinInt1ToEvtgTrigger

enumerator kINPUTMUX_SctOut0ToEvtgTrigger

enumerator kINPUTMUX_SctOut1ToEvtgTrigger

enumerator kINPUTMUX_SctOut2ToEvtgTrigger

enumerator kINPUTMUX_SctOut3ToEvtgTrigger

enumerator kINPUTMUX_Ctimer0M3ToEvtgTrigger

enumerator kINPUTMUX_Ctimer1M3ToEvtgTrigger

enumerator kINPUTMUX_Ctimer2M3ToEvtgTrigger

enumerator kINPUTMUX_Ctimer2M2ToEvtgTrigger

enumerator kINPUTMUX_Ctimer3M2ToEvtgTrigger

enumerator kINPUTMUX_Ctimer4M2ToEvtgTrigger

enumerator kINPUTMUX_GpioIntBmatchToEvtgTrigger

enumerator kINPUTMUX_Adc0IrqToEvtgTrigger

enumerator kINPUTMUX_Adc1IrqToEvtgTrigger

enumerator kINPUTMUX_Adc0Tcomp0ToEvtgTrigger

enumerator kINPUTMUX_Adc0Tcomp1ToEvtgTrigger

enumerator kINPUTMUX_Adc0Tcomp2ToEvtgTrigger

enumerator kINPUTMUX_Adc0Tcomp3ToEvtgTrigger

enumerator kINPUTMUX_Adc1Tcomp0ToEvtgTrigger

enumerator kINPUTMUX_Adc1Tcomp1ToEvtgTrigger

enumerator kINPUTMUX_Adc1Tcomp2ToEvtgTrigger

enumerator kINPUTMUX_Adc1Tcomp3ToEvtgTrigger

enumerator kINPUTMUX_Cmp0OutToEvtgTrigger

enumerator kINPUTMUX_Cmp1OutToEvtgTrigger

enumerator kINPUTMUX_Cmp2OutToEvtgTrigger

enumerator kINPUTMUX_Pwm0A0Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm0A0Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm0A1Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm0A1Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm0A2Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm0A2Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm0A3Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm0A3Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A0Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A0Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A1Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A1Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A2Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A2Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A3Trig0ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A3Trig1ToEvtgTrigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToEvtgTrigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToEvtgTrigger

enumerator kINPUTMUX_TrigIn0ToEvtgTrigger

enumerator kINPUTMUX_TrigIn1ToEvtgTrigger

enumerator kINPUTMUX_TrigIn2ToEvtgTrigger

enumerator kINPUTMUX_TrigIn3ToEvtgTrigger

enumerator kINPUTMUX_Lptmr0ToEvtgTrigger

enumerator kINPUTMUX_Lptmr1ToEvtgTrigger

enumerator kINPUTMUX_SincFilterCh0ToEvtgTrigger

enumerator kINPUTMUX_SincFilterCh1ToEvtgTrigger

enumerator kINPUTMUX_SincFilterCh2ToEvtgTrigger

enumerator kINPUTMUX_SincFilterCh3ToEvtgTrigger

enumerator kINPUTMUX_SincFilterCh4ToEvtgTrigger

USB-FS trigger input connections.

enumerator kINPUTMUX_Lpflexcomm0Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm1Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm2Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm3Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm4Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm5Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm6Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm7Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm8Trig3ToUsbfsTrigger

enumerator kINPUTMUX_Lpflexcomm9Trig3ToUsbfsTrigger

TSI trigger input connections.

enumerator kINPUTMUX_Lptmr0ToTsiTrigger

enumerator kINPUTMUX_Lptmr1ToTsiTrigger

EXT trigger connections.

enumerator kINPUTMUX_PinInt0ToExtTrigger

enumerator kINPUTMUX_PinInt1ToExtTrigger

enumerator kINPUTMUX_Adc0IrqToExtTrigger

enumerator kINPUTMUX_Adc1IrqToExtTrigger

enumerator kINPUTMUX_Adc0Tcomp0ToExtTrigger

enumerator kINPUTMUX_Adc1Tcomp0ToExtTrigger

enumerator kINPUTMUX_Pwm0A0Trig01ToExtTrigger

enumerator kINPUTMUX_Pwm0A1Trig01ToExtTrigger

enumerator kINPUTMUX_Pwm0A2Trig01ToExtTrigger

enumerator kINPUTMUX_Pwm0A3Trig01ToExtTrigger

enumerator kINPUTMUX_Pwm1A0Trig01ToExtTrigger

enumerator kINPUTMUX_Pwm1A1Trig01ToExtTrigger

enumerator kINPUTMUX_Pwm1A2Trig01ToExtTrigger

enumerator kINPUTMUX_Pwm1A3Trig01ToExtTrigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToExtTrigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToExtTrigger

enumerator kINPUTMUX_EvtgOut0AToExtTrigger

enumerator kINPUTMUX_EvtgOut0BToExtTrigger

enumerator kINPUTMUX_EvtgOut1AToExtTrigger

enumerator kINPUTMUX_EvtgOut1BToExtTrigger

enumerator kINPUTMUX_EvtgOut2AToExtTrigger

enumerator kINPUTMUX_EvtgOut2BToExtTrigger

enumerator kINPUTMUX_EvtgOut3AToExtTrigger

enumerator kINPUTMUX_EvtgOut3BToExtTrigger

enumerator kINPUTMUX_Lptmr0ToExtTrigger

enumerator kINPUTMUX_Lptmr1ToExtTrigger

enumerator kINPUTMUX_SctOut0ToExtTrigger

enumerator kINPUTMUX_SctOut1ToExtTrigger

enumerator kINPUTMUX_SctOut2ToExtTrigger

enumerator kINPUTMUX_SctOut3ToExtTrigger

enumerator kINPUTMUX_SctOut4ToExtTrigger

enumerator kINPUTMUX_SctOut5ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm0Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm1Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm2Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm3Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm4Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm5Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm6Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm7Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm8Trig3ToExtTrigger

enumerator kINPUTMUX_Lpflexcomm9Trig3ToExtTrigger

enumerator kINPUTMUX_Cmp0OutToExtTrigger

enumerator kINPUTMUX_Cmp1OutToExtTrigger

enumerator kINPUTMUX_Cmp2OutToExtTrigger

enumerator kINPUTMUX_EnetPpsOut0ToExtTrigger

SINC Filter channel trigger input connections.

enumerator kINPUTMUX_PinInt0ToSincFilterChTrigger

enumerator kINPUTMUX_PinInt1ToSincFilterChTrigger

enumerator kINPUTMUX_SctOut4ToSincFilterChTrigger

enumerator kINPUTMUX_SctOut5ToSincFilterChTrigger

enumerator kINPUTMUX_SctOut9ToSincFilterChTrigger

enumerator kINPUTMUX_Ctimer0M3ToSincFilterChTrigger

enumerator kINPUTMUX_Ctimer1M3ToSincFilterChTrigger

enumerator kINPUTMUX_Ctimer2M3ToSincFilterChTrigger

enumerator kINPUTMUX_Ctimer3M3ToSincFilterChTrigger

enumerator kINPUTMUX_Ctimer4M3ToSincFilterChTrigger

enumerator kINPUTMUX_ArmTxevToSincFilterChTrigger

enumerator kINPUTMUX_GpioIntBmatchToSincFilterChTrigger

enumerator kINPUTMUX_Adc0Tcomp0ToSincFilterChTrigger

enumerator kINPUTMUX_Adc0Tcomp1ToSincFilterChTrigger

enumerator kINPUTMUX_Adc0Tcomp2ToSincFilterChTrigger

enumerator kINPUTMUX_Adc0Tcomp3ToSincFilterChTrigger

enumerator kINPUTMUX_Adc1Tcomp0ToSincFilterChTrigger

enumerator kINPUTMUX_Adc1Tcomp1ToSincFilterChTrigger

enumerator kINPUTMUX_Adc1Tcomp2ToSincFilterChTrigger

enumerator kINPUTMUX_Adc1Tcomp3ToSincFilterChTrigger

enumerator kINPUTMUX_Cmp0OutToSincFilterChTrigger

enumerator kINPUTMUX_Cmp1OutToSincFilterChTrigger

enumerator kINPUTMUX_Cmp2OutToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A0Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A0Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A1Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A1Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A2Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A2Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A3Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm0A3Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A0Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A0Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A1Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A1Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A2Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A2Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A3Trig0ToSincFilterChTrigger

enumerator kINPUTMUX_Pwm1A3Trig1ToSincFilterChTrigger

enumerator kINPUTMUX_Qdc0CmpPosMatchToSincFilterChTrigger

enumerator kINPUTMUX_Qdc1CmpPosMatchToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut0AToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut0BToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut1AToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut1BToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut2AToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut2BToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut3AToSincFilterChTrigger

enumerator kINPUTMUX_EvtgOut3BToSincFilterChTrigger

enumerator kINPUTMUX_Lptmr0ToSincFilterChTrigger

enumerator kINPUTMUX_Lptmr1ToSincFilterChTrigger

enumerator kINPUTMUX_FlexioCh0ToSincFilterChTrigger

enumerator kINPUTMUX_FlexioCh1ToSincFilterChTrigger

enumerator kINPUTMUX_FlexioCh2ToSincFilterChTrigger

enumerator kINPUTMUX_FlexioCh3ToSincFilterChTrigger

enumerator kINPUTMUX_WuuToSincFilterChTrigger

OPAMP0 trigger input connections.

enumerator kINPUTMUX_PinInt0ToOpamp0Trigger

enumerator kINPUTMUX_PinInt1ToOpamp0Trigger

enumerator kINPUTMUX_PinInt2ToOpamp0Trigger

enumerator kINPUTMUX_PinInt3ToOpamp0Trigger

enumerator kINPUTMUX_SctOut4ToOpamp0Trigger

enumerator kINPUTMUX_SctOut5ToOpamp0Trigger

enumerator kINPUTMUX_SctOut6ToOpamp0Trigger

enumerator kINPUTMUX_SctOut7ToOpamp0Trigger

enumerator kINPUTMUX_SctOut8ToOpamp0Trigger

enumerator kINPUTMUX_Ctimer0M3ToOpamp0Trigger

enumerator kINPUTMUX_Ctimer1M3ToOpamp0Trigger

enumerator kINPUTMUX_Ctimer2M3ToOpamp0Trigger

enumerator kINPUTMUX_Ctimer3M3ToOpamp0Trigger

enumerator kINPUTMUX_Ctimer4M3ToOpamp0Trigger

enumerator kINPUTMUX_GpioIntBmatchToOpamp0Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToOpamp0Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToOpamp0Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToOpamp0Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToOpamp0Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToOpamp0Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToOpamp0Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToOpamp0Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A0Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A0Trig1ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A1Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A1Trig1ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A2Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A2Trig1ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A3Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm0A3Trig1ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A0Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A0Trig1ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A1Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A1Trig1ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A2Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A3Trig0ToOpamp0Trigger

enumerator kINPUTMUX_Pwm1A3Trig1ToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut0AToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut0BToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut1AToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut1BToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut2AToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut2BToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut3AToOpamp0Trigger

enumerator kINPUTMUX_EvtgOut3BToOpamp0Trigger

enumerator kINPUTMUX_TrigIn0ToOpamp0Trigger

enumerator kINPUTMUX_TrigIn1ToOpamp0Trigger

enumerator kINPUTMUX_TrigIn2ToOpamp0Trigger

enumerator kINPUTMUX_TrigIn3ToOpamp0Trigger

enumerator kINPUTMUX_FlexioCh4ToOpamp0Trigger

enumerator kINPUTMUX_FlexioCh5ToOpamp0Trigger

enumerator kINPUTMUX_FlexioCh6ToOpamp0Trigger

enumerator kINPUTMUX_FlexioCh7ToOpamp0Trigger

OPAMP1 trigger input connections.

enumerator kINPUTMUX_PinInt0ToOpamp1Trigger

enumerator kINPUTMUX_PinInt1ToOpamp1Trigger

enumerator kINPUTMUX_PinInt2ToOpamp1Trigger

enumerator kINPUTMUX_PinInt3ToOpamp1Trigger

enumerator kINPUTMUX_SctOut4ToOpamp1Trigger

enumerator kINPUTMUX_SctOut5ToOpamp1Trigger

enumerator kINPUTMUX_SctOut6ToOpamp1Trigger

enumerator kINPUTMUX_SctOut7ToOpamp1Trigger

enumerator kINPUTMUX_SctOut8ToOpamp1Trigger

enumerator kINPUTMUX_Ctimer0M3ToOpamp1Trigger

enumerator kINPUTMUX_Ctimer1M3ToOpamp1Trigger

enumerator kINPUTMUX_Ctimer2M3ToOpamp1Trigger

enumerator kINPUTMUX_Ctimer3M3ToOpamp1Trigger

enumerator kINPUTMUX_Ctimer4M3ToOpamp1Trigger

enumerator kINPUTMUX_GpioIntBmatchToOpamp1Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToOpamp1Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToOpamp1Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToOpamp1Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToOpamp1Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToOpamp1Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToOpamp1Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToOpamp1Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A0Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A0Trig1ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A1Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A1Trig1ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A2Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A2Trig1ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A3Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm0A3Trig1ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A0Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A0Trig1ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A1Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A1Trig1ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A2Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A3Trig0ToOpamp1Trigger

enumerator kINPUTMUX_Pwm1A3Trig1ToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut0AToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut0BToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut1AToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut1BToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut2AToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut2BToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut3AToOpamp1Trigger

enumerator kINPUTMUX_EvtgOut3BToOpamp1Trigger

enumerator kINPUTMUX_TrigIn0ToOpamp1Trigger

enumerator kINPUTMUX_TrigIn1ToOpamp1Trigger

enumerator kINPUTMUX_TrigIn2ToOpamp1Trigger

enumerator kINPUTMUX_TrigIn3ToOpamp1Trigger

enumerator kINPUTMUX_FlexioCh4ToOpamp1Trigger

enumerator kINPUTMUX_FlexioCh5ToOpamp1Trigger

enumerator kINPUTMUX_FlexioCh6ToOpamp1Trigger

enumerator kINPUTMUX_FlexioCh7ToOpamp1Trigger

OPAMP2 trigger input connections.

enumerator kINPUTMUX_PinInt0ToOpamp2Trigger

enumerator kINPUTMUX_PinInt1ToOpamp2Trigger

enumerator kINPUTMUX_PinInt2ToOpamp2Trigger

enumerator kINPUTMUX_PinInt3ToOpamp2Trigger

enumerator kINPUTMUX_SctOut4ToOpamp2Trigger

enumerator kINPUTMUX_SctOut5ToOpamp2Trigger

enumerator kINPUTMUX_SctOut6ToOpamp2Trigger

enumerator kINPUTMUX_SctOut7ToOpamp2Trigger

enumerator kINPUTMUX_SctOut8ToOpamp2Trigger

enumerator kINPUTMUX_Ctimer0M3ToOpamp2Trigger

enumerator kINPUTMUX_Ctimer1M3ToOpamp2Trigger

enumerator kINPUTMUX_Ctimer2M3ToOpamp2Trigger

enumerator kINPUTMUX_Ctimer3M3ToOpamp2Trigger

enumerator kINPUTMUX_Ctimer4M3ToOpamp2Trigger

enumerator kINPUTMUX_GpioIntBmatchToOpamp2Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToOpamp2Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToOpamp2Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToOpamp2Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToOpamp2Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToOpamp2Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToOpamp2Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToOpamp2Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A0Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A0Trig1ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A1Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A1Trig1ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A2Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A2Trig1ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A3Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm0A3Trig1ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A0Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A0Trig1ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A1Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A1Trig1ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A2Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A3Trig0ToOpamp2Trigger

enumerator kINPUTMUX_Pwm1A3Trig1ToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut0AToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut0BToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut1AToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut1BToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut2AToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut2BToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut3AToOpamp2Trigger

enumerator kINPUTMUX_EvtgOut3BToOpamp2Trigger

enumerator kINPUTMUX_TrigIn0ToOpamp2Trigger

enumerator kINPUTMUX_TrigIn1ToOpamp2Trigger

enumerator kINPUTMUX_TrigIn2ToOpamp2Trigger

enumerator kINPUTMUX_TrigIn3ToOpamp2Trigger

enumerator kINPUTMUX_FlexioCh4ToOpamp2Trigger

enumerator kINPUTMUX_FlexioCh5ToOpamp2Trigger

enumerator kINPUTMUX_FlexioCh6ToOpamp2Trigger

enumerator kINPUTMUX_FlexioCh7ToOpamp2Trigger

FLEXCOMM0 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm0Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm0Trigger

enumerator kINPUTMUX_PinInt6ToFlexcomm0Trigger

enumerator kINPUTMUX_SctOut5ToFlexcomm0Trigger

enumerator kINPUTMUX_SctOut6ToFlexcomm0Trigger

enumerator kINPUTMUX_SctOut7ToFlexcomm0Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm0Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm0Trigger

enumerator kINPUTMUX_Ctimer2M0ToFlexcomm0Trigger

enumerator kINPUTMUX_Ctimer3M0ToFlexcomm0Trigger

enumerator kINPUTMUX_Ctimer4M0ToFlexcomm0Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm0Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm0Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm0Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm0Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm0Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm0Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm0Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm0Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm0Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm0Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm0Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm0Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm0Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm0Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm0Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm0Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm0Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm0Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm0Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm0Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm0Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm0Trigger

enumerator kINPUTMUX_WuuToFlexcomm0Trigger

FLEXCOMM1 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm1Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm1Trigger

enumerator kINPUTMUX_PinInt6ToFlexcomm1Trigger

enumerator kINPUTMUX_SctOut5ToFlexcomm1Trigger

enumerator kINPUTMUX_SctOut6ToFlexcomm1Trigger

enumerator kINPUTMUX_SctOut7ToFlexcomm1Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm1Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm1Trigger

enumerator kINPUTMUX_Ctimer2M0ToFlexcomm1Trigger

enumerator kINPUTMUX_Ctimer3M0ToFlexcomm1Trigger

enumerator kINPUTMUX_Ctimer4M0ToFlexcomm1Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm1Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm1Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm1Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm1Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm1Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm1Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm1Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm1Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm1Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm1Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm1Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm1Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm1Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm1Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm1Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm1Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm1Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm1Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm1Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm1Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm1Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm1Trigger

enumerator kINPUTMUX_WuuToFlexcomm1Trigger

FLEXCOMM2 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm2Trigger

enumerator kINPUTMUX_PinInt6ToFlexcomm2Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm2Trigger

enumerator kINPUTMUX_SctOut5ToFlexcomm2Trigger

enumerator kINPUTMUX_SctOut8ToFlexcomm2Trigger

enumerator kINPUTMUX_SctOut9ToFlexcomm2Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm2Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm2Trigger

enumerator kINPUTMUX_Ctimer2M1ToFlexcomm2Trigger

enumerator kINPUTMUX_Ctimer3M1ToFlexcomm2Trigger

enumerator kINPUTMUX_Ctimer4M1ToFlexcomm2Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm2Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm2Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm2Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm2Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm2Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm2Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm2Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm2Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm2Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm2Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm2Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm2Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm2Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm2Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm2Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm2Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm2Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm2Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm2Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm2Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm2Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm2Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm2Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm2Trigger

enumerator kINPUTMUX_WuuToFlexcomm2Trigger

FLEXCOMM3 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm3Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm3Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm3Trigger

enumerator kINPUTMUX_SctOut5ToFlexcomm3Trigger

enumerator kINPUTMUX_SctOut8ToFlexcomm3Trigger

enumerator kINPUTMUX_SctOut9ToFlexcomm3Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm3Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm3Trigger

enumerator kINPUTMUX_Ctimer2M1ToFlexcomm3Trigger

enumerator kINPUTMUX_Ctimer3M1ToFlexcomm3Trigger

enumerator kINPUTMUX_Ctimer4M1ToFlexcomm3Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm3Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm3Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm3Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm3Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm3Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm3Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm3Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm3Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm3Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm3Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm3Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm3Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm3Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm3Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm3Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm3Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm3Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm3Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm3Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm3Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm3Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm3Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm3Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm3Trigger

enumerator kINPUTMUX_WuuToFlexcomm3Trigger

FLEXCOMM4 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm4Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm4Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm4Trigger

enumerator kINPUTMUX_SctOut0ToFlexcomm4Trigger

enumerator kINPUTMUX_SctOut1ToFlexcomm4Trigger

enumerator kINPUTMUX_SctOut2ToFlexcomm4Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm4Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm4Trigger

enumerator kINPUTMUX_Ctimer2M2ToFlexcomm4Trigger

enumerator kINPUTMUX_Ctimer3M2ToFlexcomm4Trigger

enumerator kINPUTMUX_Ctimer4M2ToFlexcomm4Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm4Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm4Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm4Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm4Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm4Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm4Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm4Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm4Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm4Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm4Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm4Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm4Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm4Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm4Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm4Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm4Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm4Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm4Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm4Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm4Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm4Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm4Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm4Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm4Trigger

enumerator kINPUTMUX_WuuToFlexcomm4Trigger

FLEXCOMM5 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm5Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm5Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm5Trigger

enumerator kINPUTMUX_SctOut0ToFlexcomm5Trigger

enumerator kINPUTMUX_SctOut1ToFlexcomm5Trigger

enumerator kINPUTMUX_SctOut2ToFlexcomm5Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm5Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm5Trigger

enumerator kINPUTMUX_Ctimer2M2ToFlexcomm5Trigger

enumerator kINPUTMUX_Ctimer3M2ToFlexcomm5Trigger

enumerator kINPUTMUX_Ctimer4M2ToFlexcomm5Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm5Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm5Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm5Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm5Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm5Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm5Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm5Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm5Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm5Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm5Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm5Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm5Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm5Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm5Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm5Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm5Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm5Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm5Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm5Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm5Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm5Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm5Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm5Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm5Trigger

enumerator kINPUTMUX_WuuToFlexcomm5Trigger

FLEXCOMM6 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm6Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm6Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm6Trigger

enumerator kINPUTMUX_SctOut0ToFlexcomm6Trigger

enumerator kINPUTMUX_SctOut3ToFlexcomm6Trigger

enumerator kINPUTMUX_SctOut4ToFlexcomm6Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm6Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm6Trigger

enumerator kINPUTMUX_Ctimer2M3ToFlexcomm6Trigger

enumerator kINPUTMUX_Ctimer3M3ToFlexcomm6Trigger

enumerator kINPUTMUX_Ctimer4M3ToFlexcomm6Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm6Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm6Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm6Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm6Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm6Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm6Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm6Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm6Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm6Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm6Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm6Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm6Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm6Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm6Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm6Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm6Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm6Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm6Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm6Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm6Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm6Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm6Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm6Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm6Trigger

enumerator kINPUTMUX_WuuToFlexcomm6Trigger

FLEXCOMM7 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm7Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm7Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm7Trigger

enumerator kINPUTMUX_SctOut0ToFlexcomm7Trigger

enumerator kINPUTMUX_SctOut3ToFlexcomm7Trigger

enumerator kINPUTMUX_SctOut4ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer2M3ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer3M3ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer4M3ToFlexcomm7Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm7Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm7Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm7Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm7Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm7Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm7Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm7Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm7Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm7Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm7Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm7Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm7Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm7Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm7Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm7Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm7Trigger

enumerator kINPUTMUX_WuuToFlexcomm7Trigger

FLEXCOMM8 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm8Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm8Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm8Trigger

enumerator kINPUTMUX_SctOut0ToFlexcomm8Trigger

enumerator kINPUTMUX_SctOut3ToFlexcomm8Trigger

enumerator kINPUTMUX_SctOut4ToFlexcomm8Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm8Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm8Trigger

enumerator kINPUTMUX_Ctimer2M3ToFlexcomm8Trigger

enumerator kINPUTMUX_Ctimer3M3ToFlexcomm8Trigger

enumerator kINPUTMUX_Ctimer4M3ToFlexcomm8Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm8Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm8Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm8Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm8Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm8Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm8Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm8Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm8Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm8Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm8Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm8Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm8Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm8Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm8Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm8Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm8Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm8Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm8Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm8Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm8Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm8Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm8Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm8Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm8Trigger

enumerator kINPUTMUX_WuuToFlexcomm8Trigger

FLEXCOMM9 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm9Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm9Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm9Trigger

enumerator kINPUTMUX_SctOut0ToFlexcomm9Trigger

enumerator kINPUTMUX_SctOut3ToFlexcomm9Trigger

enumerator kINPUTMUX_SctOut4ToFlexcomm9Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm9Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm9Trigger

enumerator kINPUTMUX_Ctimer2M0ToFlexcomm9Trigger

enumerator kINPUTMUX_Ctimer3M0ToFlexcomm9Trigger

enumerator kINPUTMUX_Ctimer4M0ToFlexcomm9Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm9Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm9Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm9Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm9Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm9Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm9Trigger

enumerator kINPUTMUX_Cmp2OutToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm9Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm9Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm9Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm9Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm9Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm9Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm9Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm9Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm9Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm9Trigger

enumerator kINPUTMUX_FlexioCh5ToFlexcomm9Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm9Trigger

enumerator kINPUTMUX_FlexioCh7ToFlexcomm9Trigger

enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm9Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm9Trigger

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm9Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm9Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm9Trigger

enumerator kINPUTMUX_WuuToFlexcomm9Trigger

FlexIO trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexioTrigger

enumerator kINPUTMUX_PinInt5ToFlexioTrigger

enumerator kINPUTMUX_PinInt6ToFlexioTrigger

enumerator kINPUTMUX_PinInt7ToFlexioTrigger

enumerator kINPUTMUX_SctOut5ToFlexioTrigger

enumerator kINPUTMUX_SctOut6ToFlexioTrigger

enumerator kINPUTMUX_SctOut7ToFlexioTrigger

enumerator kINPUTMUX_SctOut8ToFlexioTrigger

enumerator kINPUTMUX_SctOut9ToFlexioTrigger

enumerator kINPUTMUX_Ctimer0M1ToFlexioTrigger

enumerator kINPUTMUX_Ctimer1M1ToFlexioTrigger

enumerator kINPUTMUX_Ctimer2M1ToFlexioTrigger

enumerator kINPUTMUX_Ctimer3M1ToFlexioTrigger

enumerator kINPUTMUX_Ctimer4M1ToFlexioTrigger

enumerator kINPUTMUX_Lptmr0ToFlexioTrigger

enumerator kINPUTMUX_Lptmr1ToFlexioTrigger

enumerator kINPUTMUX_ArmTxevToFlexioTrigger

enumerator kINPUTMUX_GpioIntBmatchToFlexioTrigger

enumerator kINPUTMUX_Adc0Tcomp0ToFlexioTrigger

enumerator kINPUTMUX_Adc0Tcomp1ToFlexioTrigger

enumerator kINPUTMUX_Adc0Tcomp2ToFlexioTrigger

enumerator kINPUTMUX_Adc0Tcomp3ToFlexioTrigger

enumerator kINPUTMUX_Adc1Tcomp0ToFlexioTrigger

enumerator kINPUTMUX_Adc1Tcomp1ToFlexioTrigger

enumerator kINPUTMUX_Adc1Tcomp2ToFlexioTrigger

enumerator kINPUTMUX_Adc1Tcomp3ToFlexioTrigger

enumerator kINPUTMUX_Cmp0OutToFlexioTrigger

enumerator kINPUTMUX_Cmp1OutToFlexioTrigger

enumerator kINPUTMUX_Cmp2OutToFlexioTrigger

enumerator kINPUTMUX_Pwm0A0Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm0A0Trig1ToFlexioTrigger

enumerator kINPUTMUX_Pwm0A1Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm0A1Trig1ToFlexioTrigger

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm0A2Trig1ToFlexioTrigger

enumerator kINPUTMUX_Pwm0A3Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm0A3Trig1ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A1Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A1Trig1ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A2Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A3Trig0ToFlexioTrigger

enumerator kINPUTMUX_Pwm1A3Trig1ToFlexioTrigger

enumerator kINPUTMUX_EvtgOut0AToFlexioTrigger

enumerator kINPUTMUX_EvtgOut0BToFlexioTrigger

enumerator kINPUTMUX_EvtgOut1AToFlexioTrigger

enumerator kINPUTMUX_EvtgOut1BToFlexioTrigger

enumerator kINPUTMUX_EvtgOut2AToFlexioTrigger

enumerator kINPUTMUX_EvtgOut2BToFlexioTrigger

enumerator kINPUTMUX_EvtgOut3AToFlexioTrigger

enumerator kINPUTMUX_EvtgOut3BToFlexioTrigger

enumerator kINPUTMUX_TrigIn0ToFlexioTrigger

enumerator kINPUTMUX_TrigIn1ToFlexioTrigger

enumerator kINPUTMUX_TrigIn2ToFlexioTrigger

enumerator kINPUTMUX_TrigIn3ToFlexioTrigger

enumerator kINPUTMUX_TrigIn4ToFlexioTrigger

enumerator kINPUTMUX_SincFilterCh0ToFlexioTrigger

enumerator kINPUTMUX_SincFilterCh1ToFlexioTrigger

enumerator kINPUTMUX_SincFilterCh2ToFlexioTrigger

enumerator kINPUTMUX_SincFilterCh3ToFlexioTrigger

enumerator kINPUTMUX_SincFilterCh4ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm0Trig0ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm0Trig1ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm0Trig2ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm1Trig0ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm1Trig1ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm1Trig2ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm2Trig0ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm2Trig1ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm2Trig2ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm3Trig0ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm3Trig1ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm3Trig2ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm3Trig3ToFlexioTrigger

enumerator kINPUTMUX_WuuToFlexioTrigger

enum _inputmux_signal_t

INPUTMUX signal enable/disable type.

Values:

enumerator kINPUTMUX_FlexSpi0RxToDma0Ch1Ena

DMA0 REQ ENABLE0 signal.

enumerator kINPUTMUX_FlexSpi0TxToDma0Ch2Ena

enumerator kINPUTMUX_PinInt0ToDma0Ch3Ena

enumerator kINPUTMUX_PinInt1ToDma0Ch4Ena

enumerator kINPUTMUX_PinInt2ToDma0Ch5Ena

enumerator kINPUTMUX_PinInt3ToDma0Ch6Ena

enumerator kINPUTMUX_Ctimer0M0ToDma0Ch7Ena

enumerator kINPUTMUX_Ctimer0M1ToDma0Ch8Ena

enumerator kINPUTMUX_Ctimer1M0ToDma0Ch9Ena

enumerator kINPUTMUX_Ctimer1M1ToDma0Ch10Ena

enumerator kINPUTMUX_Ctimer2M0ToDma0Ch11Ena

enumerator kINPUTMUX_Ctimer2M1ToDma0Ch12Ena

enumerator kINPUTMUX_Ctimer3M0ToDma0Ch13Ena

enumerator kINPUTMUX_Ctimer3M1ToDma0Ch14Ena

enumerator kINPUTMUX_Ctimer4M0ToDma0Ch15Ena

enumerator kINPUTMUX_Ctimer4M1ToDma0Ch16Ena

enumerator kINPUTMUX_Wuu0ToDma0Ch17Ena

enumerator kINPUTMUX_Micfil0FifoRequestToDma0Ch18Ena

enumerator kINPUTMUX_Sct0Dma0ToDma0Ch19Ena

enumerator kINPUTMUX_Sct0Dma1ToDma0Ch20Ena

enumerator kINPUTMUX_Adc0FifoARequestToDma0Ch21Ena

enumerator kINPUTMUX_Adc0FifoBRequestToDma0Ch22Ena

enumerator kINPUTMUX_Adc1FifoARequestToDma0Ch23Ena

enumerator kINPUTMUX_Adc1FifoBRequestoDma0Ch24Ena

enumerator kINPUTMUX_Dac0FifoRequestToDma0Ch25Ena

enumerator kINPUTMUX_Dac1FifoRequestToDma0Ch26Ena

enumerator kINPUTMUX_HpDac0FifoRequestToDma0Ch27Ena

enumerator kINPUTMUX_HsCmp0DmaRequestToDma0Ch28Ena

enumerator kINPUTMUX_HsCmp1DmaRequestToDma0Ch29Ena

enumerator kINPUTMUX_HsCmp2DmaRequestToDma0Ch30Ena

enumerator kINPUTMUX_Evtg0Out0AToDma0Ch31Ena

DMA0 REQ ENABLE1 signal.

enumerator kINPUTMUX_Evtg0Out0BToDma0Ch32Ena

enumerator kINPUTMUX_Evtg0Out1AToDma0Ch33Ena

enumerator kINPUTMUX_Evtg0Out1BToDma0Ch34Ena

enumerator kINPUTMUX_Evtg0Out2AToDma0Ch35Ena

enumerator kINPUTMUX_Evtg0Out2BToDma0Ch36Ena

enumerator kINPUTMUX_Evtg0Out3AToDma0Ch37Ena

enumerator kINPUTMUX_Evtg0Out3BToDma0Ch38Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt0ToDma0Ch39Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt1ToDma0Ch40Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt2ToDma0Ch41Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt3ToDma0Ch42Ena

enumerator kINPUTMUX_FlexPwm0ReqVal0ToDma0Ch43Ena

enumerator kINPUTMUX_FlexPwm0ReqVal1ToDma0Ch44Ena

enumerator kINPUTMUX_FlexPwm0ReqVal2ToDma0Ch45Ena

enumerator kINPUTMUX_FlexPwm0ReqVal3ToDma0Ch46Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt0ToDma0Ch47Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt1ToDma0Ch48Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt2ToDma0Ch49Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt3ToDma0Ch50Ena

enumerator kINPUTMUX_FlexPwm1ReqVal0ToDma0Ch51Ena

enumerator kINPUTMUX_FlexPwm1ReqVal1ToDma0Ch52Ena

enumerator kINPUTMUX_FlexPwm1ReqVal2ToDma0Ch53Ena

enumerator kINPUTMUX_FlexPwm1ReqVal3ToDma0Ch54Ena

enumerator kINPUTMUX_Itrc0TmprOut0ToDma0Ch55Ena

enumerator kINPUTMUX_Itrc0TmprOut1ToDma0Ch56Ena

enumerator kINPUTMUX_Lptmr0ToDma0Ch57Ena

enumerator kINPUTMUX_Lptmr1ToDma0Ch58Ena

enumerator kINPUTMUX_FlexCan0DmaRequestToDma0Ch59Ena

enumerator kINPUTMUX_FlexCan1DmaRequestToDma0Ch60Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister0RequestToDma0Ch61Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister1RequestToDma0Ch62Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister2RequestToDma0Ch63Ena
DMA0 REQ ENABLE2 signal.

enumerator kINPUTMUX_FlexIO0ShiftRegister3RequestToDma0Ch64Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister4RequestToDma0Ch65Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister5RequestToDma0Ch66Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister6RequestToDma0Ch67Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister7RequestToDma0Ch68Ena

enumerator kINPUTMUX_LpFlexcomm0RxToDma0Ch69Ena

enumerator kINPUTMUX_LpFlexcomm0TxToDma0Ch70Ena

enumerator kINPUTMUX_LpFlexcomm1RxToDma0Ch71Ena

enumerator kINPUTMUX_LpFlexcomm1TxToDma0Ch72Ena

enumerator kINPUTMUX_LpFlexcomm2RxToDma0Ch73Ena

enumerator kINPUTMUX_LpFlexcomm2TxToDma0Ch74Ena

enumerator kINPUTMUX_LpFlexcomm3RxToDma0Ch75Ena

enumerator kINPUTMUX_LpFlexcomm3TxToDma0Ch76Ena

enumerator kINPUTMUX_LpFlexcomm4RxToDma0Ch77Ena

enumerator kINPUTMUX_LpFlexcomm4TxToDma0Ch78Ena

enumerator kINPUTMUX_LpFlexcomm5RxToDma0Ch79Ena

enumerator kINPUTMUX_LpFlexcomm5TxToDma0Ch80Ena

enumerator kINPUTMUX_LpFlexcomm6RxToDma0Ch81Ena

enumerator kINPUTMUX_LpFlexcomm6TxToDma0Ch82Ena

enumerator kINPUTMUX_LpFlexcomm7RxToDma0Ch83Ena

enumerator kINPUTMUX_LpFlexcomm7TxToDma0Ch84Ena

enumerator kINPUTMUX_LpFlexcomm8RxToDma0Ch85Ena

enumerator kINPUTMUX_LpFlexcomm8TxToDma0Ch86Ena

enumerator kINPUTMUX_LpFlexcomm9RxToDma0Ch87Ena

enumerator kINPUTMUX_LpFlexcomm9TxToDma0Ch88Ena

enumerator kINPUTMUX_ESpi0Ch0ToDma0Ch89Ena

enumerator kINPUTMUX_ESpi0Ch1ToDma0Ch90Ena

enumerator kINPUTMUX_EmvSim0RxToDma0Ch91Ena

enumerator kINPUTMUX_EmvSim0TxToDma0Ch92Ena

enumerator kINPUTMUX_EmvSim1RxToDma0Ch93Ena

enumerator kINPUTMUX_EmvSim1TxToDma0Ch94Ena

enumerator kINPUTMUX_I3c0RxToDma0Ch95Ena

DMA0 REQ ENABLE3 signal.

enumerator kINPUTMUX_I3c0TxToDma0Ch96Ena

enumerator kINPUTMUX_I3c1RxToDma0Ch97Ena

enumerator kINPUTMUX_I3c1TxToDma0Ch98Ena

enumerator kINPUTMUX_Sai0RxToDma0Ch99Ena

enumerator kINPUTMUX_Sai0TxToDma0Ch100Ena

enumerator kINPUTMUX_Sai1RxToDma0Ch101Ena

enumerator kINPUTMUX_Sai1TxToDma0Ch102Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt0ToDma0Ch103Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt1ToDma0Ch104Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt2ToDma0Ch105Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt3ToDma0Ch106Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt4ToDma0Ch107Ena

enumerator kINPUTMUX_Gpio0PinEventRequest0ToDma0Ch108Ena

enumerator kINPUTMUX_Gpio0PinEventRequest1ToDma0Ch109Ena

enumerator kINPUTMUX_Gpio1PinEventRequest0ToDma0Ch110Ena

enumerator kINPUTMUX_Gpio1PinEventRequest1ToDma0Ch111Ena

enumerator kINPUTMUX_Gpio2PinEventRequest0ToDma0Ch112Ena

enumerator kINPUTMUX_Gpio2PinEventRequest1ToDma0Ch113Ena

enumerator kINPUTMUX_Gpio3PinEventRequest0ToDma0Ch114Ena

enumerator kINPUTMUX_Gpio3PinEventRequest1ToDma0Ch115Ena

enumerator kINPUTMUX_Gpio4PinEventRequest0ToDma0Ch116Ena

enumerator kINPUTMUX_Gpio4PinEventRequest1ToDma0Ch117Ena

enumerator kINPUTMUX_Gpio5PinEventRequest0ToDma0Ch118Ena

enumerator kINPUTMUX_Gpio5PinEventRequest1ToDma0Ch119Ena

enumerator kINPUTMUX_Tsi0EndOfScanToDma0Ch120Ena

enumerator kINPUTMUX_Tsi0OutOfRangeToDma0Ch121Ena

DMA1 REQ ENABLE0 signal.

enumerator kINPUTMUX_FlexSpi0RxToDma1Ch1Ena

enumerator kINPUTMUX_FlexSpi0TxToDma1Ch2Ena

enumerator kINPUTMUX_PinInt0ToDma1Ch3Ena

enumerator kINPUTMUX_PinInt1ToDma1Ch4Ena

enumerator kINPUTMUX_PinInt2ToDma1Ch5Ena

enumerator kINPUTMUX_PinInt3ToDma1Ch6Ena

enumerator kINPUTMUX_Ctimer0M0ToDma1Ch7Ena

enumerator kINPUTMUX_Ctimer0M1ToDma1Ch8Ena

enumerator kINPUTMUX_Ctimer1M0ToDma1Ch9Ena

enumerator kINPUTMUX_Ctimer1M1ToDma1Ch10Ena

enumerator kINPUTMUX_Ctimer2M0ToDma1Ch11Ena

enumerator kINPUTMUX_Ctimer2M1ToDma1Ch12Ena

enumerator kINPUTMUX_Ctimer3M0ToDma1Ch13Ena

enumerator kINPUTMUX_Ctimer3M1ToDma1Ch14Ena

enumerator kINPUTMUX_Ctimer4M0ToDma1Ch15Ena

enumerator kINPUTMUX_Ctimer4M1ToDma1Ch16Ena

enumerator kINPUTMUX_Wuu0ToDma1Ch17Ena

enumerator kINPUTMUX_Micfil0FifoRequestToDma1Ch18Ena

enumerator kINPUTMUX_Sct0Dma0ToDma1Ch19Ena

enumerator kINPUTMUX_Sct0Dma1ToDma1Ch20Ena

enumerator kINPUTMUX_Adc0FifoARequestToDma1Ch21Ena

enumerator kINPUTMUX_Adc0FifoBRequestToDma1Ch22Ena

enumerator kINPUTMUX_Adc1FifoARequestToDma1Ch23Ena

enumerator kINPUTMUX_Adc1FifoBRequestToDma1Ch24Ena

enumerator kINPUTMUX_Dac0FifoRequestToDma1Ch25Ena

enumerator kINPUTMUX_Dac1FifoRequestToDma1Ch26Ena

enumerator kINPUTMUX_HpDac0FifoRequestToDma1Ch27Ena

enumerator kINPUTMUX_HsCmp0DmaRequestToDma1Ch28Ena

enumerator kINPUTMUX_HsCmp1DmaRequestToDma1Ch29Ena

enumerator kINPUTMUX_HsCmp2DmaRequestToDma1Ch30Ena

enumerator kINPUTMUX_Evtg0Out0AToDma1Ch31Ena
DMA1 REQ ENABLE1 signal.

enumerator kINPUTMUX_Evtg0Out0BToDma1Ch32Ena

enumerator kINPUTMUX_Evtg0Out1AToDma1Ch33Ena

enumerator kINPUTMUX_Evtg0Out1BToDma1Ch34Ena

enumerator kINPUTMUX_Evtg0Out2AToDma1Ch35Ena

enumerator kINPUTMUX_Evtg0Out2BToDma1Ch36Ena

enumerator kINPUTMUX_Evtg0Out3AToDma1Ch37Ena

enumerator kINPUTMUX_Evtg0Out3BToDma1Ch38Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt0ToDma1Ch39Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt1ToDma1Ch40Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt2ToDma1Ch41Ena

enumerator kINPUTMUX_FlexPwm0ReqCapt3ToDma1Ch42Ena

enumerator kINPUTMUX_FlexPwm0ReqVal0ToDma1Ch43Ena

enumerator kINPUTMUX_FlexPwm0ReqVal1ToDma1Ch44Ena

enumerator kINPUTMUX_FlexPwm0ReqVal2ToDma1Ch45Ena

enumerator kINPUTMUX_FlexPwm0ReqVal3ToDma1Ch46Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt0ToDma1Ch47Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt1ToDma1Ch48Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt2ToDma1Ch49Ena

enumerator kINPUTMUX_FlexPwm1ReqCapt3ToDma1Ch50Ena

enumerator kINPUTMUX_FlexPwm1ReqVal0ToDma1Ch51Ena

enumerator kINPUTMUX_FlexPwm1ReqVal1ToDma1Ch52Ena

enumerator kINPUTMUX_FlexPwm1ReqVal2ToDma1Ch53Ena

enumerator kINPUTMUX_FlexPwm1ReqVal3ToDma1Ch54Ena

enumerator kINPUTMUX_Itrc0TmprOut0ToDma1Ch55Ena

enumerator kINPUTMUX_Itrc0TmprOut1ToDma1Ch56Ena

enumerator kINPUTMUX_Lptmr0ToDma1Ch57Ena

enumerator kINPUTMUX_Lptmr1ToDma1Ch58Ena

enumerator kINPUTMUX_FlexCan0DmaRequestToDma1Ch59Ena

enumerator kINPUTMUX_FlexCan1DmaRequestToDma1Ch60Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister0RequestToDma1Ch61Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister1RequestToDma1Ch62Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister2RequestToDma1Ch63Ena
DMA1 REQ ENABLE2 signal.

enumerator kINPUTMUX_FlexIO0ShiftRegister3RequestToDma1Ch64Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister4RequestToDma1Ch65Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister5RequestToDma1Ch66Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister6RequestToDma1Ch67Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister7RequestToDma1Ch68Ena

enumerator kINPUTMUX_LpFlexcomm0RxToDma1Ch69Ena

enumerator kINPUTMUX_LpFlexcomm0TxToDma1Ch70Ena

enumerator kINPUTMUX_LpFlexcomm1RxToDma1Ch71Ena

enumerator kINPUTMUX_LpFlexcomm1TxToDma1Ch72Ena

enumerator kINPUTMUX_LpFlexcomm2RxToDma1Ch73Ena

enumerator kINPUTMUX_LpFlexcomm2TxToDma1Ch74Ena

enumerator kINPUTMUX_LpFlexcomm3RxToDma1Ch75Ena

enumerator kINPUTMUX_LpFlexcomm3TxToDma1Ch76Ena

enumerator kINPUTMUX_LpFlexcomm4RxToDma1Ch77Ena

enumerator kINPUTMUX_LpFlexcomm4TxToDma1Ch78Ena

enumerator kINPUTMUX_LpFlexcomm5RxToDma1Ch79Ena

enumerator kINPUTMUX_LpFlexcomm5TxToDma1Ch80Ena

enumerator kINPUTMUX_LpFlexcomm6RxToDma1Ch81Ena

enumerator kINPUTMUX_LpFlexcomm6TxToDma1Ch82Ena

enumerator kINPUTMUX_LpFlexcomm7RxToDma1Ch83Ena

enumerator kINPUTMUX_LpFlexcomm7TxToDma1Ch84Ena

enumerator kINPUTMUX_LpFlexcomm8RxToDma1Ch85Ena

enumerator kINPUTMUX_LpFlexcomm8TxToDma1Ch86Ena

enumerator kINPUTMUX_LpFlexcomm9RxToDma1Ch87Ena

enumerator kINPUTMUX_LpFlexcomm9TxToDma1Ch88Ena

enumerator kINPUTMUX_ESpi0Ch0ToDma1Ch89Ena

enumerator kINPUTMUX_ESpi0Ch1ToDma1Ch90Ena

enumerator kINPUTMUX_EmvSim0RxToDma1Ch91Ena

enumerator kINPUTMUX_EmvSim0TxToDma1Ch92Ena

enumerator kINPUTMUX_EmvSim1RxToDma1Ch93Ena

enumerator kINPUTMUX_EmvSim1TxToDma1Ch94Ena

enumerator kINPUTMUX_I3c0RxToDma1Ch95Ena
DMA1 REQ ENABLE3 signal.

enumerator kINPUTMUX_I3c0TxToDma1Ch96Ena

enumerator kINPUTMUX_I3c1RxToDma1Ch97Ena

enumerator kINPUTMUX_I3c1TxToDma1Ch98Ena

enumerator kINPUTMUX_Sai0RxToDma1Ch99Ena

enumerator kINPUTMUX_Sai0TxToDma1Ch100Ena

enumerator kINPUTMUX_Sai1RxToDma1Ch101Ena

enumerator kINPUTMUX_Sai1TxToDma1Ch102Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt0ToDma1Ch103Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt1ToDma1Ch104Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt2ToDma1Ch105Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt3ToDma1Ch106Ena

enumerator kINPUTMUX_Sinc0IpdReqSincAlt4ToDma1Ch107Ena

enumerator kINPUTMUX_Gpio0PinEventRequest0ToDma1Ch108Ena

enumerator kINPUTMUX_Gpio0PinEventRequest1ToDma1Ch109Ena

enumerator kINPUTMUX_Gpio1PinEventRequest0ToDma1Ch110Ena

enumerator kINPUTMUX_Gpio1PinEventRequest1ToDma1Ch111Ena

enumerator kINPUTMUX_Gpio2PinEventRequest0ToDma1Ch112Ena

enumerator kINPUTMUX_Gpio2PinEventRequest1ToDma1Ch113Ena

enumerator kINPUTMUX_Gpio3PinEventRequest0ToDma1Ch114Ena

enumerator kINPUTMUX_Gpio3PinEventRequest1ToDma1Ch115Ena

enumerator kINPUTMUX_Gpio4PinEventRequest0ToDma1Ch116Ena

enumerator kINPUTMUX_Gpio4PinEventRequest1ToDma1Ch117Ena

enumerator kINPUTMUX_Gpio5PinEventRequest0ToDma1Ch118Ena

enumerator kINPUTMUX_Gpio5PinEventRequest1ToDma1Ch119Ena

enumerator kINPUTMUX_Tsi0EndOfScanToDma1Ch120Ena

enumerator kINPUTMUX_Tsi0OutOfRangeToDma1Ch121Ena

typedef enum *_inputmux_connection_t* inputmux_connection_t
INPUTMUX connections type.

typedef enum *_inputmux_signal_t* inputmux_signal_t
INPUTMUX signal enable/disable type.

SCT0_INMUX0
Periphinmux IDs.

TIMER0CAPTSEL0

TIMER0TRIGIN

TIMER1CAPTSEL0

TIMER1TRIGIN

TIMER2CAPTSEL0

TIMER2TRIGIN

SMARTDMAARCHB_INMUX0

PINTSEL0

FREQMEAS_REF_REG

FREQMEAS_TAR_REG

TIMER3CAPTSEL0

TIMER3TRIGIN

TIMER4CAPTSEL0

TIMER4TRIGIN

CMP0_TRIG_REG

ADC0_TRIG0

ADC1_TRIG0

DAC0_TRIG_REG

DAC1_TRIG_REG

DAC2_TRIG_REG

QDC0_TRIG_REG

QDC0_HOME_REG

QDC0_INDEX_REG

QDC0_PHASEB_REG

QDC0_PHASEA_REG

QDC1_TRIG_REG

QDC1_HOME_REG

QDC1_INDEX_REG

QDC1_PHASEB_REG

QDC1_PHASEA_REG

FlexPWM0_SM0_EXTSYNC_REG

FlexPWM0_SM1_EXTSYNC_REG

FlexPWM0_SM2_EXTSYNC_REG

FlexPWM0_SM3_EXTSYNC_REG

FlexPWM0_SM0_EXTA_REG

FlexPWM0_SM1_EXTA_REG

FlexPWM0_SM2_EXTA_REG

FlexPWM0_SM3_EXTA_REG

FlexPWM0_EXTFORCE_REG

FlexPWM0_FAULT0_REG

FlexPWM0_FAULT1_REG

FlexPWM0_FAULT2_REG

FlexPWM0_FAULT3_REG

FlexPWM1_SM0_EXTSYNC_REG

FlexPWM1_SM1_EXTSYNC_REG

FlexPWM1_SM2_EXTSYNC_REG

FlexPWM1_SM3_EXTSYNC_REG

FlexPWM1_SM0_EXTA_REG

FlexPWM1_SM1_EXTA_REG

FlexPWM1_SM2_EXTA_REG

FlexPWM1_SM3_EXTA_REG

FlexPWM1_EXTFORCE_REG

FlexPWM1_FAULT0_REG

FlexPWM1_FAULT1_REG

FlexPWM1_FAULT2_REG

FlexPWM1_FAULT3_REG

PWM0_EXT_CLK_REG

PWM1_EXT_CLK_REG

EVTG_TRIG0_REG

USBFS_TRIG_REG

TSI_TRIG_REG

EXT_TRIG0_REG

CMP1_TRIG_REG

CMP2_TRIG_REG

SINC_FILTER_CH0_REG

OPAMP0_TRIG_REG

OPAMP1_TRIG_REG

OPAMP2_TRIG_REG

FLEXCOMM0_TRIG_REG

FLEXCOMM1_TRIG_REG

FLEXCOMM2_TRIG_REG

FLEXCOMM3_TRIG_REG

FLEXCOMM4_TRIG_REG

FLEXCOMM5_TRIG_REG

FLEXCOMM6_TRIG_REG

FLEXCOMM7_TRIG_REG

FLEXCOMM8_TRIG_REG

FLEXCOMM9_TRIG_REG

FLEXIO_TRIG0_REG

DMA0_REQ_ENABLE0_REG

DMA0_REQ_ENABLE1_REG

DMA0_REQ_ENABLE2_REG

DMA0_REQ_ENABLE3_REG

DMA1_REQ_ENABLE0_REG

DMA1_REQ_ENABLE1_REG

DMA1_REQ_ENABLE2_REG

DMA1_REQ_ENABLE3_REG

ENA_SHIFT

PMUX_SHIFT

FSL_INPUTMUX_DRIVER_VERSION
Group interrupt driver version for SDK.

void INPUTMUX_Init(void *base)
Initialize INPUTMUX peripheral.

This function enables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

Return values
None. –

void INPUTMUX_AttachSignal(void *base, uint16_t index, *inputmux_connection_t* connection)
Attaches a signal.

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
```

In this example, INTMUX has 8 registers for PINT, PINT_SEL0~PINT_SEL7. With parameter index specified as 2, this function configures register PINT_SEL2.

Parameters

- base – Base address of the INPUTMUX peripheral.
- index – The serial number of destination register in the group of INPUTMUX registers with same name.
- connection – Applies signal from source signals collection to target signal.

Return values

None. –

void INPUTMUX_EnableSignal(void *base, *inputmux_signal_t* signal, bool enable)

Enable/disable a signal.

This function gates the INPUTPMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.
- signal – Enable signal register id and bit offset.
- enable – Selects enable or disable.

Return values

None. –

void INPUTMUX_Deinit(void *base)

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

Return values

None. –

2.48 INTM: Interrupt Monitor Driver

FSL_INTM_DRIVER_VERSION

INTM driver version.

enum _intm_monitor

Interrupt monitors.

Values:

enumerator kINTM_Monitor1

enumerator kINTM_Monitor2

enumerator kINTM_Monitor3

enumerator kINTM_Monitor4

typedef enum *_intm_monitor* intm_monitor_t

Interrupt monitors.

typedef struct *_intm_monitor_config* intm_monitor_config_t

INTM interrupt source configuration structure.

typedef struct *_intm_config* intm_config_t

INTM configuration structure.

void INTM_GetDefaultConfig(*intm_config_t* *config)

Fill in the INTM config struct with the default settings.

The default values are:

```
config[0].irqnumber = NotAvail_IRQn;
config[0].maxtimer = 1000U;
config[1].irqnumber = NotAvail_IRQn;
config[1].maxtimer = 1000U;
config[2].irqnumber = NotAvail_IRQn;
config[2].maxtimer = 1000U;
config[3].irqnumber = NotAvail_IRQn;
config[3].maxtimer = 1000U;
config->enable = false;
```

Parameters

- config – Pointer to user's INTM config structure.

void INTM_Init(INTM_Type *base, const *intm_config_t* *config)

Ungates the INTM clock and configures the peripheral for basic operation.

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

Parameters

- base – INTM peripheral base address
- config – Pointer to user's INTM config structure.

void INTM_Deinit(INTM_Type *base)

Disables the INTM module.

Parameters

- base – INTM peripheral base address

static inline void INTM_EnableCycleCount(INTM_Type *base, bool enable)

Enable the cycle count timer mode.

Monitor mode enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

Parameters

- base – INTM peripheral base address.
- enable – Enable the cycle count or not.

static inline void INTM_AckIrq(INTM_Type *base, IRQn_Type irq)

Interrupt Acknowledge.

Call this function in ISR to acknowledge interrupt.

Parameters

- base – INTM peripheral base address.
- irq – Handle interrupt number.

static inline void INTM_SetInterruptRequestNumber(INTM_Type *base, *intm_monitor_t* intms, IRQn_Type irq)

Interrupt Request Select.

This function is used to set the interrupt request number to monitor or check.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.
- irq – Interrupt request number to monitor.

Returns

Select the interrupt request number to monitor.

static inline void INTM_SetMaxTime(INTM_Type *base, *intm_monitor_t* intms, uint32_t count)

Set the maximum count time.

This function is to set the maximum time from interrupt generation to confirmation.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.
- count – Timer maximum count.

static inline void INTM_ClearTimeCount(INTM_Type *base, *intm_monitor_t* intms)

Clear the timer period in units of count.

This function is used to clear the INTM_TIMERa register.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

static inline uint32_t INTM_GetTimeCount(INTM_Type *base, *intm_monitor_t* intms)

Gets the timer period in units of count.

This function is used to get the number of INTM clock cycles from interrupt request to confirmation interrupt processing. If this number exceeds the set maximum time, will be an error signal.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

static inline bool INTM_GetStatusFlags(INTM_Type *base, *intm_monitor_t* intms)

Interrupt monitor status.

This function indicates whether the INTM_TIMERa value has exceeded the INTM_LATENCYa value. If any interrupt source in INTM_TIMERa exceeds the programmed delay value, the monitor state can be cleared by calling the INTM_ClearTimeCount() API to clear the corresponding INTM_TIMERa register.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

Returns

Whether INTM_TIMER value has exceeded INTM_LATENCY value. false:INTM_TIMER value has not exceeded the INTM_LATENCY value; true:INTM_TIMER value has exceeded the INTM_LATENCY value.

struct _intm_monitor_config

#include <fsl_intm.h> INTM interrupt source configuration structure.

Public Members

uint32_t maxtimer

Set the maximum timer

IRQn_Type irqnumber

Select the interrupt request number to monitor.

struct _intm_config

#include <fsl_intm.h> INTM configuration structure.

Public Members

bool enable

Interrupt source monitor config. enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

2.49 Inline Prince Encryption Decryption

FSL_IPED_DRIVER_VERSION

IPED driver version. Version 2.2.0.

Current version: 2.2.0

Change log:

- Version 2.2.0
  - Renamed CSS to ELS
- Version 2.1.1
  - Fix build error due to renamed symbols
- Version 2.1.0
  - Add IPED_Config() (including CMPA write) and IPED_Reconfig() features.
- Version 2.0.0
  - Initial version

enum _iped_status

Values:

enumerator kStatus_IPED_RegionIsLocked

enum _iped_lock

Values:

enumerator kIPED_RegionUnlock

enumerator kIPED_RegionLock

enum _iped_cmpa

Values:

enumerator kIPED_SkipCMPA

enumerator kIPED_WriteCMPA

typedef uint32_t iped_region_t

typedef uint32_t iped_prince_rounds_t

typedef enum *_iped_lock* iped_lock_t

typedef enum *_iped_cmpa* iped_cmpa_t

typedef struct *_flexspi_iped_region_option* flexspi_iped_prot_region_option_t

typedef struct *_flexspi_iped_region_arg* flexspi_iped_region_arg_t

static inline void IPED_EncryptEnable(FLEXSPI_Type *base)

Enable data encryption.

This function enables IPED on-the-fly data encryption.

Parameters

- base – IPED peripheral address.

static inline void IPED_EncryptDisable(FLEXSPI_Type *base)

Disable data encryption.

This function disables IPED on-the-fly data encryption.

Parameters

- base – IPED peripheral address.

static inline void IPED_SetLock(FLEXSPI_Type *base, *iped_region_t* region)

Locks access for specified region registers or data mask register.

This function sets lock on specified region.

Parameters

- base – IPED peripheral address.
- region – number to lock

status_t IPED_SetRegionAddressRange(FLEXSPI_Type *base, *iped_region_t* region, uint32_t start_address, uint32_t end_address)

Sets IPED region address range.

This function configures IPED region address range.

Parameters

- base – IPED peripheral address.
- region – Selection of the IPED region to be configured.
- start_address – Start address for region.
- end_address – End address for region.

status_t IPED_GetRegionAddressRange(FLEXSPI_Type *base, *iped_region_t* region, uint32_t *start_address, uint32_t *end_address)

Gets IPED region base address.

This function reads current start and end address settings for selected region.

Parameters

- base – IPED peripheral address.
- region – Selection of the IPED region to be configured.
- start_address – Start address for region.
- end_address – End address for region.

status_t IPED_SetRegionIV(FLEXSPI_Type *base, *iped_region_t* region, const uint8_t iv[8])

Sets the IPED region IV.

This function sets specified AES IV for the given region.

Parameters

- base – IPED peripheral address.
- region – Selection of the IPED region to be configured.
- iv – 64-bit AES IV in little-endian byte order.

static inline void IPED_SetPrinceRounds(FLEXSPI_Type *base, *iped_prince_rounds_t* rounds)

Sets the IPED region IV.

This function sets specified AES IV for the given region.

Parameters

- base – IPED peripheral address.
- rounds – Number of PRINCE rounds used during encryption/decryption

status_t IPED_Configure(*api_core_context_t* *coreCtx, *flexspi_iped_region_arg_t* *config, *iped_lock_t* lock, *iped_cmpa_t* writeCmpa)

Configures IPED setting.

This function does the initial IPED configuration via ROM IAP API call. IPED_SR_x configuration for each region configuration is stored into FFR (CMPA). IPED IV erase counters (MCTR_INT_IV_CTRx) in CFPA are updated accordingly.

Note: This function is expected to be called once in the device lifetime, typically during the initial device provisioning (especially if programming the CMPA pages in PFR flash is enabled).

Parameters

- coreCtx – The pointer to the ROM API driver context structure.
- config – The pointer to the IPED driver configuration structure.
- lock – Locks the IPED configuration, if CMPA write enabled, also sets the IPEDx_START bits[1:0] 01 - Enabled, 10,11 - Enabled & locked
- writeCmpa – If selected, IPED configuration will be programmed in PFR flash using ROM API. Note: This can not be reverted!!

Return values

- kStatus_Success –
- kStatus_CommandUnsupported –
- kStatus_InvalidArgument –

- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatusMemoryRangeInvalid –
- kStatus_Fail –
- kStatus_OutOfRange –
- #kStatus_SPI_BaudrateNotSupport –

status_t IPED_Reconfigure(*api_core_context_t* *coreCtx, *flexspi_iped_region_arg_t* *config)

Configures IPED setting.

This function is used to re-configure IPED IP based on configuration stored in FFR. This function also needs to be called after wake up from power-down mode to regenerate IV encryption key in ELS key store whose presence is necessary for correct IPED operation during erase and write operations to encrypted regions of internal flash memory (dependency for correct operation of MEM_Erase() and MEM_Write() after wake up from power-down mode).

Parameters

- coreCtx – The pointer to the ROM API driver context structure.
- config – The pointer to the IPED driver configuration structure. If NULL CMPA cinfiguration is read and used. Note: when providing config structure, you have to call Reconfigure for each IPED region individually starting with Region 0. Region 0 must be enabled as a base region.

Return values

- kStatus_Success –
- kStatus_Fail –

kIPED_Region0

IPED region 0

kIPED_Region1

IPED region 1

kIPED_Region2

IPED region 2

kIPED_Region3

IPED region 3

kIPED_PrinceRounds12

kIPED_PrinceRounds22

IPED_TAG

IPED fixed tag in flexspi_iped_region_arg_t structure.

IPED_TAG_SHIFT

IPED_REGION_COUNT

IPED region count.

IPED_RW_ENABLE_VAL

IPED_RW_DISABLE_VAL

NXP_DIE_EXT_MEM_ENC_SK

Define for ELS key store indexes.

NXP_DIE_MEM_IV_ENC_SK

CFPA_VER_OFFSET
: CFPA version and IV indexes (see Protected Flash Region table)

CFPA_IPED_IV_OFFSET

CFPA_SCRATCH_VER
: CFPA scrach version and IV addresses (see Protected Flash Region table)

CFPA_SCRATCH_IV

IPED_START_ADDR_LOCK_EN_MASK
: CMPA start address, end address, lock and enable bit-field masks (see Protected Flash Region table)

IPED_START_ADDR_MASK

IPED_END_ADDR_MASK

SYSCON_ELS_KDF_MASK
: KDF mask and key properties for NXP_DIE_MEM_IV_ENC_SK (see SYSCON documentation)

CMPA_IPED_START_OFFSET
: CMPA Start address index (see Protected Flash Region table)

IPED_ADDRESS_MASK
: CMPA Start address mask (see Protected Flash Region table)

IPED_ENABLE_MASK
: CMPA Start address enable/lock mask bits (see Protected Flash Region table)

CMPA_PAGE_SIZE
: CMPA page size (see Protected Flash Region table)

struct _flexspi_iped_region_option
: *#include <fsl_iped.h>*

struct _flexspi_iped_region_arg
: *#include <fsl_iped.h>*

struct IPED_CMPA_page
: *#include <fsl_iped.h>* IPED - CMPA page layout.

Public Members

uint8_t RESERVED_0[144]
: Reserved 0, offset: 0x00

__IO uint32_t IPED0_START
: IPED0_START, offset: 0x90

__IO uint32_t IPED0_END
: IPED0_END, offset: 0x94

__IO uint32_t IPED1_START
: IPED1_START, offset: 0x98

__IO uint32_t IPED1_END
: IPED1_END, offset: 0x9C

__IO uint32_t IPED2_START
: IPED2_START, offset: 0xA0

___IO uint32_t IPED2_END
IPED2_END, offset: 0xA4

___IO uint32_t IPED3_START
IPED3_START, offset: 0xA8

___IO uint32_t IPED3_END
IPED3_END, offset: 0xAC

uint8_t RESERVED_1[336]
Reserved 1, offset: 0xB0

2.50 IRTC: IRTC Driver

status_t IRTC_Init(RTC_Type *base, const *irtc_config_t* *config)

Ungates the IRTC clock and configures the peripheral for basic operation.

This function initiates a soft-reset of the IRTC module, this has not effect on DST, calendaring, standby time and tamper detect registers.

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

Parameters

- base – IRTC peripheral base address
- config – Pointer to user's IRTC config structure.

Returns
kStatus_Success If the driver is initialized successfully.

Returns
kStatus_Fail if we cannot disable register write protection

Returns
kStatus_InvalidArgument If the input parameters are wrong.

status_t IRTC_Deinit(RTC_Type *base)

Gate the IRTC clock.

Parameters

- base – IRTC peripheral base address

Returns
kStatus_Success If the driver is initialized successfully.

Returns
kStatus_InvalidArgument If the input parameters are wrong.

void IRTC_GetDefaultConfig(*irtc_config_t* *config)

Fill in the IRTC config struct with the default settings.

The default values are:

```
config->wakeupSelect = true;
config->timerStdMask = false;
config->alrmMatch = kRTC_MatchSecMinHr;
```

Parameters

- config – Pointer to user's IRTC config structure.

status_t IRTC_SetDatetime(RTC_Type *base, const *irtc_datetime_t* *datetime)

Sets the IRTC date and time according to the given time structure.

The IRTC counter is started after the time is set.

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to structure where the date and time details to set are stored

Returns

kStatus_Success: success in setting the time and starting the IRTC kStatus_InvalidArgument: failure. An error occurs because the datetime format is incorrect.

void IRTC_GetDatetime(RTC_Type *base, *irtc_datetime_t* *datetime)

Gets the IRTC time and stores it in the given time structure.

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to structure where the date and time details are stored.

status_t IRTC_SetAlarm(RTC_Type *base, const *irtc_datetime_t* *alarmTime)

Sets the IRTC alarm time.

Note: weekDay field of alarmTime is not used during alarm match and should be set to 0

Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the alarm kStatus_InvalidArgument: error in setting the alarm. Error occurs because the alarm datetime format is incorrect.

void IRTC_GetAlarm(RTC_Type *base, *irtc_datetime_t* *datetime)

Returns the IRTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the alarm date and time details are stored.

static inline void IRTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected IRTC interrupts.

Parameters

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

static inline void IRTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected IRTC interrupts.

Parameters

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

static inline uint32_t IRTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled IRTC interrupts.

Parameters

- base – IRTC peripheral base address

Returns

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

static inline uint32_t IRTC_GetStatusFlags(RTC_Type *base)

Gets the IRTC status flags.

Parameters

- base – IRTC peripheral base address

Returns

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

static inline void IRTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clears the IRTC status flags.

Parameters

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

void IRTC_SetDaylightTime(RTC_Type *base, const *irtc_daylight_time_t* *datetime)

Sets the IRTC daylight savings start and stop date and time.

It also enables the daylight saving bit in the IRTC control register

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to a structure where the date and time details are stored.

void IRTC_GetDaylightTime(RTC_Type *base, *irtc_daylight_time_t* *datetime)

Gets the IRTC daylight savings time and stores it in the given time structure.

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to a structure where the date and time details are stored.

void IRTC_SetCoarseCompensation(RTC_Type *base, uint8_t compensationValue, uint8_t compensationInterval)

Enables the coarse compensation and sets the value in the IRTC compensation register.

Parameters

- base – IRTC peripheral base address

- compensationValue – Compensation value is a 2's complement value.
- compensationInterval – Compensation interval.

void IRTC_SetFineCompensation(RTC_Type *base, uint8_t integralValue, uint8_t fractionValue, bool accumulateFractional)

Enables the fine compensation and sets the value in the IRTC compensation register.

Parameters

- base – The IRTC peripheral base address
- integralValue – Compensation integral value; twos complement value of the integer part
- fractionValue – Compensation fraction value expressed as number of clock cycles of a fixed 4.194304Mhz clock that have to be added.
- accumulateFractional – Flag indicating if we want to add to previous fractional part; true: Add to previously accumulated fractional part, false: Start afresh and overwrite current value

static inline void IRTC_EnableSubsecondCounter(RTC_Type *base, bool enable)

Enable the RTC wake-up timer.

1HZ clock out selected via call to API IRTC_ConfigClockOut in order for the subsecond counter to synchronize with the RTC_SECONDS counter.

Parameters

- base – RTC peripheral base address
- enable – Use/Un-use the sub-second counter.
  - true: Use RTC wake-up timer at the same time.
  - false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.

static inline uint32_t IRTC_GetSubsecondCount(RTC_Type *base)

Read the actual RTC sub-second COUNT value.

Parameters

- base – RTC peripheral base address

Returns

The actual RTC sub-second COUNT value.

static inline void IRTC_SetWakeupCount(RTC_Type *base, bool enable1kHzClk, uint32_t wakeupValue)

Set countdown value to the RTC wake timer counter register.

Parameters

- base – RTC peripheral base address
- enable1kHzClk – Enable 1kHz clock source for the wake timer, else use the 32kHz clock.
- wakeupValue – The value to be loaded into the WAKE register in wake timer counter.

static inline uint32_t IRTC_GetWakeupCount(RTC_Type *base)

Read the actual value from the WAKE register value in RTC wake timer.

Parameters

- base – RTC peripheral base address

Returns
The actual value of the WAKE register value in wake timer counter.

FSL_IRTC_DRIVER_VERSION

enum _irtc_clock_select
IRTC clock select.

Values:

enumerator kIRTC_Clk16K
16.384 kHz clock is selected.

enumerator kIRTC_Clk32K
32.768 kHz clock is selected.

enum _irtc_interrupt_enable
List of IRTC interrupts.

Values:

enumerator kIRTC_AlarmInterruptEnable
Alarm Interrupt Enable

enumerator kIRTC_DayInterruptEnable
Days Interrupt Enable

enumerator kIRTC_HourInterruptEnable
Hours Interrupt Enable

enumerator kIRTC_MinInterruptEnable
Minutes Interrupt Enable

enumerator kIRTC_1hzInterruptEnable
1 Hz interval Interrupt Enable

enumerator kIRTC_2hzInterruptEnable
2 Hz interval Interrupt Enable

enumerator kIRTC_4hzInterruptEnable
4 Hz interval Interrupt Enable

enumerator kIRTC_8hzInterruptEnable
8 Hz interval Interrupt Enable

enumerator kIRTC_16hzInterruptEnable
16 Hz interval Interrupt Enable

enumerator kIRTC_32hzInterruptEnable
32 Hz interval Interrupt Enable

enumerator kIRTC_64hzInterruptEnable
64 Hz interval Interrupt Enable

enumerator kIRTC_128hzInterruptEnable
128 Hz interval Interrupt Enable

enumerator kIRTC_256hzInterruptEnable
256 Hz interval Interrupt Enable

enumerator kIRTC_512hzInterruptEnable
512 Hz interval Interrupt Enable

enumerator kIRTC_WakeTimerInterruptEnable
Wake timer Interrupt Enable

enumerator kIRTC_TamperQueueFullInterruptEnable
Tamper queue full Interrupt Enable

enum _irtc_status_flags
List of IRTC flags.

Values:

enumerator kIRTC_AlarmFlag
Alarm Status flag

enumerator kIRTC_DayFlag
Days Status flag

enumerator kIRTC_HourFlag
Hour Status flag

enumerator kIRTC_MinFlag
Minutes Status flag

enumerator kIRTC_1hzFlag
1 Hz interval status flag

enumerator kIRTC_2hzFlag
2 Hz interval status flag

enumerator kIRTC_4hzFlag
4 Hz interval status flag

enumerator kIRTC_8hzFlag
8 Hz interval status flag

enumerator kIRTC_16hzFlag
16 Hz interval status flag

enumerator kIRTC_32hzFlag
32 Hz interval status flag

enumerator kIRTC_64hzFlag
64 Hz interval status flag

enumerator kIRTC_128hzFlag
128 Hz interval status flag

enumerator kIRTC_256hzFlag
256 Hz interval status flag

enumerator kIRTC_512hzFlag
512 Hz interval status flag

enumerator kIRTC_InvalidFlag
Indicates if time/date counters are invalid

enumerator kIRTC_WriteProtFlag
Write protect enable status flag

enumerator kIRTC_CmpIntFlag
Compensation interval status flag

enumerator kIRTC_CmpDoneFlag
Compensation done flag

enumerator kIRTC_BusErrFlag
Bus error flag

enumerator kIRTC_WakeTimerFlag
Wake timer status flag

enum _irtc_alarm_match
IRTC alarm match options.

Values:

enumerator kRTC_MatchSecMinHr
Only match second, minute and hour

enumerator kRTC_MatchSecMinHrDay
Only match second, minute, hour and day

enumerator kRTC_MatchSecMinHrDayMnth
Only match second, minute, hour, day and month

enumerator kRTC_MatchSecMinHrDayMnthYr
Only match second, minute, hour, day, month and year

enum _irtc_clockout_sel
IRTC clockout select.

Values:

enumerator kIRTC_ClkoutNo
No clock out

enumerator kIRTC_ClkoutFine1Hz
clock out fine 1Hz

enumerator kIRTC_Clkout32kHz
clock out 32.768kHz

enumerator kIRTC_ClkoutCoarse1Hz
clock out coarse 1Hz

typedef enum *_irtc_clock_select* irtc_clock_select_t
IRTC clock select.

typedef enum *_irtc_interrupt_enable* irtc_interrupt_enable_t
List of IRTC interrupts.

typedef enum *_irtc_status_flags* irtc_status_flags_t
List of IRTC flags.

typedef enum *_irtc_alarm_match* irtc_alarm_match_t
IRTC alarm match options.

typedef enum *_irtc_clockout_sel* irtc_clockout_sel_t
IRTC clockout select.

typedef struct *_irtc_datetime* irtc_datetime_t
Structure is used to hold the date and time.

typedef struct *_irtc_daylight_time* irtc_daylight_time_t
Structure is used to hold the daylight saving time.

typedef struct *_irtc_config* irtc_config_t

RTC config structure.

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

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

status_t IRTC_SetWriteProtection(RTC_Type *base, bool lock)

Locks or unlocks IRTC registers for write access.

Note: When the registers are unlocked, they remain in unlocked state for 2 seconds, after which they are locked automatically. After power-on-reset, the registers come out unlocked and they are locked automatically 15 seconds after power on.

Parameters

- base – IRTC peripheral base address
- lock – true: Lock IRTC registers; false: Unlock IRTC registers.

Returns

kStatus_Success: if lock or unlock operation is successful kStatus_Fail: if lock or unlock operation fails even after multiple retry attempts

static inline void IRTC_Reset(RTC_Type *base)

Performs a software reset on the IRTC module.

Clears contents of alarm, interrupt (status and enable except tamper interrupt enable bit) registers, STATUS[CMP_DONE] and STATUS[BUS_ERR]. This has no effect on DST, calendaring, standby time and tamper detect registers.

Parameters

- base – IRTC peripheral base address

void IRTC_ConfigClockOut(RTC_Type *base, *irtc_clockout_sel_t* clkOut)

Select which clock to output from RTC.

Select which clock to output from RTC for other modules to use inside SoC, for example, RTC subsystem needs RTC to output 1HZ clock for sub-second counter.

Parameters

- base – IRTC peripheral base address
- clkOut – select clock to use for output,

void IRTC_ConfigClockSelect(RTC_Type *base, *irtc_clock_select_t* clkSelect)

Select which clock is used by RTC.

Select which clock is used by RTC to output to the peripheral and divided to generate a 512 Hz clock and a 1 Hz clock.

Parameters

- base – IRTC peripheral base address
- clkSelect – select clock used by RTC

static inline void IRTC_EnableClockOutputToPeripheral(RTC_Type *base, bool enable)

Determines whether the selected clock is output to other peripherals.

Determines whether the selected clock is output to other peripherals.

Parameters

- base – IRTC peripheral base address
- enable – determine whether the selected clock is output to other peripherals

IRTC_STATUS_W1C_BITS

struct _irtc_datetime

#include <fsl_irtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1984 to 2239.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t weekDay

Range from 0(Sunday) to 6(Saturday).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _irtc_daylight_time

#include <fsl_irtc.h> Structure is used to hold the daylight saving time.

Public Members

uint8_t startMonth

Range from 1 to 12

uint8_t endMonth

Range from 1 to 12

uint8_t startDay

Range from 1 to 31 (depending on month)

uint8_t endDay

Range from 1 to 31 (depending on month)

uint8_t startHour

Range from 0 to 23

uint8_t endHour

Range from 0 to 23

struct _irtc_config

#include <fsl_irtc.h> RTC config structure.

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

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

Public Members

irtc_alarm_match_t alrmMatch

Pick one option from enumeration :: irtc_alarm_match_t

irtc_clock_select_t clockSelect

Pick one option from enumeration :: irtc_clock_select_t

bool disableClockOutput

true: The selected clock is not output to other peripherals; false: The selected clock is output to other peripherals

2.51 Intrusion and Tamper Response Controller

2.52 ITRC

status_t ITRC_SetActionToEvent(ITRC_Type *base, *itrc_out_signals_t* out, *itrc_input_signals_t* in, *itrc_lock_t* lock, *itrc_enable_t* enable)

Set ITRC Action to Event.

This function sets input Event signal to corresponding output Action response signal.

Parameters

- base – ITRC peripheral base address
- out – ITRC OUT signal action
- in – ITRC IN signal event
- lock – if set locks INx_SEL configuration. This can be cleared only by PMC Core reset.
- enable – if set input Event will be selected for output Action, otherwise disable (if not already locked).

Returns

kStatus_Success if success, kStatus_InvalidArgument otherwise

void ITRC_SetSWEvent0(ITRC_Type *base)

Trigger ITRC SW Event 0.

This funciton set SW_EVENT0 register with value !=0 which triggers ITRC SW Event 0.

Parameters

- base – ITRC peripheral base address

void ITRC_SetSWEvent1(ITRC_Type *base)

Trigger ITRC SW Event 1.

This funciton set SW_EVENT1 register with value !=0 which triggers ITRC SW Event 1.

Parameters

- base – ITRC peripheral base address

uint32_t ITRC_GetStatus(ITRC_Type *base)

Get ITRC Status.

This function returns ITRC register status.

Parameters

- base – ITRC peripheral base address

Returns

Value of ITRC STATUS register

status_t ITRC_ClearStatus(ITRC_Type *base, uint32_t word)

Clear ITRC status.

This function clears corresponding ITRC event or action in STATUS register.

Parameters

- base – ITRC peripheral base address
- word – 32bit word represent corresponding event/action in STATUS register to be cleared (see ITRC_STATUS_INx/OUTx_STATUS)

Returns

kStatus_Success if success, kStatus_InvalidArgument otherwise

status_t ITRC_ClearAllStatus(ITRC_Type *base)

Clear All ITRC status.

This function clears all event and action status.

Parameters

- base – ITRC peripheral base address

Returns

kStatus_Success if success

status_t ITRC_Init(ITRC_Type *base)

Initialize ITRC.

This function initializes ITRC by enabling IRQ.

Parameters

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

Returns

Status of the init operation

void ITRC_Deinit(ITRC_Type *base)

Deinitialize ITRC.

This function deinitializes ITRC by disabling IRQ.

Parameters

- base – ITRC peripheral base address

FSL_ITRC_DRIVER_VERSION

Defines ITRC driver version 2.4.0.

Change log:

- Version 2.4.0

- Rework the input signal definition for better flexibility

- Version 2.3.0
  - Update names of kITRC_SwEvent1/2 to kITRC_SwEvent0/1 to align with RM
- Version 2.2.0
  - Update driver to new version and input events
- Version 2.1.0
  - Make SYSCON glitch platform dependent
- Version 2.0.0
  - initial version

enum _itrc_input_signals

Values:

enum _itrc_lock

Values:

enumerator kITRC_Unlock

enumerator kITRC_Lock

enum _itrc_enable

Values:

enumerator kITRC_Enable

enumerator kITRC_Disable

enum _itrc_out_signals

Values:

enumerator kITRC_Irq

enumerator kITRC_ElsReset

enumerator kITRC_PufZeroize

enumerator kITRC_RamZeroize

enumerator kITRC_ChipReset

enumerator kITRC_TamperOut

enumerator kITRC_TamperOut1

typedef enum *_itrc_input_signals* itrc_input_signals_t

typedef enum *_itrc_lock* itrc_lock_t

typedef enum *_itrc_enable* itrc_enable_t

typedef enum *_itrc_out_signals* itrc_out_signals_t

void ITRC0_DriverIRQHandler(void)

ITRC_STATUS_IN2_STATUS_MASK

ITRC_STATUS_IN3_STATUS_MASK

ITRC_STATUS_IN9_STATUS_MASK

ITRC_STATUS1_IN17_STATUS_MASK

ITRC_STATUS1_IN19_STATUS_MASK

ITRC_STATUS1_IN24_21_STATUS_MASK

ITRC_STATUS1_IN32_25_STATUS_MASK

ITRC_STATUS1_IN46_STATUS_MASK

IN_0_15_EVENTS_MASK

OUT_ACTIONS_MASK

ITRC_OUT_COUNT

ITRC

2.53 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_SIZEALIGN(var, alignbytes)
: Macro to define a variable with L1 d-cache line size alignment

  Macro to define a variable with L2 cache line size alignment

  Macro to change a value to a given size aligned value

AT_NONCACHEABLE_SECTION(var)
: Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)
: Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_CACHE_LINE_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

AT_QUICKACCESS_SECTION_CODE(func)

Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)

Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)

Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

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.

2.54 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.

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.

2.55 Lpc_freqme

void FREQME_Init(FREQME_Type *base, const *freq_measure_config_t* *config)
Initialize freqme module, set operate mode, operate mode attribute and initialize measurement cycle.

Parameters

- base – FREQME peripheral base address.
- config – The pointer to module basic configuration, please refer to freq_measure_config_t.

void FREQME_GetDefaultConfig(*freq_measure_config_t* *config)
Get default configuration.

```
config->operateMode = kFREQME_FreqMeasurementMode;
config->operateModeAttribute.refClkScaleFactor = 0U;
config->enableContinuousMode                   = false;
config->startMeasurement                       = false;
```

Parameters

- config – The pointer to module basic configuration, please refer to freq_measure_config_t.

static inline void FREQME_StartMeasurementCycle(FREQME_Type *base)
Start frequency or pulse width measurement process.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_TerminateMeasurementCycle(FREQME_Type *base)
Force the termination of any measurement cycle currently in progress and resets RESULT or just reset RESULT if the module in idle state.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_EnableContinuousMode(FREQME_Type *base, bool enable)
Enable/disable Continuous mode.

Parameters

- base – FREQME peripheral base address.
- enable – Used to enable/disable continuous mode,
  - **true** Enable Continuous mode.
  - **false** Disable Continuous mode.

static inline bool FREQME_CheckContinuousMode(FREQME_Type *base)

Check whether continuous mode is enabled.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Continuous mode is enabled, the measurement is performed continuously.
- False – Continuous mode is disabled.

static inline void FREQME_SetOperateMode(FREQME_Type *base, *freqme_operate_mode_t* operateMode)

Set operate mode of freqme module.

Parameters

- base – FREQME peripheral base address.
- operateMode – The operate mode to be set, please refer to freqme_operate_mode_t.

static inline bool FREQME_CheckOperateMode(FREQME_Type *base)

Check module's operate mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Pulse width measurement mode.
- False – Frequency measurement mode.

static inline void FREQME_SetMinExpectedValue(FREQME_Type *base, uint32_t minValue)

Set the minimum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- minValue – The minimum value to set, please note that this value is 31 bits width.

static inline void FREQME_SetMaxExpectedValue(FREQME_Type *base, uint32_t maxValue)

Set the maximum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- maxValue – The maximum value to set, please note that this value is 31 bits width.

uint32_t FREQME_CalculateTargetClkFreq(FREQME_Type *base, uint32_t refClkFrequency)

Calculate the frequency of selected target clock

Note: The formula: Ftarget = (RESULT - 2) * Freference / 2 ^ REF_SCALE.

Note: This function only useful when the operate mode is selected as frequency measurement mode.

Parameters

- base – FREQME peripheral base address.
- refClkFrequency – The frequency of reference clock.

Returns
The frequency of target clock the unit is Hz, if the output result is 0, please check the module's operate mode.

static inline uint8_t FREQME_GetReferenceClkScaleValue(FREQME_Type *base)

Get reference clock scaling factor.

Parameters

- base – FREQME peripheral base address.

Returns
Reference clock scaling factor, the reference count cycle is 2 ^ ref_scale.

static inline void FREQME_SetPulsePolarity(FREQME_Type *base, *freqme_pulse_polarity_t* pulsePolarity)

Set pulse polarity when operate mode is selected as Pulse Width Measurement mode.

Parameters

- base – FREQME peripheral base address.
- pulsePolarity – The pulse polarity to be set, please refer to freqme_pulse_polarity_t.

static inline bool FREQME_CheckPulsePolarity(FREQME_Type *base)

Check pulse polarity when the operate mode is selected as pulse width measurement mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Low period.
- False – High period.

static inline uint32_t FREQME_GetMeasurementResult(FREQME_Type *base)

Get measurement result, if operate mode is selected as pulse width measurement mode this function can be used to calculate pulse width.

Note: Pulse width = counter result / Frequency of target clock.

Parameters

- base – FREQME peripheral base address.

Returns
Measurement result.

static inline uint32_t FREQME_GetInterruptStatusFlags(FREQME_Type *base)

Get interrupt status flags, such as overflow interrupt status flag, underflow interrupt status flag, and so on.

Parameters

- base – FREQME peripheral base address.

Returns

Current interrupt status flags, should be the OR'ed value of _freqme_interrupt_status_flags.

static inline void FREQME_ClearInterruptStatusFlags(FREQME_Type *base, uint32_t statusFlags)

Clear interrupt status flags.

Parameters

- base – FREQME peripheral base address.
- statusFlags – The combination of interrupt status flags to clear, should be the OR'ed value of _freqme_interrupt_status_flags.

static inline void FREQME_EnableInterrupts(FREQME_Type *base, uint32_t masks)

Enable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to enable, should be the OR'ed value of _freqme_interrupt_enable.

static inline void FREQME_DisableInterrupts(FREQME_Type *base, uint32_t masks)

Disable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to disable, should be the OR'ed value of _freqme_interrupt_enable.

FSL_FREQME_DRIVER_VERSION

FREQME driver version 2.1.2.

enum _freqme_interrupt_status_flags

The enumeration of interrupt status flags. .

Values:

enumerator kFREQME_UnderflowInterruptStatusFlag

Indicate the measurement is just done and the result is less than minimun value.

enumerator kFREQME_OverflowInterruptStatusFlag

Indicate the measurement is just done and the result is greater than maximum value.

enumerator kFREQME_ReadyInterruptStatusFlag

Indicate the measurement is just done and the result is ready to read.

enumerator kFREQME_AllInterruptStatusFlags

All interrupt status flags.

enum _freqme_interrupt_enable

The enumeration of interrupts, including underflow interrupt, overflow interrupt, and result ready interrupt. .

Values:

enumerator kFREQME_UnderflowInterruptEnable

Enable interrupt when the result is less than minimum value.

enumerator kFREQME_OverflowInterruptEnable

Enable interrupt when the result is greater than maximum value.

enumerator kFREQME_ReadyInterruptEnable

Enable interrupt when a measurement completes and the result is ready.

enum _freqme_operate_mode

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

Values:

enumerator kFREQME_FreqMeasurementMode

The module works in the frequency measurement mode.

enumerator kFREOME_PulseWidthMeasurementMode

The module works in the pulse width measurement mode.

enum _freqme_pulse_polarity

The enumeration of pulse polarity.

Values:

enumerator kFREQME_PulseHighPeriod

Select high period of the reference clock.

enumerator kFREQME_PulseLowPeriod

Select low period of the reference clock.

typedef enum *_freqme_operate_mode* freqme_operate_mode_t

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

typedef enum *_freqme_pulse_polarity* freqme_pulse_polarity_t

The enumeration of pulse polarity.

typedef union *_freqme_mode_attribute* freqme_mode_attribute_t

The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

---

typedef struct *_freq_measure_config* freq_measure_config_t

The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

union _freqme_mode_attribute

#include <fsl_freqme.h> The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

Public Members

uint8_t refClkScaleFactor

Only useful in frequency measurement operate mode, used to set the reference clock counter scaling factor.

freqme_pulse_polarity_t pulsePolarity

Only Useful in pulse width measurement operate mode, used to set period polarity.

struct _freq_measure_config

#include <fsl_freqme.h> The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

Public Members

freqme_operate_mode_t operateMode

Select operate mode, please refer to freqme_operate_mode_t.

freqme_mode_attribute_t operateModeAttribute

Used to set the attribute of the selected operate mode, if the operate mode is selected as kFREQME_FreqMeasurementMode set freqme_mode_attribute_t::refClkScaleFactor, if operate mode is selected as kFREOME_PulseWidthMeasurementMode, please set freqme_mode_attribute_t::pulsePolarity.

bool enableContinuousMode

Enable/disable continuous mode, if continuous mode is enable, the measurement is performed continuously and the result for the last completed measurement is available in the result register.

2.56 LPCMP: Low Power Analog Comparator Driver

void LPCMP_Init(LPCMP_Type *base, const *lpcmp_config_t* *config)

Initialize the LPCMP.

This function initializes the LPCMP module. The operations included are:

- Enabling the clock for LPCMP module.
- Configuring the comparator.
- Enabling the LPCMP module. Note: For some devices, multiple LPCMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the LPCMPs. Check the chip reference manual for the clock assignment of the LPCMP.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp_config_t” structure.

void LPCMP_Deinit(LPCMP_Type *base)

De-initializes the LPCMP module.

This function de-initializes the LPCMP module. The operations included are:

- Disabling the LPCMP module.
- Disabling the clock for LPCMP module.

This function disables the clock for the LPCMP. Note: For some devices, multiple LPCMP instance shares the same clock gate. In this case, before disabling the clock for the LPCMP, ensure that all the LPCMP instances are not used.

Parameters

- base – LPCMP peripheral base address.

void LPCMP_GetDefaultConfig(*lpcmp_config_t* *config)

Gets an available pre-defined settings for the comparator's configuration.

This function initializes the comparator configuration structure to these default values:

```
config->enableStopMode      = false;
config->enableOutputPin     = false;
config->enableCmpToDacLink  = false;
config->useUnfilteredOutput = false;
config->enableInvertOutput  = false;
config->hysteresisMode      = kLPCMP_HysteresisLevel0;
config->powerMode           = kLPCMP_LowSpeedPowerMode;
config->functionalSourceClock = kLPCMP_FunctionalClockSource0;
config->plusInputSrc        = kLPCMP_PlusInputSrcMux;
config->minusInputSrc       = kLPCMP_MinusInputSrcMux;
```

Parameters

- config – Pointer to "lpcmp_config_t" structure.

static inline void LPCMP_Enable(LPCMP_Type *base, bool enable)

Enable/Disable LPCMP module.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable the module, and "false" means disable the module.

void LPCMP_SetInputChannels(LPCMP_Type *base, uint32_t positiveChannel, uint32_t negativeChannel)

Select the input channels for LPCMP. This function determines which input is selected for the negative and positive mux.

Parameters

- base – LPCMP peripheral base address.
- positiveChannel – Positive side input channel number. Available range is 0-7.
- negativeChannel – Negative side input channel number. Available range is 0-7.

static inline void LPCMP_EnableDMA(LPCMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events. Normally, the LPCMP generates a CPU interrupt if there is a rising/falling event. When DMA support is enabled and the rising/falling interrupt is enabled , the rising/falling event forces a DMA transfer request rather than a CPU interrupt instead.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable DMA support, and "false" means disable DMA support.

void LPCMP_SetFilterConfig(LPCMP_Type *base, const *lpcmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_filter_config_t" structure.

void LPCMP_SetDACConfig(LPCMP_Type *base, const *lpcmp_dac_config_t* *config)

Configure the internal DAC module.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_dac_config_t" structure. If config is "NULL", disable internal DAC.

static inline void LPCMP_EnableInterrupts(LPCMP_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See "_lpcmp_interrupt_enable".

static inline void LPCMP_DisableInterrupts(LPCMP_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See "_lpcmp_interrupt_enable".

static inline uint32_t LPCMP_GetStatusFlags(LPCMP_Type *base)

Get the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.

Returns

Mask value for the asserted flags. See "_lpcmp_status_flags".

static inline void LPCMP_ClearStatusFlags(LPCMP_Type *base, uint32_t mask)

Clear the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for the flags. See "_lpcmp_status_flags".

static inline void LPCMP_EnableWindowMode(LPCMP_Type *base, bool enable)

Enable/Disable window mode.When any windowed mode is active, COUTA is clocked by the bus clock whenever WINDOW = 1. The last latched value is held when WINDOW = 0. The optionally inverted comparator output COUT_RAW is sampled on every bus clock when WINDOW=1 to generate COUTA.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable window mode, and "false" means disable window mode.

void LPCMP_SetWindowControl(LPCMP_Type *base, const *lpcmp_window_control_config_t* *config)

Configure the window control, users can use this API to implement operations on the window, such as inverting the window signal, setting the window closing event(only valid in windowing mode), and setting the COUTA signal after the window is closed(only valid in windowing mode).

Parameters

- base – LPCMP peripheral base address.
- config – Pointer "lpcmp_window_control_config_t" structure.

void LPCMP_SetRoundRobinConfig(LPCMP_Type *base, const *lpcmp_roundrobin_config_t* *config)

Configure the roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer "lpcmp_roundrobin_config_t" structure.

static inline void LPCMP_EnableRoundRobinMode(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable roundrobin mode, and "false" means disable roundrobin mode.

void LPCMP_SetRoundRobinInternalTimer(LPCMP_Type *base, uint32_t value)

brief Configure the roundrobin internal timer reload value.

param base LPCMP peripheral base address. param value RoundRobin internal timer reload value, allowed range:0x0UL-0xFFFFFFFUL.

static inline void LPCMP_EnableRoundRobinInternalTimer(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin internal timer, note that this function is only valid when using the internal trigger source.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable roundrobin internal timer, and "false" means disable roundrobin internal timer.

static inline void LPCMP_SetPreSetValue(LPCMP_Type *base, uint8_t mask)

Set preset value for all channels, users can set all channels' preset vaule through this API, for example, if the mask set to 0x03U means channel0 and channel2's preset value set to 1U and other channels' preset value set to 0U.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetComparisonResult(LPCMP_Type *base)

Get comparison results for all channels, users can get all channels' comparison results through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' comparison result.

static inline void LPCMP_ClearInputChangedFlags(LPCMP_Type *base, uint8_t mask)

Clear input changed flags for single channel or multiple channels, users can clear input changed flag of a single channel or multiple channels through this API, for example, if the mask set to 0x03U means clear channel0 and channel2's input changed flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetInputChangedFlags(LPCMP_Type *base)

Get input changed flags for all channels, Users can get all channels' input changed flags through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' changed flag.

FSL_LPCMP_DRIVER_VERSION

LPCMP driver version 2.3.2.

enum _lpcmp_status_flags

LPCMP status falgs mask.

Values:

enumerator kLPCMP_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kLPCMP_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kLPCMP_OutputRoundRobinEventFlag

Detects when any channel's last comparison result is different from the pre-set value in trigger mode.

enumerator kLPCMP_OutputAssertEventFlag

Return the current value of the analog comparator output. The flag does not support W1C.

enum _lpcmp_interrupt_enable

LPCMP interrupt enable/disable mask.

Values:

enumerator kLPCMP_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kLPCMP_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enumerator kLPCMP_RoundRobinInterruptEnable

Comparator round robin mode interrupt occurred when the comparison result changes for a given channel.

enum _lpcmp_hysteresis_mode

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

Values:

enumerator kLPCMP_HysteresisLevel0

The hard block output has level 0 hysteresis internally.

enumerator kLPCMP_HysteresisLevel1

The hard block output has level 1 hysteresis internally.

enumerator kLPCMP_HysteresisLevel2
The hard block output has level 2 hysteresis internally.

enumerator kLPCMP_HysteresisLevel3
The hard block output has level 3 hysteresis internally.

enum _lpcmp_power_mode
LPCMP nano mode.

Values:

enumerator kLPCMP_LowSpeedPowerMode
Low speed comparison mode is selected.

enumerator kLPCMP_HighSpeedPowerMode
High speed comparison mode is selected.

enumerator kLPCMP_NanoPowerMode
Nano power comparator is enabled.

enum _lpcmp_dac_reference_voltage_source
Internal DAC reference voltage source.

Values:

enumerator kLPCMP_VrefSourceVin1
vrefh_int is selected as resistor ladder network supply reference Vin.

enumerator kLPCMP_VrefSourceVin2
vrefh_ext is selected as resistor ladder network supply reference Vin.

enum _lpcmp_functional_source_clock
LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

Values:

enumerator kLPCMP_FunctionalClockSource0
Select functional mode clock source0.

enumerator kLPCMP_FunctionalClockSource1
Select functional mode clock source1.

enumerator kLPCMP_FunctionalClockSource2
Select functional mode clock source2.

enumerator kLPCMP_FunctionalClockSource3
Select functional mode clock source3.

enum _lpcmp_couta_signal
Set the COUTA signal value when the window is closed.

Values:

enumerator kLPCMP_COUTASignalNoSet
NO set the COUTA signal value when the window is closed.

enumerator kLPCMP_COUTASignalLow
Set COUTA signal low(0) when the window is closed.

enumerator kLPCMP_COUTASignalHigh
Set COUTA signal high(1) when the window is closed.

enum _lpcmp_close_window_event

Set COUT event, which can close the active window in window mode.

Values:

enumerator kLPCMP_CLoseWindowEventNoSet

No Set COUT event, which can close the active window in window mode.

enumerator kLPCMP_CloseWindowEventRisingEdge

Set rising edge COUT signal as COUT event.

enumerator kLPCMP_CloseWindowEventFallingEdge

Set falling edge COUT signal as COUT event.

enumerator kLPCMP_CLoseWindowEventBothEdge

Set both rising and falling edge COUT signal as COUT event.

enum _lpcmp_roundrobin_fixedmuxport

LPCMP round robin mode fixed mux port.

Values:

enumerator kLPCMP_FixedPlusMuxPort

Fixed plus mux port.

enumerator kLPCMP_FixedMinusMuxPort

Fixed minus mux port.

enum _lpcmp_roundrobin_clock_source

LPCMP round robin mode clock source selection.

Note: In different devices,the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

Values:

enumerator kLPCMP_RoundRobinClockSource0

Select roundrobin mode clock source0.

enumerator kLPCMP_RoundRobinClockSource1

Select roundrobin mode clock source1.

enumerator kLPCMP_RoundRobinClockSource2

Select roundrobin mode clock source2.

enumerator kLPCMP_RoundRobinClockSource3

Select roundrobin mode clock source3.

enum _lpcmp_roundrobin_trigger_source

LPCMP round robin mode trigger source.

Values:

enumerator kLPCMP_TriggerSourceExternally

Select external trigger source.

enumerator kLPCMP_TriggerSourceInternally

Select internal trigger source.

typedef enum *_lpcmp_hysteresis_mode* lpcmp_hysteresis_mode_t

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

typedef enum *_lpcmp_power_mode* lpcmp_power_mode_t

LPCMP nano mode.

typedef enum *_lpcmp_dac_reference_voltage_source* lpcmp_dac_reference_voltage_source_t
Internal DAC reference voltage source.

typedef enum *_lpcmp_functional_source_clock* lpcmp_functional_source_clock_t
LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

typedef enum *_lpcmp_couta_signal* lpcmp_couta_signal_t
Set the COUTA signal value when the window is closed.

typedef enum *_lpcmp_close_window_event* lpcmp_close_window_event_t
Set COUT event, which can close the active window in window mode.

typedef enum *_lpcmp_roundrobin_fixedmuxport* lpcmp_roundrobin_fixedmuxport_t
LPCMP round robin mode fixed mux port.

typedef enum *_lpcmp_roundrobin_clock_source* lpcmp_roundrobin_clock_source_t
LPCMP round robin mode clock source selection.

Note: In different devices,the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

typedef enum *_lpcmp_roundrobin_trigger_source* lpcmp_roundrobin_trigger_source_t
LPCMP round robin mode trigger source.

typedef struct *_lpcmp_filter_config* lpcmp_filter_config_t
Configure the filter.

typedef struct *_lpcmp_dac_config* lpcmp_dac_config_t
configure the internal DAC.

typedef struct *_lpcmp_config* lpcmp_config_t
Configures the comparator.

typedef struct *_lpcmp_window_control_config* lpcmp_window_control_config_t
Configure the window mode control.

typedef struct *_lpcmp_roundrobin_config* lpcmp_roundrobin_config_t
Configure the round robin mode.

LPCMP_CCR1_COUTA_CFG_MASK

LPCMP_CCR1_COUTA_CFG_SHIFT

LPCMP_CCR1_COUTA_CFG(x)

LPCMP_CCR1_EVT_SEL_CFG_MASK

LPCMP_CCR1_EVT_SEL_CFG_SHIFT

LPCMP_CCR1_EVT_SEL_CFG(x)

struct lpcmp_filter_config
#include <fsl_lpcmp.h> Configure the filter.

Public Members

bool enableSample
Decide whether to use the external SAMPLE as a sampling clock input.

uint8_t filterSampleCount

Filter Sample Count. Available range is 1-7; 0 disables the filter.

uint8_t filterSamplePeriod

Filter Sample Period. The divider to the bus clock. Available range is 0-255. The sampling clock must be at least 4 times slower than the system clock to the comparator. So if enableSample is “false”, filterSamplePeriod should be set greater than 4.

struct _lpcmp_dac_config

#include <fsl_lpcmp.h> configure the internal DAC.

Public Members

bool enableLowPowerMode

Decide whether to enable DAC low power mode.

lpcmp_dac_reference_voltage_source_t referenceVoltageSource

Internal DAC supply voltage reference source.

uint8_t DACValue

Value for the DAC Output Voltage. Different devices has different available range, for specific values, please refer to the reference manual.

struct _lpcmp_config

#include <fsl_lpcmp.h> Configures the comparator.

Public Members

bool enableStopMode

Decide whether to enable the comparator when in STOP modes.

bool enableOutputPin

Decide whether to enable the comparator is available in selected pin.

bool useUnfilteredOutput

Decide whether to use unfiltered output.

bool enableInvertOutput

Decide whether to inverts the comparator output.

lpcmp_hysteresis_mode_t hysteresisMode

LPCMP hysteresis mode.

lpcmp_power_mode_t powerMode

LPCMP power mode.

lpcmp_functional_source_clock_t functionalSourceClock

Select LPCMP functional mode clock source.

struct _lpcmp_window_control_config

#include <fsl_lpcmp.h> Configure the window mode control.

Public Members

bool enableInvertWindowSignal

True: enable invert window signal, False: disable invert window signal.

lpcmp_couta_signal_t COUTASignal

Decide whether to define the COUTA signal value when the window is closed.

lpcmp_close_window_event_t closeWindowEvent

Decide whether to select COUT event signal edge defines a COUT event to close window.

struct __lpcmp__roundrobin__config

#include <fsl_lpcmp.h> Configure the round robin mode.

Public Members

uint8_t initDelayModules

Comparator and DAC initialization delay modulus, See Reference Manual and DataSheet for specific value.

uint8_t sampleClockNumbers

Specify the number of the round robin clock cycles(0~3) to wait after scanning the active channel before sampling the channel's comparison result.

uint8_t channelSampleNumbers

Specify the number of samples for one channel, note that channelSampleNumbers must not smaller than sampleTimeThreshhold.

uint8_t sampleTimeThreshhold

Specify that for one channel, when (sampleTimeThreshhold + 1) sample results are "1",the final result is "1", otherwise the final result is "0", note that the sampleTimeThreshhold must not be larger than channelSampleNumbers.

lpcmp_roundrobin_clock_source_t roundrobinClockSource

Decide which clock source to choose in round robin mode.

lpcmp_roundrobin_trigger_source_t roundrobinTriggerSource

Decide which trigger source to choose in round robin mode.

lpcmp_roundrobin_fixedmuxport_t fixedMuxPort

Decide which mux port to choose as fixed channel in round robin mode.

uint8_t fixedChannel

Indicate which channel of the fixed mux port is used in round robin mode.

uint8_t checkerChannelMask

Indicate which channel of the non-fixed mux port to check its voltage value in round robin mode, for example, if checkerChannelMask set to 0x11U means select channel 0 and channel 4 as checker channel.

2.57 LPFLEXCOMM: LPFLEXCOMM Driver

2.58 LPFLEXCOMM Driver

FSL_LP_FLEXCOMM_DRIVER_VERSION

FlexCOMM driver version.

enum LP_FLEXCOMM_PERIPH_T

LP_FLEXCOMM peripheral modes.

Values:

enumerator LP_FLEXCOMM_PERIPH_NONE
No peripheral

enumerator LP_FLEXCOMM_PERIPH_LPUART
LPUART peripheral

enumerator LP_FLEXCOMM_PERIPH_LPSPI
LPSPI Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2C
LPI2C Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2CAndLPUART
LPI2C and LPUART Peripheral

enum _lpflexcomm_interrupt_flag
LP_FLEXCOMM interrupt source flags.

Values:

enumerator kLPFLEXCOMM_I2cSlaveInterruptFlag

enumerator kLPFLEXCOMM_I2cMasterInterruptFlag

enumerator kLPFLEXCOMM_SpiInterruptFlag

enumerator kLPFLEXCOMM_UartRxInterruptFlag

enumerator kLPFLEXCOMM_UartTxInterruptFlag

enumerator kLPFLEXCOMM_AllInterruptFlag

typedef void (*lpflexcomm_irq_handler_t)(uint32_t instance, void *handle)
Typedef for interrupt handler.

IRQn_Type const kFlexcommIrqs[]
Array with IRQ number for each LP_FLEXCOMM module.

uint32_t LP_FLEXCOMM_GetInstance(void *base)
Returns instance number for LP_FLEXCOMM module with given base address.

uint32_t LP_FLEXCOMM_GetBaseAddress(uint32_t instance)
Returns for LP_FLEXCOMM base address.

uint32_t LP_FLEXCOMM_GetInterruptStatus(uint32_t instance)
brief Returns for LP_FLEXCOMM interrupt source,see _lpflexcomm_interrupt_flag.

status_t LP_FLEXCOMM_Init(uint32_t instance, *LP_FLEXCOMM_PERIPH_T* periph)
Initializes LP_FLEXCOMM and selects peripheral mode according to the second parameter.

void LP_FLEXCOMM_Deinit(uint32_t instance)
Deinitializes LP_FLEXCOMM.

void LP_FLEXCOMM_SetIRQHandler(uint32_t instance, *lpflexcomm_irq_handler_t* handler, void *lpflexcommHandle, *LP_FLEXCOMM_PERIPH_T* periph)
Sets IRQ handler for given LP_FLEXCOMM module. It is used by drivers register IRQ handler according to LP_FLEXCOMM mode.

2.59 LPI2C: Low Power Inter-Integrated Circuit Driver

FSL_LPI2C_DRIVER_VERSION
LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy
The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle
The slave driver is idle.

enumerator kStatus_LPI2C_Nak
The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError
FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError
Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost
Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout
SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress
Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail
DMA request failed.

enumerator kStatus_LPI2C_Timeout
Timeout polling status flags.

I2C_RETRY_TIMES
Retry times for waiting flag.

2.60 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)
Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster             = true;
masterConfig->debugEnable              = false;
masterConfig->ignoreAck                = false;
masterConfig->pinConfig                = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz              = 100000U;
masterConfig->busIdleTimeout_ns        = 0;
masterConfig->pinLowTimeout_ns         = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
```

(continues on next page)

(continued from previous page)

```
masterConfig->hostRequest.enable     = false;
masterConfig->hostRequest.source     = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity   = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to lpi2c_master_config_t.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags().

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted,

followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_LPI2C_Nak.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.

- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferHandleIRQ(uint32_t instance, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- instance – The LPI2C instance.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag
Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag
Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag
End Packet flag

enumerator kLPI2C_MasterStopDetectFlag
Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag
NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag
Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag
FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag
Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag
Data match flag

enumerator kLPI2C_MasterBusyFlag
Master busy flag

enumerator kLPI2C_MasterBusBusyFlag
Bus busy flag

enumerator kLPI2C_MasterClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
Errors to check for.

enum _lpi2c_direction
Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write
Master transmit.

enumerator kLPI2C_Read
Master receive.

enum _lpi2c_master_pin_config
LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain
LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Values:

enumerator kLPI2C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag
Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t
Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t
LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t
LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t
LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t
Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t
LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t
LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)
Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base
The LPI2C peripheral base address.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(uint32_t instance, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct _lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster
Whether to enable master mode.

bool enableDoze
Whether master is enabled in doze mode.

bool debugEnable
Enable transfers to continue when halted in debug mode.

bool ignoreAck
Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig
The pin configuration option.

uint32_t baudRate_Hz
Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns
Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns
Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns
Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns
Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest
Host request options.

struct _lpi2c_match_config

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
Data match configuration setting.

bool rxDataMatchOnly
When set to true, received data is ignored until a successful match.

uint32_t match0
Match value 0.

uint32_t match1
Match value 1.

struct __lpi2c_master_transfer
#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
Bit mask of options for the transfer. See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefaultFlag for normal transfers.

uint16_t slaveAddress
The 7-bit slave address.

lpi2c_direction_t direction
Either kLPI2C_Read or kLPI2C_Write.

uint32_t subaddress
Sub address. Transferred MSB first.

size_t subaddressSize
Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
Pointer to data to transfer.

size_t dataSize
Number of bytes to transfer.

struct __lpi2c_master_handle
#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
Transfer state machine current state.

uint16_t remainingBytes
Remaining byte count in current state.

uint8_t *buf
Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

struct hostRequest

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.61 LPI2C Master DMA Driver

void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle, *lpi2c_master_edma_transfer_callback_t* callback, void *userData)

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

typedef struct *_lpi2c_master_edma_handle* lpi2c_master_edma_handle_t

typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *status_t* completionStatus, void *userData)

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Param base
The LPI2C peripheral base address.

Param handle
Handle associated with the completed transfer.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

struct _lpi2c_master_edma_handle

#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base

LPI2C base pointer.

bool isBusy

Transfer state machine current state.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[10]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

edma_handle_t *rx

Handle for receive DMA channel.

edma_handle_t *tx

Handle for transmit DMA channel.

edma_tcd_t tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.62 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave              = true;
slaveConfig->address0                 = 0U;
slaveConfig->address1                 = 0U;
slaveConfig->addressMatchMode         = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable         = true;
slaveConfig->filterEnable             = true;
slaveConfig->enableGeneralCall        = false;
slaveConfig->sclStall.enableAck       = false;
slaveConfig->sclStall.enableTx        = true;
slaveConfig->sclStall.enableRx        = true;
slaveConfig->sclStall.enableAddress   = true;
slaveConfig->ignoreAck                = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns  = 0;
slaveConfig->sclGlitchFilterWidth_ns  = 0;
slaveConfig->dataValidDelay_ns        = 0;
slaveConfig->clockHoldTime_ns         = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t.

void LPI2C_SlaveInit(LPI2C_Type *base, const *lpi2c_slave_config_t* *slaveConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use LPI2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_slave_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_slave_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags().

static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.

- interruptMask – Bit mask of interrupts to enable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the kLPI2C_SlaveTransmitAckFlag is asserted. This only happens if you enable the sclStall.enableAck field of the lpi2c_slave_config_t configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

- ackOrNack – Pass true for an ACK or false for a NAK.

static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL,false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns

Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns

Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of lpi2c_slave_transfer_event_t enumerators for the events you wish to receive. The kLPI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kLPI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together lpi2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kLPI2C_SlaveAllEvents to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_LPI2C_Busy – Slave transfers have already been started on this handle.

status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

Return values

- kStatus_Success –
- kStatus_LPI2C_Idle –

void LPI2C_SlaveTransferHandleIRQ(uint32_t instance, void *lpi2cSlaveHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- instance – The LPI2C instance.
- lpi2cSlaveHandle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag
Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag
Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag
Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag
Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag
Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag
FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag
Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag
Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag
General call flag

enumerator kLPI2C_SlaveBusyFlag
Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag
Bus busy flag

enumerator kLPI2C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags
Errors to check for.

enum _lpi2c_slave_address_match
LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0
Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1
Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1
Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK. When this event is set, the driver will no longer decide to reply to ack/nack.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the LPI2C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct __lpi2c_slave_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave
Enable slave mode.

uint8_t address0
Slave's 7-bit address.

uint8_t address1
Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode
Address matching options.

bool filterDozeEnable
Enable digital glitch filter in doze mode.

bool filterEnable
Enable digital glitch filter.

bool enableGeneralCall
Enable general call address matching.

bool ignoreAck
Continue transfers after a NACK is detected.

bool enableReceivedAddressRead
Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns
Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns
Width in nanoseconds of the clock hold time.

struct __lpi2c_slave_transfer

#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event
Reason the callback is being invoked.

uint8_t receivedAddress
Matching address send by master.

uint8_t *data
Transfer buffer

size_t dataSize
Transfer size

status_t completionStatus
Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount
Number of bytes actually transferred since start or last repeated start.

struct __lpi2c_slave_handle
#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer
LPI2C slave transfer copy.

bool isBusy
Whether transfer is busy.

bool wasTransmit
Whether the last transfer was a transmit.

uint32_t eventMask
Mask of enabled events.

uint32_t transferredCount
Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck
Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

2.63 LPSPI: Low Power Serial Peripheral Interface

2.64 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure lpspi_master_config_t.
- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the lpspi_master_config_t structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to lpspi_master_config_t structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi_slave_config_t.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the lpspi_slave_config_t structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to lpspi_slave_config_t structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns
The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns
The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type _lpspi_flags.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

bool LPSPI_CheckTransferArgument(LPSPI_Type *base, *lpspi_transfer_t* *transfer, bool isEdma)

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, *lpspi_master_slave_mode_t* mode)

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, *lpspi_which_pcs_t* select)

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI_SetPCSContinous(LPSPI_Type *base, bool IsContinous)

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.

- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, *lpspi_delay_type_t* whichDelay)

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.

uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, *lpspi_delay_type_t* whichDelay, uint32_t srcClock_Hz)

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler).

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nano-seconds.
- whichDelay – The desired delay to configuration, which must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.
- data – The data word to be sent.

static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The data read from the data buffer.

void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_master_transfer_callback_t* callback, void *userData)

Initializes the LPSPI master handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_master_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(uint32_t instance, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- instance – LPSPI instance.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(uint32_t instance, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- instance – LPSPI instance index.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy
LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error
LPSPI driver error.

enumerator kStatus_LPSPI_Idle
LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange
LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout
LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag
Transmit data flag

enumerator kLPSPI_RxDataReadyFlag
Receive data flag

enumerator kLPSPI_WordCompleteFlag
Word Complete flag

enumerator kLPSPI_FrameCompleteFlag
Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag
Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag
Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag
Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag
Data Match flag

enumerator kLPSPI_ModuleBusyFlag
Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable
LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable
Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0

PCS[0]

enumerator kLPSPI_Pcs1

PCS[1]

enumerator kLPSPI_Pcs2

PCS[2]

enumerator kLPSPI_Pcs3

PCS[3]

enum _lpspi_pcs_polarity_config

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh

PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow

PCS Active Low (idles high)

enum _lpspi_pcs_polarity

LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow

Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow

Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow

Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow

Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow

Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity

LPSPI clock polarity configuration.

Values:

enumerator kLPSPI_ClockPolarityActiveHigh

CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow

CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase

LPSPI clock phase configuration.

Values:

enumerator kLPSPI_ClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction

LPSPI data shifter direction options.

Values:

enumerator kLPSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kLPSPI_LsbFirst

Data transfers start with least significant bit.

enum _lpspi_host_request_select

LPSPI Host Request select configuration.

Values:

enumerator kLPSPI_HostReqExtPin

Host Request is an ext pin.

enumerator kLPSPI_HostReqInternalTrigger

Host Request is an internal trigger.

enum _lpspi_match_config

LPSPI Match configuration options.

Values:

enumerator kLPSI_MatchDisabled

LPSPI Match Disabled.

enumerator kLPSI_1stWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordEqualsM0and2ndWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0andNxtWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordAndM1EqualsM0andM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1

LPSPI Match Enabled.

enum _lpspi_pin_config

LPSPI pin (SDO and SDI) configuration.

Values:

enumerator kLPSPI_SdiInSdoOut

LPSPI SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut
LPSPI SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut
LPSPI SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut
LPSPI SDO input, SDI output.

enum _lpspi_data_out_config
LPSPI data output configuration.

Values:

enumerator kLpspiDataOutRetained
Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate
Data out is tristated when chip select is de-asserted

enum _lpspi_pcs_function_config
LPSPI cs function configuration.

Values:

enumerator kLPSPI_PcsAsCs
PCS pin select as cs function

enumerator kLPSPI_PcsAsData
PCS pin select as date function

enum _lpspi_transfer_width
LPSPI transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPI delay type selection.

Values:

enumerator kLPSPI_PcsToSck
PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
Delay between transfers.

enum _lpspi_transfer_config_flag_for_master
Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterWidth1

LPSPI master width shift macro, internal used LPSPI master transfer 1bit

enumerator kLPSPI_MasterWidth2

LPSPI master width shift macro, internal used LPSPI master transfer 2bit

enumerator kLPSPI_MasterWidth4

LPSPI master width shift macro, internal used LPSPI master transfer 4bit

enumerator kLPSPI_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPI slave transfer configFlags.

Values:

enumerator kLPSPI_SlavePcs0

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS0 signal

enumerator kLPSPI_SlavePcs1

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS1 signal

enumerator kLPSPI_SlavePcs2

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS2 signal

enumerator kLPSPI_SlavePcs3

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS3 signal

enumerator kLPSPI_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

enum _lpspi_transfer_state

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator kLPSPI_Idle

Nothing in the transmitter/receiver.

enumerator kLPSPI_Busy

Transfer queue is not finished.

enumerator kLPSPI_Error

Transfer error.

typedef enum *_lpspi_master_slave_mode* lpspi_master_slave_mode_t

LPSPI master or slave mode configuration.

typedef enum *_lpspi_which_pcs_config* lpspi_which_pcs_t

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum *_lpspi_pcs_polarity_config* lpspi_pcs_polarity_config_t

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *_lpspi_clock_polarity* lpspi_clock_polarity_t

LPSPI clock polarity configuration.

typedef enum *_lpspi_clock_phase* lpspi_clock_phase_t

LPSPI clock phase configuration.

typedef enum *_lpspi_shift_direction* lpspi_shift_direction_t

LPSPI data shifter direction options.

typedef enum *_lpspi_host_request_select* lpspi_host_request_select_t

LPSPI Host Request select configuration.

typedef enum *_lpspi_match_config* lpspi_match_config_t

LPSPI Match configuration options.

typedef enum *_lpspi_pin_config* lpspi_pin_config_t

LPSPI pin (SDO and SDI) configuration.

typedef enum *_lpspi_data_out_config* lpspi_data_out_config_t

LPSPI data output configuration.

typedef enum *_lpspi_pcs_function_config* lpspi_pcs_function_config_t

LPSPI cs function configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t

LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t

LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t

LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t

LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t

Forward declaration of the _lpspi_master_handle typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t

Forward declaration of the _lpspi_slave_handle typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)

Master completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)

Slave completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_lpspi_transfer* lpspi_transfer_t

LPSPI master/slave transfer structure.

volatile uint8_t g_lpspiDummyData[]

Global variable for dummy data value setting.

LPSPI_DUMMY_DATA

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

SPI_RETRY_TIMES

Retry times for waiting flag.

LPSPI_MASTER_PCS_SHIFT

LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_PCS_MASK

LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_WIDTH_SHIFT

LPSPI master width shift macro, internal used

LPSPI_MASTER_WIDTH_MASK
LPSPI master width shift mask, internal used

LPSPI_SLAVE_PCS_SHIFT
LPSPI slave PCS shift macro , internal used.

LPSPI_SLAVE_PCS_MASK
LPSPI slave PCS shift macro , internal used.

struct _lpspi_master_config
#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate
Baud Rate for LPSPI.

uint32_t bitsPerFrame
Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
Clock polarity.

lpspi_clock_phase_t cpha
Clock phase.

lpspi_shift_direction_t direction
MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec
PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec
Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec
After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs
Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
Desired PCS active high or low

lpspi_pin_config_t pinCfg
Configures which pins are used for input and output data during single bit transfers.

lpspi_pcs_function_config_t pcsFunc
Configures cs pins function.

lpspi_data_out_config_t dataOutConfig
Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay
Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct _lpspi_slave_config
#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame
: Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
: Clock polarity.

lpspi_clock_phase_t cpha
: Clock phase.

lpspi_shift_direction_t direction
: MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs
: Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
: Desired PCS active high or low

lpspi_pin_config_t pinCfg
: Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
: Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct _lpspi_transfer
: *#include <fsl_lpspi.h>* LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData
: Send buffer.

uint8_t *rxData
: Receive buffer.

volatile size_t dataSize
: Transfer bytes.

uint32_t configFlags
: Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

struct _lpspi_master_handle
: *#include <fsl_lpspi.h>* LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
: Is PCS continuous in transfer.

volatile bool writeTcrInIsr
: A flag that whether should write TCR in ISR.

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile bool isTxMask
A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame
Number of bytes in each frame

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

uint32_t txBuffIfNull
Used if the txData is NULL.

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

struct __lpspi_slave_handle
#include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
A flag that whether should byte swap.

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
: Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

2.65 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION
: LPSPI EDMA driver version.

typedef struct *_lpspi_master_edma_handle* lpspi_master_edma_handle_t
: Forward declaration of the _lpspi_master_edma_handle typedefs.

typedef struct *_lpspi_slave_edma_handle* lpspi_slave_edma_handle_t
: Forward declaration of the _lpspi_slave_edma_handle typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)
: Completion callback function pointer type.

  Param base
  : LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size

should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- configFlags – transfer configuration flags. _lpspi_transfer_config_flag_for_master.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

Returns

Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure, config field is not uesed.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

- kStatus_InvalidArgument – The transfer structure is invalid.

Returns
Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, size_t *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_slave_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.

- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle)

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.

status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, size_t *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns

status of status_t.

struct __lpspi_master_edma_handle

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool isByteSwap
: A flag that whether should byte swap.

uint32_t totalByteCount
: Number of transfer bytes

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

uint32_t transmitCommand
: Used to write TCR for DMA purpose.

volatile int8_t oneFifoBlockRxWatermark
: Used to change RXWATER in FCR for DMA purpose.

volatile int8_t lastBytesRxWatermark
: Used to change RXWATER in FCR for DMA purpose.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[6]
: SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
: *#include <fsl_lpspi_edma.h>* LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

uint32_t totalByteCount
: Number of transfer bytes

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

volatile int8_t oneFifoBlockRxWatermark
Used to change RXWATER in FCR for DMA purpose.

volatile int8_t lastBytesRxWatermark
Used to change RXWATER in FCR for DMA purpose.

volatile uint8_t state
LPSPI transfer state.

lpspi_slave_edma_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg

edma_tcd_t lpspiSoftwareTCD[5]
SoftwareTCD, internal used

2.66 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

2.67 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.68 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

void LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, flase to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in lpuart_handle_t.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators _lpuart_flags. To check for a specific status, compare the return value with enumerators in the _lpuart_flags. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorInRxDataRegFlag, kLPUART_ParityErrorInRxDataRegFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of the enumeration _uart_interrupt_enable.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

static inline uint32_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.

- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

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

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferStartRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART_TransferStopRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, *lpuart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_LPUART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(uint32_t instance, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- instance – LPUART instance.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually
Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select
LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_CtsStateChangeInterruptEnable
Change of state on CTS_B pin. bit 0

enumerator kLPUART_DsrStateChangeInterruptEnable
Change of state on DSR_B pin. bit 1

enumerator kLPUART_RinStateChangeInterruptEnable
Change of state on RIN_B pin. bit 2

enumerator kLPUART_DcdStateChangeInterruptEnable
Change of state on DCD_B pin. bit 3

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxCounter0TimeoutInterruptEnable
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutInterruptEnable
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutInterruptEnable
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutInterruptEnable
Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2InterruptEnable
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1InterruptEnable
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_CtsStateChangeFlag
Change of state on CTS_B pin. bit 2

enumerator kLPUART_DsrStateChangeFlag
Change of state on DSR_B pin. bit 3

enumerator kLPUART_RinStateChangeFlag
Change of state on RIN_B pin. bit 4

enumerator kLPUART_DcdStateChangeFlag
Change of state on DCD_B pin. bit 5

enumerator kLPUART_RxCounter0TimeoutFlag
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutFlag
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutFlag
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutFlag
Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_TxDataRegEmptyFlag

Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

enum _lpuart_timeout_condition

LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

Values:

enumerator kLPUART_TimeoutAfterCharacters

Timeout occurs when the number of characters specified by timeoutValue are received.

enumerator kLPUART_TimeoutAfterIdle

Timeout occurs when rx/tx remains idle for timeoutValue of bit clocks after idle condition is detected.

enumerator kLPUART_TimeoutAfterNext

Timeout occurs when rx/tx remains idle for timeoutValue of bit clocks after next character is received/transmitted.

enumerator kLPUART_TimeoutAfterIdleBeforeExtended

Timeout occurs when tx/rx is idle for larger than timeoutValue of bit clocks and smaller than tx/rx extended timeout value.

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
LPUART transmit CTS configure.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef enum *_lpuart_timeout_condition* lpuart_timeout_condition_t
LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

typedef struct *_lpuart_timeout_counter_config* lpuart_timeout_counter_config_t
LPUART timeout counter configuration structure.

typedef struct *_lpuart_timeout_config* lpuart_timeout_config_t
LPUART timeout configuration structure.

typedef struct *_lpuart_config* lpuart_config_t
LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
LPUART transfer callback function.

typedef void (*lpuart_irq_handler_t)(uint32_t instance, void *handle)

const IRQn_Type s_lpuartIRQ[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_timeout_counter_config
#include <fsl_lpuart.h> LPUART timeout counter configuration structure.

Public Members

bool enableCounter
Eneble the timeout counter.

lpuart_timeout_condition_t timeoutCondition
Timeout condition.

uint16_t timeoutValue
: Timeout value.

struct __lpuart__timeout__config
: *#include <fsl_lpuart.h>* LPUART timeout configuration structure.

Public Members

uint16_t rxExtendedTimeoutValue
: The number of bits since the last stop bit that is required for an idle condition to be detected. Enable this will disable rxIdleType and rxIdleConfig. Set to 0 to disable.

uint16_t txExtendedTimeoutValue
: The transmitter idle time in number of bits (baud rate) whenever an idle character is queued through the transmit FIFO.

lpuart_timeout_counter_config_t rxCounter0
: Rx counter 0 configuration.

lpuart_timeout_counter_config_t rxCounter1
: Rx counter 1 configuration.

lpuart_timeout_counter_config_t txCounter0
: Tx counter 0 configuration.

lpuart_timeout_counter_config_t txCounter1
: Tx counter 1 configuration.

struct __lpuart__config
: *#include <fsl_lpuart.h>* LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
: LPUART baud rate

lpuart_parity_mode_t parityMode
: Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
: Data bits count, eight (default), seven

bool isMsb
: Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
: Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
: TX FIFO watermark

uint8_t rxFifoWatermark
: RX FIFO watermark

bool enableRxRTS
: RX RTS enable

bool enableTxCTS
: TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

lpuart_timeout_config_t timeoutConfig
Timeout configuration.

bool enableSingleWire
Use TXD pin as the source for the receiver. When enabled the TXD pin should be configured as open drain.

uint8_t rtsDelay
Delay the negation of RTS by the configured number of bit clocks.

bool enableTx
Enable TX

bool enableRx
Enable RX

struct __lpuart__transfer
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __lpuart__handle
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

volatile size_t txDataSize
Size of the remaining data to send.

size_t txDataSizeAll
Size of the data to send out.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t rxDataSizeAll
Size of the data to receive.

size_t rxRingBufferSize
Size of the ring buffer.

volatile uint16_t rxRingBufferHead
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
: Callback function.

void *userData
: LPUART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state.

bool isSevenDataBits
: Seven data bits flag.

bool is16bitData
: 16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed166___

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 ___unnamed168___

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 ___unnamed170___

Public Members

uint8_t *volatile rxData
: Address of remaining data to receive.

uint16_t *volatile rxData16
: Address of remaining data to receive.

union ___unnamed172___

Public Members

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16

Start address of the receiver ring buffer.

2.69 LPUART eDMA Driver

void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

status_t LPUART_SendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferEdmaHandleIRQ(uint32_t instance, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- instance – LPUART peripheral index.
- lpuartEdmaHandle – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

typedef struct *_lpuart_edma_handle* lpuart_edma_handle_t

typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

struct _lpuart_edma_handle

#include <fsl_lpuart_edma.h> LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback

Callback function.

void *userData

LPUART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.70 Mailbox

static inline void MAILBOX_Init(MAILBOX_Type *base)

Initializes the MAILBOX module.

This function enables the MAILBOX clock only.

Parameters

- base – MAILBOX peripheral base address.

static inline void MAILBOX_Deinit(MAILBOX_Type *base)

De-initializes the MAILBOX module.

This function disables the MAILBOX clock only.

Parameters

- base – MAILBOX peripheral base address.

FSL_MAILBOX_DRIVER_VERSION

MAILBOX driver version.

static inline uint32_t MAILBOX_GetMutex(MAILBOX_Type *base)

Get MUTEX state and lock mutex.

Note: Returns '1' if the mutex was taken or '0' if another resources has the mutex locked. Once a mutex is taken, it can be returned with the MAILBOX_SetMutex() function.

Parameters

- base – MAILBOX peripheral base address.

Returns

See note

static inline void MAILBOX_SetMutex(MAILBOX_Type *base)

Set MUTEX state.

Note: Sets mutex state to '1' and allows other resources to get the mutex.

Parameters

- base – MAILBOX peripheral base address.

FSL_COMPONENT_ID

2.71 MCX_CMC: Core Mode Controller Driver

enum __cmc_power_mode_protection

CMC power mode Protection enumeration.

Values:

enumerator kCMC_AllowDeepSleepMode

Allow Deep Sleep mode.

enumerator kCMC_AllowPowerDownMode
Allow Power Down mode.

enumerator kCMC_AllowDeepPowerDownMode
Allow Deep Power Down mode.

enumerator kCMC_AllowAllLowPowerModes
Allow Deep Sleep, Power Down, Deep Power Down modes.

enum _cmc_wakeup_sources
Wake up sources from the previous low power mode entry.

Note: kCMC_WakeupFromUsbFs, kCMC_WakeupFromITRC, kCMC_WakeupFromCpu1 are not supported in MCXA family.

Values:

enumerator kCMC_WakeupFromResetInterruptOrPowerDown
Wakeup source is reset interrupt, or wake up from Deep Power Down.

enumerator kCMC_WakeupFromDebugReuqest
Wakeup source is debug request.

enumerator kCMC_WakeupFromInterrupt
Wakeup source is interrupt.

enumerator kCMC_WakeupFromDMAWakeup
Wakeup source is DMA Wakeup.

enumerator kCMC_WakeupFromWUURequest
Wakeup source is WUU request.

enumerator kCMC_WakeupFromUsbFs
Wakeup source is USBFS(USB0).

enumerator kCMC_WakeupFromITRC
Wakeup source is ITRC.

enumerator kCMC_WakeupFromCpu1
Wakeup source is CPU1.

enum _cmc_system_reset_interrupt_enable
System Reset Interrupt enable enumeration.

Values:

enumerator kCMC_PinResetInterruptEnable
Pin Reset interrupt enable.

enumerator kCMC_DAPResetInterruptEnable
DAP Reset interrupt enable.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetInterruptEnable
Low Power Acknowledge Timeout Reset interrupt enable.

enumerator kCMC_SoftwareResetInterruptEnable
Software Reset interrupt enable.

enumerator kCMC_LockupResetInterruptEnable
Lockup Reset interrupt enable.

enum _cmc_system_reset_interrupt_flag

CMC System Reset Interrupt Status flag.

Values:

enumerator kCMC_PinResetInterruptFlag

Pin Reset interrupt flag.

enumerator kCMC_DAPResetInterruptFlag

DAP Reset interrupt flag.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetFlag

Low Power Acknowledge Timeout Reset interrupt flag.

enumerator kCMC_SoftwareResetInterruptFlag

Software Reset interrupt flag.

enumerator kCMC_LockupResetInterruptFlag

Lock up Reset interrupt flag.

enum _cmc_system_sram_arrays

CMC System SRAM arrays low power mode enable enumeration.

Values:

enumerator kCMC_RAMX0

Used to control RAMX0.

enumerator kCMC_RAMX1

Used to control RAMX1.

enumerator kCMC_RAMX2

Used to control RAMX2.

enumerator kCMC_RAMB

Used to control RAMB.

enumerator kCMC_RAMC0

Used to control RAMC0.

enumerator kCMC_RAMC1

Used to control RAMC1.

enumerator kCMC_RAMD0

Used to control RAMD0.

enumerator kCMC_RAMD1

Used to control RAMD1.

enumerator kCMC_RAME0

Used to control RAME0.

enumerator kCMC_RAME1

Used to control RAME1.

enumerator kCMC_RAMF0

Used to control RAMF0.

enumerator kCMC_RAMF1

Used to control RAMF1.

enumerator kCMC_RAMG0_RAMG1

Used to control RAMG0 and RAMG1.

enumerator kCMC_RAMG2_RAMG3
Used to control RAMG2 and RAMG3.

enumerator kCMC_RAMH0_RAMH1
Used to control RAMH0 and RAMH1.

enumerator kCMC_LPCAC
Used to control LPCAC.

enumerator kCMC_DMA0_DMA1_PKC
Used to control DMA0, DMA1 and PKC.

enumerator kCMC_USB0
Used to control USB0.

enumerator kCMC_PQ
Used to control PQ.

enumerator kCMC_CAN0_CAN1_ENET_USB1
Used to control CAN0, CAN1, ENET, USB1.

enumerator kCMC_FlexSPI
Used to control FlexSPI.

enumerator kCMC_AllSramArrays
Mask of all System SRAM arrays.

enum _cmc_system_reset_sources
System reset sources enumeration.

Values:

enumerator kCMC_WakeUpReset
The reset caused by a wakeup from Power Down or Deep Power Down mode.

enumerator kCMC_PORReset
The reset caused by power on reset detection logic.

enumerator kCMC_WarmReset
The last reset source is a warm reset source.

enumerator kCMC_FatalReset
The last reset source is a fatal reset source.

enumerator kCMC_PinReset
The reset caused by the RESET_b pin.

enumerator kCMC_DAPReset
The reset caused by a reset request from the Debug Access port.

enumerator kCMC_ResetTimeout
The reset caused by a timeout or other error condition in the system reset generation.

enumerator kCMC_LowPowerAcknowledgeTimeoutReset
The reset caused by a timeout in low power mode entry logic.

enumerator kCMC_SCGReset
The reset caused by a loss of clock or loss of lock event in the SCG.

enumerator kCMC_SoftwareReset
The reset caused by a software reset request.

enumerator kCMC_LockUoReset
The reset caused by the ARM core indication of a LOCKUP event.

enumerator kCMC_JTAGSystemReset
The reset caused by a JTAG system reset request.

enum __cmc_core_clock_gate_status
Indicate the core clock was gated.

Values:

enumerator kCMC_CoreClockNotGated
Core clock not gated.

enumerator kCMC_CoreClockGated
Core clock was gated due to low power mode entry.

enum __cmc_clock_mode
CMC clock mode enumeration.

Values:

enumerator kCMC_GateNoneClock
No clock gating.

enumerator kCMC_GateCoreClock
Gate Core clock.

enumerator kCMC_GateCorePlatformClock
Gate Core clock and platform clock.

enumerator kCMC_GateAllSystemClocks
Gate all System clocks, without getting core entering into low power mode.

enumerator kCMC_GateAllSystemClocksEnterLowPowerMode
Gate all System clocks, with core entering into low power mode.

enum __cmc_low_power_mode
CMC power mode enumeration.

Values:

enumerator kCMC_ActiveOrSleepMode
Select Active/Sleep mode.

enumerator kCMC_DeepSleepMode
Select Deep Sleep mode when a core executes WFI or WFE instruction.

enumerator kCMC_PowerDownMode
Select Power Down mode when a core executes WFI or WFE instruction.

enumerator kCMC_DeepPowerDown
Select Deep Power Down mode when a core executes WFI or WFE instruction.

typedef enum *_cmc_core_clock_gate_status* cmc_core_clock_gate_status_t
Indicate the core clock was gated.

typedef enum *_cmc_clock_mode* cmc_clock_mode_t
CMC clock mode enumeration.

typedef enum *_cmc_low_power_mode* cmc_low_power_mode_t
CMC power mode enumeration.

typedef struct *_cmc_reset_pin_config* cmc_reset_pin_config_t

CMC reset pin configuration.

typedef struct *_cmc_power_domain_config* cmc_power_domain_config_t

power mode configuration for each power domain.

FSL_CMC_DRIVER_VERSION

CMC driver version 2.5.0.

void CMC_SetClockMode(CMC_Type *base, *cmc_clock_mode_t* mode)

Sets clock mode.

This function configs the amount of clock gating when the core asserts Sleeping due to WFI, WFE or SLEEPONEXIT.

Parameters

- base – CMC peripheral base address.
- mode – System clock mode.

static inline void CMC_LockClockModeSetting(CMC_Type *base)

Locks the clock mode setting.

After invoking this function, any clock mode setting will be blocked.

Parameters

- base – CMC peripheral base address.

static inline *cmc_core_clock_gate_status_t* CMC_GetCoreClockGatedStatus(CMC_Type *base)

Gets the core clock gated status.

This function get the status to indicate whether the core clock is gated. The core clock gated status can be cleared by software.

Parameters

- base – CMC peripheral base address.

Returns

The status to indicate whether the core clock is gated.

static inline void CMC_ClearCoreClockGatedStatus(CMC_Type *base)

Clears the core clock gated status.

This function clear clock status flag by software.

Parameters

- base – CMC peripheral base address.

static inline uint8_t CMC_GetWakeupSource(CMC_Type *base)

Gets the Wakeup Source.

This function gets the Wakeup sources from the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Wakeup sources from the previous low power mode entry. See _cmc_wakeup_sources for details.

static inline *cmc_clock_mode_t* CMC_GetClockMode(CMC_Type *base)

Gets the Clock mode.

This function gets the clock mode of the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Low Power status.

static inline uint32_t CMC_GetSystemResetStatus(CMC_Type *base)

Gets the System reset status.

This function returns the system reset status. Those status updates on every MAIN Warm Reset to indicate the type/source of the most recent reset.

Parameters

- base – CMC peripheral base address.

Returns

The most recent system reset status. See _cmc_system_reset_sources for details.

static inline uint32_t CMC_GetStickySystemResetStatus(CMC_Type *base)

Gets the sticky system reset status since the last WAKE Cold Reset.

This function gets all source of system reset that have generated a system reset since the last WAKE Cold Reset, and that have not been cleared by software.

Parameters

- base – CMC peripheral base address.

Returns

System reset status that have not been cleared by software. See _cmc_system_reset_sources for details.

static inline void CMC_ClearStickySystemResetStatus(CMC_Type *base, uint32_t mask)

Clears the sticky system reset status flags.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the sticky system reset status to be cleared.

static inline uint8_t CMC_GetResetCount(CMC_Type *base)

Gets the number of reset sequences completed since the last Cold Reset.

Parameters

- base – CMC peripheral base address.

Returns

The number of reset sequences.

void CMC_SetPowerModeProtection(CMC_Type *base, uint32_t allowedModes)

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes. This should be done before set the lowPower mode for each power doamin.

The allowed lowpower modes are passed as bit map. For example, to allow Sleep and DeepSleep, use CMC_SetPowerModeProtection(CMC_base, kCMC_AllowSleepMode|kCMC_AllowDeepSleepMode). To allow all low power modes, use CMC_SetPowerModeProtection(CMC_base, kCMC_AllowAllLowPowerModes).

Parameters

- base – CMC peripheral base address.

- allowedModes – Bitmaps of the allowed power modes. See _cmc_power_mode_protection for details.

static inline void CMC_LockPowerModeProtectionSetting(CMC_Type *base)

Locks the power mode protection.

This function locks the power mode protection. After invoking this function, any power mode protection setting will be ignored.

Parameters

- base – CMC peripheral base address.

static inline void CMC_SetGlobalPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Config the same lowPower mode for all power domain.

This function configures the same low power mode for MAIN power domian and WAKE power domain.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See cmc_low_power_mode_t for details.

static inline void CMC_SetMAINPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Configures entry into low power mode for the MAIN Power domain.

This function configures the low power mode for the MAIN power domian, when the core executes WFI/WFE instruction. The available lowPower modes are defined in the cmc_low_power_mode_t.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See cmc_low_power_mode_t for details.

static inline *cmc_low_power_mode_t* CMC_GetMAINPowerMode(CMC_Type *base)

Gets the power mode of the MAIN Power domain.

Parameters

- base – CMC peripheral base address.

Returns

The power mode of MAIN Power domain. See cmc_low_power_mode_t for details.

void CMC_ConfigResetPin(CMC_Type *base, const *cmc_reset_pin_config_t* *config)

Configure reset pin.

This function configures reset pin. When enabled, the low power filter is enabled in both Active and Low power modes, the reset filter is only enabled in Active mode. When both filers are enabled, they operate in series.

Parameters

- base – CMC peripheral base address.
- config – Pointer to the reset pin config structure.

static inline void CMC_EnableSystemResetInterrupt(CMC_Type *base, uint32_t mask)

Enable system reset interrupts.

This function enables the system reset interrupts. The assertion of non-fatal warm reset can be delayed for 258 cycles of the 32K_CLK clock while an enabled interrupt is generated. Then Software can perform a graceful shutdown or abort the non-fatal warm reset provided the pending reset source is cleared by resetting the reset source and then clearing the pending flag.

Parameters

- base – CMC peripheral base address.
- mask – System reset interrupts. See _cmc_system_reset_interrupt_enable for details.

static inline void CMC_DisableSystemResetInterrupt(CMC_Type *base, uint32_t mask)

Disable system reset interrupts.

This function disables the system reset interrupts.

Parameters

- base – CMC peripheral base address.
- mask – System reset interrupts. See _cmc_system_reset_interrupt_enable for details.

static inline uint32_t CMC_GetSystemResetInterruptFlags(CMC_Type *base)

Gets System Reset interrupt flags.

This function returns the System reset interrupt flags.

Parameters

- base – CMC peripheral base address.

Returns

System reset interrupt flags. See _cmc_system_reset_interrupt_flag for details.

static inline void CMC_ClearSystemResetInterruptFlags(CMC_Type *base, uint32_t mask)

Clears System Reset interrupt flags.

This function clears system reset interrupt flags. The pending reset source can be cleared by resetting the source of the reset and then clearing the pending flags.

Parameters

- base – CMC peripheral base address.
- mask – System Reset interrupt flags. See _cmc_system_reset_interrupt_flag for details.

static inline void CMC_EnableNonMaskablePinInterrupt(CMC_Type *base, bool enable)

Enable/Disable Non maskable Pin interrupt.

Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Non maskable pin interrupt. true - enable Non-maskable pin interrupt. false - disable Non-maskable pin interupt.

static inline uint8_t CMC_GetISPMODEPinLogic(CMC_Type *base)

Gets the logic state of the ISPMODE_n pin.

This function returns the logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

Parameters

- base – CMC peripheral base address.

Returns
The logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

static inline void CMC_ClearISPMODEPinLogic(CMC_Type *base)

Clears ISPMODE_n pin state.

Parameters

- base – CMC peripheral base address.

static inline void CMC_ForceBootConfiguration(CMC_Type *base, bool assert)

Set the logic state of the BOOT_CONFIGn pin.

This function force the logic state of the Boot_Confign pin to assert on next system reset.

Parameters

- base – CMC peripheral base address.
- assert – Assert the corresponding pin or not. true - Assert corresponding pin on next system reset. false - No effect.

static inline uint32_t CMC_GetBootRomStatus(CMC_Type *base)

Gets the status information written by the BootROM.

Parameters

- base – CMC peripheral base address.

Returns
The status information written by the BootROM.

static inline void CMC_SetBootRomStatus(CMC_Type *base, uint32_t statValue)

Sets the bootROM status value.

Note: This function is useful when result of CMC_CheckBootRomRegisterWrittable() is true.

Parameters

- base – CMC peripheral base address.
- stat – The state value to set.

static inline bool CMC_CheckBootRomRegisterWrittable(CMC_Type *base)

Check if BootROM status and lock registers is writtable.

Parameters

- base – CMC peripheral base address.

Returns
The result of whether BootROM status and lock register is writtable.

- **true** BootROM status and lock registers are writtable;
- **false** BootROM status and lock registers are not writtable.

static inline void CMC_LockBootRomStatusWritten(CMC_Type *base)

After invoking this function, BootROM status and lock registers cannot be written.

Parameters

- base – CMC peripheral base address.

static inline void CMC_UnlockBootRomStatusWritten(CMC_Type *base)

After invoking this function, BootROM status and lock register can be written.s.

Parameters

- base –

void CMC_PowerOffSRAMAllMode(CMC_Type *base, uint32_t mask)

Power off the selected system SRAM always.

Note: This function power off the selected system SRAM always. The SRAM arrays should not be accessed while they are shut down. SRAM array contents are not retained if they are powered off.

Note: Once invoked, the previous settings will be overwritten.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be powered off all modes. See _cmc_system_sram_arrays for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC_PowerOnSRAMAllMode(CMC_Type *base, uint32_t mask)

Power on SRAM during all mode.

Note: Once invoked, the previous settings will be overwritten.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be powered on all modes. See _cmc_system_sram_arrays for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC_PowerOffSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)

Power off the selected system SRAM during low power modes only.

This function power off the selected system SRAM only during low power mode. SRAM array contents are not retained if they are power off.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be power off during low power mode only. See _cmc_system_sram_arrays for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC_PowerOnSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)

Power on the selected system SRAM during low power modes only.

This function power on the selected system SRAM. The SRAM arrray contents are retained in low power modes.

Parameters

- base – CMC peripheral base address.

- mask – Bitmap of the SRAM arrays to be power on during low power mode only. See _cmc_system_sram_arrays for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC_ConfigFlashMode(CMC_Type *base, bool doze, bool disable)

Configs the low power mode of the on-chip flash memory.

This function configs the low power mode of the on-chip flash memory.

Parameters

- base – CMC peripheral base address.
- doze – true: Flash is disabled while core is sleeping false: No effect.
- disable – true: Flash memory is placed in low power state. false: No effect.

static inline void CMC_EnableDebugOperation(CMC_Type *base, bool enable)

Enables/Disables debug Operation when the core sleep.

This function configs what happens to debug when core sleeps.

Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Debug when Core is sleeping. true - Debug remains enabled when the core is sleeping. false - Debug is disabled when the core is sleeping.

void CMC_PreEnterLowPowerMode(void)

Prepares to enter low power modes.

This function should be called before entering low power modes.

void CMC_PostExitLowPowerMode(void)

Recovers after wake up from low power modes.

This function should be called after wake up from low power modes. This function should be used with CMC_PreEnterLowPowerMode()

void CMC_GlobalEnterLowPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Configs the entry into the same low power mode for each power domains.

This function provides the feature to entry into the same low power mode for each power domains. Before invoking this function, please ensure the selected power mode have been allowed.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The low power mode to be entered. See cmc_low_power_mode_t for the details.

void CMC_EnterLowPowerMode(CMC_Type *base, const *cmc_power_domain_config_t* *config)

Configs the entry into different low power modes for each power domains.

This function provides the feature to entry into different low power modes for each power domains. Before invoking this function please ensure the selected modes are allowed.

Parameters

- base – CMC peripheral base address.
- config – Pointer to the cmc_power_domain_config_t structure.

bool lowpowerFilterEnable
: Low Power Filter enable.

bool resetFilterEnable
: Reset Filter enable.

uint8_t resetFilterWidth
: Width of the Reset Filter.

cmc_clock_mode_t clock_mode
: Clock mode for each power domain.

cmc_low_power_mode_t main_domain
: The low power mode of the MAIN power domain.

struct _cmc_reset_pin_config
: *#include <fsl_cmc.h>* CMC reset pin configuration.

struct _cmc_power_domain_config
: *#include <fsl_cmc.h>* power mode configuration for each power domain.

2.72 ENET: Ethernet Driver

2.73 Mcx_enet

Defines the status return codes for transaction.

Values:

enumerator kStatus_ENET_InitMemoryFail
: Status code 4000. Init failed since buffer memory was not enough.

enumerator kStatus_ENET_RxFrameError
: Status code 4001. A frame received but data error occurred.

enumerator kStatus_ENET_RxFrameFail
: Status code 4002. Failed to receive a frame.

enumerator kStatus_ENET_RxFrameEmpty
: Status code 4003. No frame arrived.

enumerator kStatus_ENET_RxFrameDrop
: Status code 4004. Rx frame was dropped since there's no buffer memory.

enumerator kStatus_ENET_TxFrameBusy
: Status code 4005. There were no resources for Tx operation.

enumerator kStatus_ENET_TxFrameFail
: Status code 4006. Transmit frame failed.

enumerator kStatus_ENET_TxFrameOverLen
: Status code 4007. Failed to send an oversize frame.

enum _enet_mii_mode
: Defines the MII/RMII mode for data interface between the MAC and the PHY.

Values:

enumerator kENET_MiiMode
MII mode for data interface.

enumerator kENET_RmiiMode
RMII mode for data interface.

enum _enet_mii_speed
Defines the 10/100 Mbps speed for the MII data interface.

Values:

enumerator kENET_MiiSpeed10M
Speed 10 Mbps.

enumerator kENET_MiiSpeed100M
Speed 100 Mbps.

enum _enet_mii_duplex
Defines the half or full duplex for the MII data interface.

Values:

enumerator kENET_MiiHalfDuplex
Half duplex mode.

enumerator kENET_MiiFullDuplex
Full duplex mode.

enum _enet_dma_burstlen
Define the DMA maximum transmit burst length.

Values:

enumerator kENET_BurstLen1
DMA burst length 1.

enumerator kENET_BurstLen2
DMA burst length 2.

enumerator kENET_BurstLen4
DMA burst length 4.

enumerator kENET_BurstLen8
DMA burst length 8.

enumerator kENET_BurstLen16
DMA burst length 16.

enumerator kENET_BurstLen32
DMA burst length 32.

enumerator kENET_BurstLen64
DMA burst length 64. eight times enabled.

enumerator kENET_BurstLen128
DMA burst length 128. eight times enabled.

enumerator kENET_BurstLen256
DMA burst length 256. eight times enabled.

enum _enet_desc_flag
Define the flag for the descriptor.

Values:

enumerator kENET_MiddleFlag
It's a middle descriptor of the frame.

enumerator kENET_LastFlagOnly
It's the last descriptor of the frame.

enumerator kENET_FirstFlagOnly
It's the first descriptor of the frame.

enumerator kENET_FirstLastFlag
It's the first and last descriptor of the frame.

enum _enet_systime_op
Define the system time adjust operation control.

Values:

enumerator kENET_SystimeAdd
System time add to.

enumerator kENET_SystimeSubtract
System time subtract.

enum _enet_ts_rollover_type
Define the system time rollover control.

Values:

enumerator kENET_BinaryRollover
System time binary rollover.

enumerator kENET_DigitalRollover
System time digital rollover.

enum _enet_special_config
Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, these is no need to set this control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the enet_config_t.

Note: "kENET_StoreAndForward" is recommended to be set when the ENET_PTP1588FEATURE_REQUIRED is defined or else the timestamp will be mess-up when the overflow happens.

Values:

enumerator kENET_DescDoubleBuffer
The double buffer is used in the Tx/Rx descriptor.

enumerator kENET_StoreAndForward
The Rx/Tx store and forward enable.

enumerator kENET_PromiscuousEnable
The promiscuous enabled.

enumerator kENET_FlowControlEnable
The flow control enabled.

enumerator kENET_BroadCastRxDisable
The broadcast disabled.

enumerator kENET_MulticastAllEnable
All multicast are passed.

enumerator kENET_8023AS2KPacket
8023as support for 2K packets.

enumerator kENET_RxChecksumOffloadEnable
The Rx checksum offload enabled.

enum _enet_dma_interrupt_enable
List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

Values:

enumerator kENET_DmaTx
Tx interrupt.

enumerator kENET_DmaTxStop
Tx stop interrupt.

enumerator kENET_DmaTxBuffUnavail
Tx buffer unavailable.

enumerator kENET_DmaRx
Rx interrupt.

enumerator kENET_DmaRxBuffUnavail
Rx buffer unavailable.

enumerator kENET_DmaRxStop
Rx stop.

enumerator kENET_DmaRxWatchdogTimeout
Rx watchdog timeout.

enumerator kENET_DmaEarlyTx
Early transmit.

enumerator kENET_DmaEarlyRx
Early receive.

enumerator kENET_DmaBusErr
Fatal bus error.

enum _enet_mac_interrupt_enable
List of mac interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

Values:

enumerator kENET_MacPmt

enumerator kENET_MacTimestamp

enum _enet_event
Defines the common interrupt event for callback use.

Values:

enumerator kENET_RxIntEvent
Receive interrupt event.

enumerator kENET_TxIntEvent
Transmit interrupt event.

enumerator kENET_WakeUpIntEvent
Wake up interrupt event.

enumerator kENET_TimeStampIntEvent
Time stamp interrupt event.

enum _enet_dma_tx_sche
Define the DMA transmit arbitration for multi-queue.

Values:

enumerator kENET_FixPri
Fixed priority. channel 0 has lower priority than channel 1.

enumerator kENET_WeightStrPri
Weighted(burst length) strict priority.

enumerator kENET_WeightRoundRobin
Weighted (weight factor) round robin.

enum _enet_mtl_multiqueue_txsche
Define the MTL Tx scheduling algorithm for multiple queues/rings.

Values:

enumerator kENET_txWeightRR
Tx weight round-robin.

enumerator kENET_txStrPrio
Tx strict priority.

enum _enet_mtl_multiqueue_rxsche
Define the MTL Rx scheduling algorithm for multiple queues/rings.

Values:

enumerator kENET_rxStrPrio
Tx weight round-robin, Rx strict priority.

enumerator kENET_rxWeightStrPrio
Tx strict priority, Rx weight strict priority.

enum _enet_mtl_rxqueuemap
Define the MTL Rx queue and DMA channel mapping.

Values:

enumerator kENET_StaticDirctMap
The received fame in Rx Qn(n = 0,1) direclty map to dma channel n.

enumerator kENET_DynamicMap
The received frame in Rx Qn(n = 0,1) map to the dma channel m(m = 0,1) related with the same Mac.

enum _enet_ptp_event_type
Defines the ENET PTP message related constant.

Values:

enumerator kENET_PtpEventMsgType
PTP event message type.

enumerator kENET_PtpSrcPortIdLen
PTP message sequence id length.

enumerator kENET_PtpEventPort
PTP event port number.

enumerator kENET_PtpGnrlPort
PTP general port number.

enum _enet_tx_offload
Define the Tx checksum offload options.

Values:

enumerator kENET_TxOffloadDisable
Disable Tx checksum offload.

enumerator kENET_TxOffloadIPHeader
Enable IP header checksum calculation and insertion.

enumerator kENET_TxOffloadIPHeaderPlusPayload
Enable IP header and payload checksum calculation and insertion.

enumerator kENET_TxOffloadAll
Enable IP header, payload and pseudo header checksum calculation and insertion.

enum _enet_vlan_tpid
Ethernet VLAN Tag protocol identifiers.

Values:

enumerator kENET_StanCvlan
C-VLAN 0x8100.

enumerator kENET_StanSvlan
S-VLAN 0x88A8.

enum _enet_vlan_ops
Ethernet VLAN operations.

Values:

enumerator kENET_NoOps
Not do anything.

enumerator kENET_VlanRemove
Remove VLAN Tag.

enumerator kENET_VlanInsert
Insert VLAN Tag.

enumerator kENET_VlanReplace
Replace VLAN Tag.

enum _enet_vlan_strip
Ethernet VLAN strip setting.

Values:

enumerator kENET_VlanNotStrip
Not strip frame.

enumerator kENET_VlanFilterPassStrip
Strip if VLAN filter passes.

enumerator kENET_VlanFilterFailStrip
Strip if VLAN filter fails.

enumerator kENET_VlanAlwaysStrip
Always strip.

enum __enet_vlan_tx_channel
Ethernet VLAN Tx channels.

Values:

enumerator kENET_VlanTagAllChannels
VLAN tag is inserted for every packets transmitted by the MAC.

enumerator kENET_VlanTagChannel0
VLAN tag is inserted for the frames transmitted by channel 0.

enumerator kENET_VlanTagChannel1
VLAN tag is inserted for the frames transmitted by channel 1.

typedef enum *_enet_mii_mode* enet_mii_mode_t
Defines the MII/RMII mode for data interface between the MAC and the PHY.

typedef enum *_enet_mii_speed* enet_mii_speed_t
Defines the 10/100 Mbps speed for the MII data interface.

typedef enum *_enet_mii_duplex* enet_mii_duplex_t
Defines the half or full duplex for the MII data interface.

typedef enum *_enet_dma_burstlen* enet_dma_burstlen_t
Define the DMA maximum transmit burst length.

typedef enum *_enet_desc_flag* enet_desc_flag_t
Define the flag for the descriptor.

typedef enum *_enet_systime_op* enet_systime_op_t
Define the system time adjust operation control.

typedef enum *_enet_ts_rollover_type* enet_ts_rollover_type_t
Define the system time rollover control.

typedef enum *_enet_special_config* enet_special_config_t
Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, these is no need to set this control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the enet_config_t.

Note: "kENET_StoreAndForward" is recommended to be set when the ENET_PTP1588FEATURE_REQUIRED is defined or else the timestamp will be mess-up when the overflow happens.

---

typedef enum *_enet_dma_interrupt_enable* enet_dma_interrupt_enable_t
List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

typedef enum *_enet_mac_interrupt_enable* enet_mac_interrupt_enable_t
List of mac interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

typedef enum *_enet_event* enet_event_t

Defines the common interrupt event for callback use.

typedef enum *_enet_dma_tx_sche* enet_dma_tx_sche_t

Define the DMA transmit arbitration for multi-queue.

typedef enum *_enet_mtl_multiqueue_txsche* enet_mtl_multiqueue_txsche_t

Define the MTL Tx scheduling algorithm for multiple queues/rings.

typedef enum *_enet_mtl_multiqueue_rxsche* enet_mtl_multiqueue_rxsche_t

Define the MTL Rx scheduling algorithm for multiple queues/rings.

typedef enum *_enet_mtl_rxqueuemap* enet_mtl_rxqueuemap_t

Define the MTL Rx queue and DMA channel mapping.

typedef enum *_enet_ptp_event_type* enet_ptp_event_type_t

Defines the ENET PTP message related constant.

typedef enum *_enet_tx_offload* enet_tx_offload_t

Define the Tx checksum offload options.

typedef enum *_enet_vlan_tpid* enet_vlan_tpid_t

Ethernet VLAN Tag protocol identifiers.

typedef enum *_enet_vlan_ops* enet_vlan_ops_t

Ethernet VLAN operations.

typedef enum *_enet_vlan_strip* enet_vlan_strip_t

Ethernet VLAN strip setting.

typedef enum *_enet_vlan_tx_channel* enet_vlan_tx_channel_t

Ethernet VLAN Tx channels.

typedef struct *_enet_rx_bd_struct* enet_rx_bd_struct_t

Defines the receive descriptor structure It has the read-format and write-back format structures. They both have the same size with different region definition. So we define common name as the recive descriptor structure. When initialize the buffer descriptors, read-format region mask bits should be used. When Rx frame has been in the buffer descriptors, write-back format region store the Rx result information.

typedef struct *_enet_tx_bd_struct* enet_tx_bd_struct_t

Defines the transmit descriptor structure It has the read-format and write-back format structure. They both has the same size with different region definition. So we define common name as the transmit descriptor structure. When initialize the buffer descriptors for Tx, read-format region mask bits should be used. When frame has been transmitted, write-back format region store the Tx result information.

typedef struct *_enet_tx_bd_config_struct* enet_tx_bd_config_struct_t

Defines the Tx BD configuration structure.

typedef struct *_enet_ptp_time* enet_ptp_time_t

Defines the ENET PTP time stamp structure.

typedef struct *enet_tx_reclaim_info* enet_tx_reclaim_info_t

Defines the Tx reclaim information structure.

typedef struct *_enet_tx_dirty_ring* enet_tx_dirty_ring_t

Defines the ENET transmit dirty addresses ring/queue structure.

typedef struct *_enet_buffer_config* enet_buffer_config_t

Defines the buffer descriptor configure structure.

Notes:

a. The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.

b. The recommended minimum Tx/Rx ring length is 4.

c. The Tx/Rx descriptor tail address shall be the address pointer to the address just after the end of the last last descriptor. because only the descriptors between the start address and the tail address will be used by DMA.

d. The decriptor address is the start address of all used contiguous memory. for example, the rxDescStartAddrAlign is the start address of rxRingLen contiguous descriptor memorise for Rx descriptor ring 0.

e. The "*rxBufferstartAddr" is the first element of rxRingLen (2*rxRingLen for double buffers) Rx buffers. It means the *rxBufferStartAddr is the Rx buffer for the first descriptor the *rxBufferStartAddr + 1 is the Rx buffer for the second descriptor or the Rx buffer for the second buffer in the first descriptor. So please make sure the rxBufferStartAddr is the address of a rxRingLen or 2*rxRingLen array.

typedef struct *enet_multiqueue_config* enet_multiqueue_config_t

Defines the configuration when multi-queue is used.

typedef void *(*enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t channel)

Defines the Rx memory buffer alloc function pointer.

typedef void (*enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t channel)

Defines the Rx memory buffer free function pointer.

typedef struct *_enet_config* enet_config_t

Defines the basic configuration structure for the ENET device.

Note:

a. Default the signal queue is used so the "multiqueueCfg" is set default with NULL. Set the pointer with a valid configration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

typedef struct *_enet_handle* enet_handle_t

typedef void (*enet_callback_t)(ENET_Type *base, *enet_handle_t* *handle, *enet_event_t* event, uint8_t channel, *enet_tx_reclaim_info_t* *txReclaimInfo, void *userData)

ENET callback function.

typedef struct *_enet_tx_bd_ring* enet_tx_bd_ring_t

Defines the ENET transmit buffer descriptor ring/queue structure.

typedef struct *_enet_rx_bd_ring* enet_rx_bd_ring_t

Defines the ENET receive buffer descriptor ring/queue structure.

typedef struct *_enet_buffer_struct* enet_buffer_struct_t

typedef struct *_enet_rx_frame_attribute_struct* enet_rx_frame_attribute_t

Rx frame attribute structure.

typedef struct *_enet_rx_frame_error* enet_rx_frame_error_t

Defines the Rx frame error structure.

typedef struct *_enet_rx_frame_struct* enet_rx_frame_struct_t

Defines the Rx frame data structure.

typedef struct *_enet_tx_config_struct* enet_tx_config_struct_t

typedef struct *_enet_tx_frame_struct* enet_tx_frame_struct_t

typedef struct *_enet_vlan_tag* enet_vlan_tag_t
Ethernet VLAN Tag.

typedef struct *_enet_vlan_tx_config* enet_vlan_tx_config_t
Ethernet VLAN configuration for Tx.

typedef struct *_enet_vlan_ctrl* enet_vlan_ctrl_t
Ethernet VLAN control.

typedef void (*enet_isr_t)(ENET_Type *base, *enet_handle_t* *handle)

const *clock_ip_name_t* s_enetClock[]
Pointers to enet clocks for each instance.

ENET_FRAME_MAX_FRAMELEN
Default maximum ethernet frame size.

ENET_FCS_LEN
Ethernet Rx frame FCS length.

ENET_ADDR_ALIGNMENT
Recommended ethernet buffer alignment.

ENET_BUFF_ALIGNMENT
Receive buffer alignment shall be 4bytes-aligned.

ENET_RING_NUM_MAX
The maximum number of Tx/Rx descriptor rings.

ENET_MTL_RXFIFOSIZE
The Rx fifo size.

ENET_MTL_TXFIFOSIZE
The Tx fifo size.

ENET_MACINT_ENUM_OFFSET
The offset for mac interrupt in enum type.

ENET_FRAME_TX_LEN_LIMITATION
The Tx frame length software limitation.

ENET_FRAME_RX_ERROR_BITS(x)
The Rx frame error bits field.

void ENET_GetDefaultConfig(*enet_config_t* *config)
Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET configure structure for ENET_Init(). User may use the initialized structure unchanged in ENET_Init(), or modify some fields of the structure before calling ENET_Init(). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

- config – The ENET mac controller configuration structure pointer.

void ENET_Init(ENET_Type *base, const *enet_config_t* *config, uint8_t *macAddr, uint32_t clkSrcHz)

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET basic configuration.

Note: As our transactional transmit API use the zero-copy transmit buffer. So there are two thing we emphasize here:

a. Tx buffer free/requeue for application should be done in the Tx interrupt handler. Please set callback: kENET_TxIntEvent with Tx buffer free/requeue process APIs.

b. The Tx interrupt is forced to open.

Parameters

- base – ENET peripheral base address.
- config – ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- clkSrcHz – ENET input reference clock.

void ENET_Deinit(ENET_Type *base)

Deinitializes the ENET module.

This function gates the module clock and disables the ENET module.

Parameters

- base – ENET peripheral base address.

status_t ENET_DescriptorInit(ENET_Type *base, *enet_config_t* *config, *enet_buffer_config_t* *bufferConfig)

Initialize for all ENET descriptors.

Note: This function finishes all Tx/Rx descriptors initialization. The descriptor initialization should be called after ENET_Init().

Parameters

- base – ENET peripheral base address.
- config – The configuration for ENET.
- bufferConfig – All buffers configuration.

status_t ENET_RxBufferAllocAll(ENET_Type *base, *enet_handle_t* *handle)

Allocates Rx buffers for all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will populate initial buffers in all BDs for receiving. Then ENET_GetRxFrame() is used to get Rx frame with zero copy, it will allocate new buffer to replace the buffer in BD taken by application, application should free those buffers after they're used.

Note: This function should be called after ENET_CreateHandler() and buffer allocating callback function should be ready.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.

void ENET_RxBufferFreeAll(ENET_Type *base, *enet_handle_t* *handle)

Frees Rx buffers in all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will free left buffers in all BDs.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.

void ENET_StartRxTx(ENET_Type *base, uint8_t txRingNum, uint8_t rxRingNum)

Starts the ENET Tx/Rx. This function enable the Tx/Rx and starts the Tx/Rx DMA. This shall be set after ENET initialization and before starting to receive the data.

Note: This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required.

Parameters

- base – ENET peripheral base address.
- rxRingNum – The number of the used Rx rings. It shall not be larger than the ENET_RING_NUM_MAX(2). If the ringNum is set with 1, the ring 0 will be used.
- txRingNum – The number of the used Tx rings. It shall not be larger than the ENET_RING_NUM_MAX(2). If the ringNum is set with 1, the ring 0 will be used.

void ENET_SetISRHandler(ENET_Type *base, *enet_isr_t* ISRHandler)

Set the second level IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

static inline void ENET_SetMII(ENET_Type *base, *enet_mii_speed_t* speed, *enet_mii_duplex_t* duplex)

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and dulpex for MAC.

Parameters

- base – ENET peripheral base address.
- speed – The speed of the RMII mode.
- duplex – The duplex of the RMII mode.

void ENET_SetSMI(ENET_Type *base, uint32_t clkSrcHz)

Sets the ENET SMI(serial management interface)- MII management interface.

Parameters

- base – ENET peripheral base address.

- clkSrcHz – ENET peripheral clock source.

static inline bool ENET_IsSMIBusy(ENET_Type *base)

Checks if the SMI is busy.

Parameters

- base – ENET peripheral base address.

Returns

The status of MII Busy status.

static inline uint16_t ENET_ReadSMIData(ENET_Type *base)

Reads data from the PHY register through SMI interface.

Parameters

- base – ENET peripheral base address.

Returns

The data read from PHY

void ENET_StartSMIWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

Sends the MDIO IEEE802.3 Clause 22 format write command.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- data – The data written to PHY.

void ENET_StartSMIRead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr)

Sends the MDIO IEEE802.3 Clause 22 format read command.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.

status_t ENET_MDIOWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- data – The data written to PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

status_t ENET_MDIORead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.

- phyAddr – The PHY address.
- regAddr – The PHY register.
- pData – The data read from PHY.

Returns
kStatus_Success MDIO access succeeds.

Returns
kStatus_Timeout MDIO access timeout.

uint32_t ENET_GetInstance(ENET_Type *base)

Get the ENET instance from peripheral base address.

Parameters

- base – ENET peripheral base address.

Returns
ENET instance.

static inline void ENET_SetMacAddr(ENET_Type *base, uint8_t *macAddr)

Sets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET_GetMacAddr(ENET_Type *base, uint8_t *macAddr)

Gets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

static inline void ENET_AcceptAllMulticast(ENET_Type *base)

Enable ENET device to accept all multicast frames.

Parameters

- base – ENET peripheral base address.

static inline void ENET_RejectAllMulticast(ENET_Type *base)

ENET device reject to accept all multicast frames.

Parameters

- base – ENET peripheral base address.

void ENET_EnterPowerDown(ENET_Type *base, uint32_t *wakeFilter)

Set the MAC to enter into power down mode. the remote power wake up frame and magic frame can wake up the ENET from the power down mode.

Parameters

- base – ENET peripheral base address.
- wakeFilter – The wakeFilter provided to configure the wake up frame fitlter. Set the wakeFilter to NULL is not required. But if you have the filter requirement, please make sure the wakeFilter pointer shall be eight continous 32-bits configuration.

static inline void ENET_ExitPowerDown(ENET_Type *base)

Set the MAC to exit power down mode. Eixt from the power down mode and recover to normal work mode.

Parameters

- base – ENET peripheral base address.

status_t ENET_SetVlanCtrl(ENET_Type *base, *enet_vlan_ctrl_t* *control)

Set VLAN control.

Parameters

- base – ENET peripheral base address.
- control – VLAN control configuration.

status_t ENET_SetTxOuterVlan(ENET_Type *base, *enet_vlan_tx_config_t* *config, *enet_vlan_tx_channel_t* channel)

Set Tx outer VLAN configuration.

Parameters

- base – ENET peripheral base address.
- config – Tx VLAN operation configuration.
- channel – The channel to apply this configuration.

status_t ENET_SetTxInnerVlan(ENET_Type *base, *enet_vlan_tx_config_t* *config)

Set Tx inner VLAN configuration.

Parameters

- base – ENET peripheral base address.
- config – Tx VLAN operation configuration.

void ENET_EnableInterrupts(ENET_Type *base, uint32_t mask)

Enables the ENET DMA and MAC interrupts.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enet_dma_interrupt_enable_t and enet_mac_interrupt_enable_t. For example, to enable the dma and mac interrupt, do the following.

```
ENET_EnableInterrupts(ENET, kENET_DmaRx | kENET_DmaTx | kENET_MacPmt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of both enumeration :: enet_dma_interrupt_enable_t and enet_mac_interrupt_enable_t.

void ENET_DisableInterrupts(ENET_Type *base, uint32_t mask)

Disables the ENET DMA and MAC interrupts.

This function disables the ENET interrupt according to the provided mask. The mask is a logical OR of enet_dma_interrupt_enable_t and enet_mac_interrupt_enable_t. For example, to disable the dma and mac interrupt, do the following.

```
ENET_DisableInterrupts(ENET, kENET_DmaRx | kENET_DmaTx | kENET_MacPmt);
```

Parameters

- base – ENET peripheral base address.

- mask – ENET interrupts to disables. This is a logical OR of both enumeration :: enet_dma_interrupt_enable_t and enet_mac_interrupt_enable_t.

static inline uint32_t ENET_GetDmaInterruptStatus(ENET_Type *base, uint8_t channel)

Gets the ENET DMA interrupt status flag.

Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than ENET_RING_NUM_MAX.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: enet_dma_interrupt_enable_t.

static inline void ENET_ClearDmaInterruptStatus(ENET_Type *base, uint8_t channel, uint32_t mask)

Clear the ENET DMA interrupt status flag.

Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than ENET_RING_NUM_MAX.
- mask – The event status of the interrupt source. This is the logical OR of members of the enumeration :: enet_dma_interrupt_enable_t.

static inline uint32_t ENET_GetMacInterruptStatus(ENET_Type *base)

Gets the ENET MAC interrupt status flag.

Parameters

- base – ENET peripheral base address.

Returns

The event status of the interrupt source. Use the enum in enet_mac_interrupt_enable_t and right shift ENET_MACINT_ENUM_OFFSET to mask the returned value to get the exact interrupt status.

void ENET_ClearMacInterruptStatus(ENET_Type *base, uint32_t mask)

Clears the ENET mac interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the enet_mac_interrupt_enable_t. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_ClearMacInterruptStatus(ENET, kENET_MacPmt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration :: enet_mac_interrupt_enable_t.

static inline bool ENET_IsTxDescriptorDmaOwn(*enet_tx_bd_struct_t* *txDesc)

Get the Tx descriptor DMA Own flag.

Parameters

- txDesc – The given Tx descriptor.

Return values

True – the dma own Tx descriptor, false application own Tx descriptor.

void ENET_SetupTxDescriptor(*enet_tx_bd_struct_t* *txDesc, void *buffer1, uint32_t bytes1, void *buffer2, uint32_t bytes2, uint32_t framelen, bool intEnable, bool tsEnable, *enet_desc_flag_t* flag, uint8_t slotNum)

Setup a given Tx descriptor. This function is a low level functional API to setup or prepare a given Tx descriptor.

Note: This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required. Transmit buffers are 'zero-copy' buffers, so the buffer must remain in memory until the packet has been fully transmitted. The buffers should be free or requeued in the transmit interrupt irq handler.

Parameters

- txDesc – The given Tx descriptor.
- buffer1 – The first buffer address in the descriptor.
- bytes1 – The bytes in the fist buffer.
- buffer2 – The second buffer address in the descriptor.
- bytes2 – The bytes in the second buffer.
- framelen – The length of the frame to be transmitted.
- intEnable – Interrupt enable flag.
- tsEnable – The timestamp enable.
- flag – The flag of this Tx desciriptor, see "enet_desc_flag_t" .
- slotNum – The slot num used for AV mode only.

static inline void ENET_UpdateTxDescriptorTail(ENET_Type *base, uint8_t channel, uint32_t txDescTailAddrAlign)

Update the Tx descriptor tail pointer. This function is a low level functional API to update the the Tx descriptor tail. This is called after you setup a new Tx descriptor to update the tail pointer to make the new descritor accessable by DMA.

Parameters

- base – ENET peripheral base address.
- channel – The Tx DMA channel.
- txDescTailAddrAlign – The new Tx tail pointer address.

static inline void ENET_UpdateRxDescriptorTail(ENET_Type *base, uint8_t channel, uint32_t rxDescTailAddrAlign)

Update the Rx descriptor tail pointer. This function is a low level functional API to update the the Rx descriptor tail. This is called after you setup a new Rx descriptor to update the tail pointer to make the new descritor accessable by DMA and to anouse the Rx poll command for DMA.

Parameters

- base – ENET peripheral base address.
- channel – The Rx DMA channel.
- rxDescTailAddrAlign – The new Rx tail pointer address.

static inline uint32_t ENET_GetRxDescriptor(*enet_rx_bd_struct_t* *rxDesc)

Gets the context in the ENET Rx descriptor. This function is a low level functional API to get the the status flag from a given Rx descriptor.

Note: This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required.

Parameters

- rxDesc – The given Rx descriptor.

Return values

The – RDES3 regions for write-back format Rx buffer descriptor.

void ENET_UpdateRxDescriptor(*enet_rx_bd_struct_t* *rxDesc, void *buffer1, void *buffer2, bool intEnable, bool doubleBuffEnable)

Updates the buffers and the own status for a given Rx descriptor. This function is a low level functional API to Updates the buffers and the own status for a given Rx descriptor.

Note: This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required.

Parameters

- rxDesc – The given Rx descriptor.
- buffer1 – The first buffer address in the descriptor.
- buffer2 – The second buffer address in the descriptor.
- intEnable – Interrupt enable flag.
- doubleBuffEnable – The double buffer enable flag.

void ENET_CreateHandler(ENET_Type *base, *enet_handle_t* *handle, *enet_config_t* *config, *enet_buffer_config_t* *bufferConfig, *enet_callback_t* callback, void *userData)

Create ENET Handler.

This is a transactional API and it's provided to store all datas which are needed during the whole transactional process. This API should not be used when you use functional APIs to do data Tx/Rx. This is funtion will store many data/flag for transactional use.

Parameters

- base – ENET peripheral base address.
- handle – ENET handler.
- config – ENET configuration.
- bufferConfig – ENET buffer configuration.
- callback – The callback function.
- userData – The application data.

status_t ENET_GetRxFrameSize(ENET_Type *base, *enet_handle_t* *handle, uint32_t *length, uint8_t channel)

Gets the size of the read frame. This function gets a received frame size from the ENET buffer descriptors.

Note: The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, ENET_ReadFrame() should be called to update the receive buffers If the result is not "kStatus_ENET_RxFrameEmpty".

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- length – The length of the valid frame received.
- channel – The DMAC channel for the Rx.

Return values

- kStatus_ENET_RxFrameEmpty – No frame received. Should not call ENET_ReadFrame to read frame.
- kStatus_ENET_RxFrameError – Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.
- kStatus_Success – Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input.

status_t ENET_ReadFrame(ENET_Type *base, *enet_handle_t* *handle, uint8_t *data, uint32_t length, uint8_t channel, *enet_ptp_time_t* *timestamp)

Reads a frame from the ENET device. This function reads a frame from the ENET DMA descriptors. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. For example use Rx dma channel 0:

```
uint32_t length;
enet_handle_t g_handle;
Comment: Get the received frame size firstly.
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    Comment: Allocate memory here with the size of "length"
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0);
    }
    else
    {
      status = ENET_ReadFrame(ENET, &g_handle, data, length, 0);
    }
}
else if (status == kStatus_ENET_RxFrameError)
{
    Comment: Update the received buffer when a error frame is received.
    ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0);
}
```

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- data – The data buffer provided by user to store the frame which memory size should be at least “length”.

- length – The size of the data buffer which is still the length of the received frame.
- channel – The Rx DMA channel. Shall not be larger than 2.
- timestamp – The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

status_t ENET_GetRxFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_rx_frame_struct_t* *rxFrame, uint8_t channel)

Receives one frame in specified BD ring with zero copy.

This function will use the user-defined allocate and free callback. Every time application gets one frame through this function, driver will allocate new buffers for the BDs whose buffers have been taken by application.

Note: This function will drop current frame and update related BDs as available for DMA if new buffers allocating fails. Application must provide a memory pool including at least BD number + 1 buffers(+2 if enable double buffer) to make this function work normally.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- rxFrame – The received frame information structure provided by user.
- channel – The Rx DMA channel. Shall not be larger than 2.

Return values

- kStatus_Success – Succeed to get one frame and allocate new memory for Rx buffer.
- kStatus_ENET_RxFrameEmpty – There's no Rx frame in the BD.
- kStatus_ENET_RxFrameError – There's issue in this receiving. In this function, issue frame will be dropped.
- kStatus_ENET_RxFrameDrop – There's no new buffer memory for BD, dropped this frame.

status_t ENET_SendFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_tx_frame_struct_t* *txFrame, uint8_t channel)

Transmits an ENET frame.

Note: The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses input buffer for Tx, application should reclaim the buffer after Tx is over.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- txFrame – The Tx frame structure.
- channel – Channel to send the frame, same with queue index.

Return values

- kStatus_Success – Send frame succeed.
- kStatus_ENET_TxFrameBusy – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus_ENET_TxFrameBusy. Also need to pay attention to reclaim Tx frame after Tx is over.
- kStatus_ENET_TxFrameOverLen – Transmit frme length exceeds the 0x3FFF limit defined by the driver.

void ENET_ReclaimTxDescriptor(ENET_Type *base, *enet_handle_t* *handle, uint8_t channel)

Reclaim Tx descriptors. This function is used to update the Tx descriptor status and store the Tx timestamp when the 1588 feature is enabled. This is called by the transmit interupt IRQ handler after the complete of a frame transmission.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- channel – The Tx DMA channnel.

void ENET_IRQHandler(ENET_Type *base, *enet_handle_t* *handle)

The ENET IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

FSL_ENET_DRIVER_VERSION

Defines the driver version.

ENET_RXDESCRIP_RD_BUFF1VALID_MASK

Defines for read format.

Buffer1 address valid.

ENET_RXDESCRIP_RD_BUFF2VALID_MASK

Buffer2 address valid.

ENET_RXDESCRIP_RD_IOC_MASK

Interrupt enable on complete.

ENET_RXDESCRIP_RD_OWN_MASK

Own bit.

ENET_RXDESCRIP_WR_ERR_MASK

Defines for write back format.

ENET_RXDESCRIP_WR_PYLOAD_MASK

ENET_RXDESCRIP_WR_PTPMSGTYPE_MASK

ENET_RXDESCRIP_WR_PTPTYPE_MASK

ENET_RXDESCRIP_WR_PTPVERSION_MASK

ENET_RXDESCRIP_WR_PTPTSA_MASK

ENET_RXDESCRIP_WR_PACKETLEN_MASK

ENET_RXDESCRIP_WR_ERRSUM_MASK

ENET_RXDESCRIP_WR_TYPE_MASK

ENET_RXDESCRIP_WR_DE_MASK

ENET_RXDESCRIP_WR_RE_MASK

ENET_RXDESCRIP_WR_OE_MASK

ENET_RXDESCRIP_WR_RS0V_MASK

ENET_RXDESCRIP_WR_RS1V_MASK

ENET_RXDESCRIP_WR_RS2V_MASK

ENET_RXDESCRIP_WR_LD_MASK

ENET_RXDESCRIP_WR_FD_MASK

ENET_RXDESCRIP_WR_CTXT_MASK

ENET_RXDESCRIP_WR_OWN_MASK

ENET_TXDESCRIP_RD_BL1_MASK
Defines for read format.

ENET_TXDESCRIP_RD_BL2_MASK

ENET_TXDESCRIP_RD_BL1(n)

ENET_TXDESCRIP_RD_BL2(n)

ENET_TXDESCRIP_RD_TTSE_MASK

ENET_TXDESCRIP_RD_IOC_MASK

ENET_TXDESCRIP_RD_FL_MASK

ENET_TXDESCRIP_RD_FL(n)

ENET_TXDESCRIP_RD_CIC(n)

ENET_TXDESCRIP_RD_TSE_MASK

ENET_TXDESCRIP_RD_SLOT(n)

ENET_TXDESCRIP_RD_SAIC(n)

ENET_TXDESCRIP_RD_CPC(n)

ENET_TXDESCRIP_RD_LDFD(n)

ENET_TXDESCRIP_RD_LD_MASK

ENET_TXDESCRIP_RD_FD_MASK

ENET_TXDESCRIP_RD_CTXT_MASK

ENET_TXDESCRIP_RD_OWN_MASK

ENET_TXDESCRIP_WB_TTSS_MASK
Defines for write back format.

ENET_ABNORM_INT_MASK

ENET_NORM_INT_MASK

___IO uint32_t rdes0
: Receive descriptor 0

___IO uint32_t rdes1
: Receive descriptor 1

___IO uint32_t rdes2
: Receive descriptor 2

___IO uint32_t rdes3
: Receive descriptor 3

___IO uint32_t tdes0
: Transmit descriptor 0

___IO uint32_t tdes1
: Transmit descriptor 1

___IO uint32_t tdes2
: Transmit descriptor 2

___IO uint32_t tdes3
: Transmit descriptor 3

void *buffer1
: The first buffer address in the descriptor.

uint32_t bytes1
: The bytes in the fist buffer.

void *buffer2
: The second buffer address in the descriptor.

uint32_t bytes2
: The bytes in the second buffer.

uint32_t framelen
: The length of the frame to be transmitted.

bool intEnable
: Interrupt enable flag.

bool tsEnable
: The timestamp enable.

enet_tx_offload_t txOffloadOps
: The Tx checksum offload option, only vaild for Queue 0.

enet_desc_flag_t flag
: The flag of this tx desciriptor, see “enet_qos_desc_flag”.

uint8_t slotNum
: The slot number used for AV mode only.

uint64_t second
: Second.

uint32_t nanosecond
: Nanosecond.

void *context
: User specified data, could be buffer address for free

bool isTsAvail
: Flag indicates timestamp available status

enet_ptp_time_t timeStamp
: Timestamp of frame

enet_tx_reclaim_info_t *txDirtyBase
: Dirty buffer descriptor base address pointer.

uint16_t txGenIdx
: Tx generate index.

uint16_t txConsumIdx
: Tx consume index.

uint16_t txRingLen
: Tx ring length.

bool isFull
: Tx ring is full flag, add this parameter to avoid waste one element.

uint8_t rxRingLen
: The length of receive buffer descriptor ring.

uint8_t txRingLen
: The length of transmit buffer descriptor ring.

enet_tx_bd_struct_t *txDescStartAddrAlign
: Aligned transmit descriptor start address.

enet_tx_bd_struct_t *txDescTailAddrAlign
: Aligned transmit descriptor tail address.

enet_tx_reclaim_info_t *txDirtyStartAddr
: Start address of the dirty Tx frame information.

enet_rx_bd_struct_t *rxDescStartAddrAlign
: Aligned receive descriptor start address.

enet_rx_bd_struct_t *rxDescTailAddrAlign
: Aligned receive descriptor tail address.

uint32_t *rxBufferStartAddr
: Start address of the Rx buffers.

uint32_t rxBuffSizeAlign
: Aligned receive data buffer size.

enet_dma_tx_sche_t dmaTxSche
: Transmit arbitation.

enet_dma_burstlen_t burstLen
: Burset len for the queue 1.

uint8_t txdmaChnWeight[(2U)]
: Transmit channel weight.

enet_mtl_multiqueue_txsche_t mtltxSche
: Transmit schedule for multi-queue.

enet_mtl_multiqueue_rxsche_t mtlrxSche
Receive schedule for multi-queue.

uint8_t rxqueweight[(2U)]
Refer to the MTL RxQ Control register.

uint32_t txqueweight[(2U)]
Refer to the MTL TxQ Quantum Weight register.

uint8_t rxqueuePrio[(2U)]
Receive queue priority.

uint8_t txqueuePrio[(2U)]
Refer to Transmit Queue Priority Mapping register.

enet_mtl_rxqueuemap_t mtlrxQuemap
Rx queue DMA Channel mapping.

uint16_t specialControl
The logicl or of enet_special_config_t

uint32_t ptpClkHz
The PTP module source clock.

enet_multiqueue_config_t *multiqueueCfg
Use both Tx/Rx queue(dma channel) 0 and 1.

uint32_t interrupt
MAC interrupt source. A logical OR of enet_dma_interrupt_enable_t and enet_mac_interrupt_enable_t.

enet_mii_mode_t miiMode
MII mode.

enet_mii_speed_t miiSpeed
MII Speed.

enet_mii_duplex_t miiDuplex
MII duplex.

uint16_t pauseDuration
Used in the Tx flow control frame, only valid when kENET_FlowControlEnable is set.

enet_rx_alloc_callback_t rxBuffAlloc
Callback to alloc memory, must be provided for zero-copy Rx.

enet_rx_free_callback_t rxBuffFree
Callback to free memory, must be provided for zero-copy Rx.

enet_tx_bd_struct_t *txBdBase
Buffer descriptor base address pointer.

uint16_t txGenIdx
Tx generate index.

uint16_t txConsumIdx
Tx consum index.

volatile uint16_t txDescUsed
Tx descriptor used number.

uint16_t txRingLen
Tx ring length.

enet_rx_bd_struct_t *rxBdBase
Buffer descriptor base address pointer.

uint16_t rxGenIdx
The current available receive buffer descriptor pointer.

uint16_t rxRingLen
Receive ring length.

uint32_t rxBuffSizeAlign
Receive buffer size.

bool multiQueEnable
Multi-queue enable status.

bool doubleBuffEnable
The double buffer enable status.

bool rxintEnable
Rx interrupt enable status.

enet_rx_bd_ring_t rxBdRing[(2U)]
Receive buffer descriptor.

enet_tx_bd_ring_t txBdRing[(2U)]
Transmit buffer descriptor.

enet_tx_dirty_ring_t txDirtyRing[(2U)]
Transmit dirty buffers addresses.

uint32_t *rxBufferStartAddr[(2U)]
The Init-Rx buffers used for reinit corrupted BD due to write-back operation.

uint32_t txLenLimitation[(2U)]
Tx frame length limitation.

enet_callback_t callback
Callback function.

void *userData
Callback function parameter.

enet_rx_alloc_callback_t rxBuffAlloc
Callback to alloc memory, must be provided for zero-copy Rx.

enet_rx_free_callback_t rxBuffFree
Callback to free memory, must be provided for zero-copy Rx.

void *buffer
The buffer stores the whole or partial frame.

uint16_t length
The byte length of this buffer.

bool isTsAvail
Rx frame timestamp is available or not.

enet_ptp_time_t timestamp
The nanosecond part timestamp of this Rx frame.

bool statsDribbleErr
The received packet has a non-integer multiple of bytes (odd nibbles).

bool statsRxErr
: Receive error.

bool statsOverflowErr
: Rx FIFO overflow error.

bool statsWatchdogTimeoutErr
: Receive watchdog timeout.

bool statsGaintPacketErr
: Receive error.

bool statsRxFcsErr
: Receive CRC error.

enet_buffer_struct_t *rxBuffArray
: Rx frame buffer structure.

uint16_t totLen
: Rx frame total length.

enet_rx_frame_attribute_t rxAttribute
: Rx frame attribute structure.

enet_rx_frame_error_t rxFrameError
: Rx frame error.

uint8_t intEnable
: Enable interrupt every time one BD is completed.

uint8_t tsEnable
: Transmit timestamp enable.

uint8_t slotNum
: Slot number control bits in AV mode.

enet_tx_offload_t txOffloadOps
: Tx checksum offload option.

enet_buffer_struct_t *txBuffArray
: Tx frame buffer structure.

uint32_t txBuffNum
: Buffer number of this Tx frame.

enet_tx_config_struct_t txConfig
: Tx extra configuation.

void *context
: Driver reclaims and gives it in Tx over callback.

enet_vlan_tpid_t tpid
: VLAN TPID.

uint16_t pcp
: VLAN Priority.

uint16_t dei
: Drop Eligible indicator.

uint16_t vid
: VLAN Identifier.

bool txDescVlan
: Use VLAN configuration in Tx descriptor.

enet_vlan_tag_t tag
: VLAN Tag.

enet_vlan_ops_t ops
: VLAN operations.

bool svlanEnable
: The MAC transmitter and receiver consider the S-VLAN packets.

bool vlanInverseMatch
: True: Marks frames without matching as match, False: Marks matched frames.

bool vidComparison
: Only takes VLAN VID as match.

bool disableVlanTypeCheck
: Not check C-VLAN and S-VLAN.

bool doubleVlanEnable
: Enable the inner VLAN operations.

bool innerVlanFilterMatch
: Takes Inner VLAN as match.

bool outerTagInRxStatus
: Set outer VLAN in Rx Status.

bool innerTagInRxStatus
: Set inner VLAN in Rx Status.

enet_vlan_tag_t rxVlanTag
: VLAN tag for Rx match.

enet_vlan_strip_t rxOuterVlanStrip
: Outer VLAN Rx strip operation.

enet_vlan_strip_t rxInnerVlanStrip
: Inner VLAN Rx strip operation.

struct __enet__rx__bd__struct
: *#include <fsl_enet.h>* Defines the receive descriptor structure It has the read-format and write-back format structures. They both have the same size with different region definition. So we define common name as the recive descriptor structure. When initialize the buffer descriptors, read-format region mask bits should be used. When Rx frame has been in the buffer descriptors, write-back format region store the Rx result information.

struct __enet__tx__bd__struct
: *#include <fsl_enet.h>* Defines the transmit descriptor structure It has the read-format and write-back format structure. They both has the same size with different region definition. So we define common name as the transmit descriptor structure. When initialize the buffer descriptors for Tx, read-format region mask bits should be used. When frame has been transmitted, write-back format region store the Tx result information.

struct __enet__tx__bd__config__struct
: *#include <fsl_enet.h>* Defines the Tx BD configuration structure.

struct __enet__ptp__time
: *#include <fsl_enet.h>* Defines the ENET PTP time stamp structure.

struct enet_tx_reclaim_info

#include <fsl_enet.h> Defines the Tx reclaim information structure.

struct _enet_tx_dirty_ring

#include <fsl_enet.h> Defines the ENET transmit dirty addresses ring/queue structure.

struct _enet_buffer_config

#include <fsl_enet.h> Defines the buffer descriptor configure structure.

Notes:

a. The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.

b. The recommended minimum Tx/Rx ring length is 4.

c. The Tx/Rx descriptor tail address shall be the address pointer to the address just after the end of the last last descriptor. because only the descriptors between the start address and the tail address will be used by DMA.

d. The decriptor address is the start address of all used contiguous memory. for example, the rxDescStartAddrAlign is the start address of rxRingLen contiguous descriptor memorise for Rx descriptor ring 0.

e. The "*rxBufferstartAddr" is the first element of rxRingLen (2*rxRingLen for double buffers) Rx buffers. It means the *rxBufferStartAddr is the Rx buffer for the first descriptor the *rxBufferStartAddr + 1 is the Rx buffer for the second descriptor or the Rx buffer for the second buffer in the first descriptor. So please make sure the rxBufferStartAddr is the address of a rxRingLen or 2*rxRingLen array.

struct enet_multiqueue_config

#include <fsl_enet.h> Defines the configuration when multi-queue is used.

struct _enet_config

#include <fsl_enet.h> Defines the basic configuration structure for the ENET device.

Note:

a. Default the signal queue is used so the "multiqueueCfg" is set default with NULL. Set the pointer with a valid configration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

struct _enet_tx_bd_ring

#include <fsl_enet.h> Defines the ENET transmit buffer descriptor ring/queue structure.

struct _enet_rx_bd_ring

#include <fsl_enet.h> Defines the ENET receive buffer descriptor ring/queue structure.

struct _enet_handle

#include <fsl_enet.h> Defines the ENET handler structure.

struct _enet_buffer_struct

#include <fsl_enet.h>

struct _enet_rx_frame_attribute_struct

#include <fsl_enet.h> Rx frame attribute structure.

struct _enet_rx_frame_error

#include <fsl_enet.h> Defines the Rx frame error structure.

struct _enet_rx_frame_struct

#include <fsl_enet.h> Defines the Rx frame data structure.

struct _enet_tx_config_struct

#include <fsl_enet.h>

struct _enet_tx_frame_struct

#include <fsl_enet.h>

struct _enet_vlan_tag

#include <fsl_enet.h> Ethernet VLAN Tag.

struct _enet_vlan_tx_config

#include <fsl_enet.h> Ethernet VLAN configuration for Tx.

struct _enet_vlan_ctrl

#include <fsl_enet.h> Ethernet VLAN control.

2.74 MCX_SPC: System Power Control driver

uint8_t SPC_GetPeriphIOIsolationStatus(SPC_Type *base)

Gets Isolation status for each power domains.

This function gets the status which indicates whether certain peripheral and the IO pads are in a latched state as a result of having been in POWERDOWN mode.

Parameters

- base – SPC peripheral base address.

Returns

Current isolation status for each power domains. See _spc_power_domains for details.

static inline void SPC_ClearPeriphIOIsolationFlag(SPC_Type *base)

Clears peripherals and I/O pads isolation flags for each power domains.

This function clears peripherals and I/O pads isolation flags for each power domains. After recovering from the POWERDOWN mode, user must invoke this function to release the I/O pads and certain peripherals to their normal run mode state. Before invoking this function, user must restore chip configuration in particular pin configuration for enabled WUU wakeup pins.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_GetBusyStatusFlag(SPC_Type *base)

Gets SPC busy status flag.

This function gets SPC busy status flag. When SPC executing any type of power mode transition in ACTIVE mode or any of the SOC low power mode, the SPC busy status flag is set and this function returns true. When changing CORE LDO voltage level and DCDC voltage level in ACTIVE mode, the SPC busy status flag is set and this function return true.

Parameters

- base – SPC peripheral base address.

Returns

Ack busy flag. true - SPC is busy. false - SPC is not busy.

static inline bool SPC_CheckLowPowerReqest(SPC_Type *base)

Checks system low power request.

Note: Only when all power domains request low power mode entry, the result of this function is true. That means when all power domains request low power mode entry, the SPC regulators will be controlled by LP_CFG register.

Parameters

- base – SPC peripheral base address.

Returns

The system low power request check result.

- **true** All power domains have requested low power mode and SPC has entered a low power state and power mode configuration are based on the LP_CFG configuration register.
- **false** SPC in active mode and ACTIVE_CFG register control system power supply.

static inline void SPC_ClearLowPowerRequest(SPC_Type *base)

Clears system low power request, set SPC in active mode.

Parameters

- base – SPC peripheral base address.

static inline *spc_power_domain_low_power_mode_t* SPC_GetRequestedLowPowerMode(SPC_Type *base)

Check the last low-power mode that the power domain requested.

Parameters

- base – SPC peripheral base address.

Returns

The last low-power mode that the power domain requested.

static inline bool SPC_CheckSwitchState(SPC_Type *base)

Checks whether the power switch is on.

Parameters

- base – SPC peripheral base address.

Return values

- true – The power switch is on.
- false – The power switch is off.

spc_power_domain_low_power_mode_t SPC_GetPowerDomainLowPowerMode(SPC_Type *base, *spc_power_domain_id_t* powerDomainId)

Gets selected power domain's requested low power mode.

Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to spc_power_domain_id_t.

Returns
The selected power domain's requested low power mode, please refer to spc_power_domain_low_power_mode_t.

static inline bool SPC_CheckPowerDomainLowPowerRequest(SPC_Type *base, *spc_power_domain_id_t* powerDomainId)

Checks power domain's low power request.

Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to spc_power_domain_id_t.

Returns
The result of power domain's low power request.

- **true** The selected power domain requests low power mode entry.
- **false** The selected power domain does not request low power mode entry.

static inline void SPC_ClearPowerDomainLowPowerRequestFlag(SPC_Type *base, *spc_power_domain_id_t* powerDomainId)

Clears selected power domain's low power request flag.

Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to spc_power_domain_id_t.

static inline void SPC_TrimSRAMLdoRefVoltage(SPC_Type *base, uint8_t trimValue)

Trims SRAM retention regulator reference voltage, trim step is 12 mV, range is around 0.48V to 0.85V.

Parameters

- base – SPC peripheral base address.
- trimValue – Reference voltage trim value.

static inline void SPC_EnableSRAMLdo(SPC_Type *base, bool enable)

Enables/disables SRAM retention LDO.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable SRAM LDO :
  - **true** Enable SRAM LDO;
  - **false** Disable SRAM LDO.

static inline void SPC_RetainSRAMArray(SPC_Type *base, uint8_t mask)

Parameters

- base – SPC peripheral base address.
- mask – The OR'ed value of SRAM Array.

static inline void SPC_UnRetainSRAMArray(SPC_Type *base, uint8_t mask)

Unretain SRAM array.

Parameters

- base – SPC peripheral base address.
- mask – The OR'ed value of SRAM Array.

void SPC_SetLowPowerRequestConfig(SPC_Type *base, const *spc_lowpower_request_config_t* *config)

Configs Low power request output pin.

This function config the low power request output pin

Parameters

- base – SPC peripheral base address.
- config – Pointer the spc_lowpower_request_config_t structure.

static inline void SPC_EnableIntegratedPowerSwitchManually(SPC_Type *base, bool enable)

Enables/disables the integrated power switch manually.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable the integrated power switch:
  - **true** Enable the integrated power switch;
  - **false** Disable the integrated power switch.

static inline void SPC_EnableIntegratedPowerSwitchAutomatically(SPC_Type *base, bool sleepGate, bool wakeupUngate)

Enables/disables the integrated power switch automatically.

To gate the integrated power switch when chip enter low power modes, and ungate the switch after wake-up from low power modes:

```
SPC_EnableIntegratedPowerSwitchAutomatically(SPC, true, true);
```

Parameters

- base – SPC peripheral base address.
- sleepGate – Enable the integrated power switch when chip enter low power modes:
  - **true** SPC asserts an output pin at low-power entry to power-gate the switch;
  - **false** SPC does not assert an output pin at low-power entry to power-gate the switch.
- wakeupUngate – Enables the switch after wake-up from low power modes:
  - **true** SPC asserts an output pin at low-power exit to power-ungate the switch;
  - **false** SPC does not assert an output pin at low-power exit to power-ungate the switch.

void SPC_ConfigVddCoreGlitchDetector(SPC_Type *base, const *spc_vdd_core_glitch_detector_config_t* *config)

Configures VDD Core Glitch detector, including ripple counter selection, timeout value and so on.

Parameters

- base – SPC peripheral base address.

- config – Pointer to the structure in type of spc_vdd_core_glitch_detector_config_t.

static inline bool SPC_CheckGlitchRippleCounterOutput(SPC_Type *base, *spc_vdd_core_glitch_ripple_counter_select_t* rippleCounter)

Checks selected 4-bit glitch ripple counter's output.

Parameters

- base – SPC peripheral base address.
- rippleCounter – The ripple counter to check, please refer to spc_vdd_core_glitch_ripple_counter_select_t.

Return values

- true – The selected ripple counter output is 1, will generate interrupt or reset based on settings.
- false – The selected ripple counter output is 0.

static inline void SPC_ClearGlitchRippleCounterOutput(SPC_Type *base, *spc_vdd_core_glitch_ripple_counter_select_t* rippleCounter)

Clears output of selected glitch ripple counter.

Parameters

- base – SPC peripheral base address.
- rippleCounter – The ripple counter to check, please refer to spc_vdd_core_glitch_ripple_counter_select_t.

static inline void SPC_LockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)

After invoking this function, writes to SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register are ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)

After invoking this function, writes to SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register are allowed.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_CheckVddCoreVoltageGlitchResetControlState(SPC_Type *base)

Checks if SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.

Parameters

- base – SPC peripheral base address.

Return values

- true – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.
- false – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is not writable.

void SPC_SetSRAMOperateVoltage(SPC_Type *base, const *spc_sram_voltage_config_t* *config)

Set SRAM operate voltage.

Parameters

- base – SPC peripheral base address.

- config – The pointer to spc_sram_voltage_config_t, specifies the configuration of sram voltage.

static inline *spc_bandgap_mode_t* SPC_GetActiveModeBandgapMode(SPC_Type *base)

Gets the Bandgap mode in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

Bandgap mode in the type of spc_bandgap_mode_t enumeration.

static inline uint32_t SPC_GetActiveModeVoltageDetectStatus(SPC_Type *base)

Gets all voltage detectors status in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

All voltage detectors status in Active mode.

status_t SPC_SetActiveModeBandgapModeConfig(SPC_Type *base, *spc_bandgap_mode_t* mode)

Configs Bandgap mode in Active mode.

Note: To disable bandgap in Active mode:

a. Disable all LVD's and HVD's in active mode;

b. Disable Glitch detect;

c. Configrue LDO's and DCDC to low drive strength in active mode;

d. Invoke this function to disable bandgap in active mode; otherwise the error status will be reported.

Note: Some other system resources(such as PLL, CMP) require bandgap to be enabled, to disable bandgap please take care of other system resources.

Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

Return values

- kStatus_SPC_BandgapModeWrong – The Bandgap can not be disabled in active mode.
- kStatus_Success – Config Bandgap mode in Active power mode successful.

static inline void SPC_EnableActiveModeCMPBandgapBuffer(SPC_Type *base, bool enable)

Enables/Disable the CMP Bandgap Buffer in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference voltage to CMP. false - Disable Buffer Stored Reference voltage to CMP.

static inline void SPC_SetActiveModeVoltageTrimDelay(SPC_Type *base, uint16_t delay)

Sets the delay when the regulators change voltage level in Active mode.

Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles.

status_t SPC_SetActiveModeRegulatorsConfig(SPC_Type *base, const *spc_active_mode_regulators_config_t* *config)

Configs all settings of regulators in Active mode at a time.

Note: This function is used to overwrite all settings of regulators(including bandgap mode, regulators' drive strength and voltage level) in active mode at a time.

Note: Enable/disable LVDs/HVDs before invoking this function.

Note: This function will check input parameters based on hardware restrictions before setting registers, if input parameters do not satisfy hardware restrictions the specific error will be reported.

Note: Some hardware restrictions not covered, application should be aware of this and follow this hardware restrictions otherwise some unkown issue may occur:

a. If Core LDO's drive strength are set to same value in both Active mode and low power mode, the voltage level should also set to same value.

b. When switching Core LDO's drive strength from low to normal, ensure the LDO_CORE high voltage level is set to same level that was set prior to switching to the LDO_CORE drive strength. Otherwise, if the LVDs are enabled, an unexpected LVD can occur.

Note: If this function can not satisfy some tricky settings, please invoke other APIs in low-level function group.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_active_mode_regulators_config_t structure.

Return values

- kStatus_Success – Config regulators in Active power mode successful.
- kStatus_SPC_BandgapModeWrong – Based on input setting, bandgap can not be disabled.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – Any of LVDs/HVDs kept enabled before invoking this function.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Fail to regulator to Over Drive Voltage due to System VDD HVD is not disabled.

- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Any of LVDs/HVDs kept enabled before invoking this function.
- kStatus_SPC_CORELDOVoltageWrong – Core LDO and System LDO do not have same voltage level.

static inline void SPC_DisableActiveModeVddCoreGlitchDetect(SPC_Type *base, bool disable)

Disables/Enables VDD Core Glitch Detect in Active mode.

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
  - **true** Disable VDD Core Low Voltage detect;
  - **false** Enable VDD Core Low Voltage detect.

static inline bool SPC_CheckActiveModeVddCoreGlitchDetectEnabled(SPC_Type *base)

Check if Glitch detect hardware is enabled in active mode.

Parameters

- base – SPC peripheral base address.

Returns

Indicate if Glitch detector is enabled.

static inline void SPC_EnableActiveModeAnalogModules(SPC_Type *base, uint32_t maskValue)

Enables analog modules in active mode.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to enable in active mode, should be the OR'ed value of spc_analog_module_control.

static inline void SPC_DisableActiveModeAnalogModules(SPC_Type *base, uint32_t maskValue)

Disables analog modules in active mode.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to disable in active mode, should be the OR'ed value of spc_analog_module_control.

static inline uint32_t SPC_GetActiveModeEnabledAnalogModules(SPC_Type *base)

Gets enabled analog modules that enabled in active mode.

Parameters

- base – SPC peripheral base address.

Returns

The mask of enabled analog modules that enabled in active mode.

static inline *spc_bandgap_mode_t* SPC_GetLowPowerModeBandgapMode(SPC_Type *base)

Gets the Bandgap mode in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns
Bandgap mode in the type of spc_bandgap_mode_t enumeration.

static inline uint32_t SPC_GetLowPowerModeVoltageDetectStatus(SPC_Type *base)

Gets the status of all voltage detectors in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns
The status of all voltage detectors in low power mode.

static inline void SPC_EnableLowPowerModeLowPowerIREF(SPC_Type *base, bool enable)

Enables/Disables Low Power IREF in low power modes.

This function enables/disables Low Power IREF. Low Power IREF can only get disabled in Deep power down mode. In other low power modes, the Low Power IREF is always enabled.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Low Power IREF. true - Enable Low Power IREF for Low Power modes. false - Disable Low Power IREF for Deep Power Down mode.

status_t SPC_SetLowPowerModeBandgapmodeConfig(SPC_Type *base, *spc_bandgap_mode_t* mode)

Configs Bandgap mode in Low Power mode.

Note: To disable Bandgap in Low-power mode:

a. Disable all LVD's ad HVD's in low power mode;

b. Disable Glitch detect in low power mode;

c. Configure LDO's and DCDC to low drive strength in low power mode;

d. Disable bandgap in low power mode; Otherwise, the error status will be reported.

Note: Some other system resources(such as PLL, CMP) require bandgap to be enabled, to disable bandgap please take care of other system resources.

Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

Return values

- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in Low Power mode is wrong.
- kStatus_Success – Config Bandgap mode in Low Power power mode successful.

static inline void SPC_EnableSRAMLdOLowPowerModeIREF(SPC_Type *base, bool enable)

Enables/disables SRAM_LDO deep power low power IREF.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable low power IREF :

- **true:** Low Power IREF is enabled ;
- **false:** Low Power IREF is disabled for power saving.

static inline void SPC_EnableLowPowerModeCMPBandgapBufferMode(SPC_Type *base, bool enable)

Enables/Disables CMP Bandgap Buffer.

This function gates CMP bandgap buffer. CMP bandgap buffer is automatically disabled and turned off in Deep Power Down mode.

Deprecated:

No longer used, please use SPC_EnableLowPowerModeCMPBandgapBuffer as instead.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference Voltage to CMP. false - Disable Buffer Stored Reference Voltage to CMP.

static inline void SPC_EnableLowPowerModeCMPBandgapBuffer(SPC_Type *base, bool enable)

Enables/Disables CMP Bandgap Buffer.

This function gates CMP bandgap buffer. CMP bandgap buffer is automatically disabled and turned off in Deep Power Down mode.

Deprecated:

No longer used.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference Voltage to CMP. false - Disable Buffer Stored Reference Voltage to CMP.

static inline void SPC_EnableLowPowerModeCoreVDDInternalVoltageScaling(SPC_Type *base, bool enable)

Enables/Disables CORE VDD IVS(Internal Voltage Scaling) in power down modes.

This function gates CORE VDD IVS. When enabled, the IVS regulator will scale the external input CORE VDD to a lower voltage level to reduce internal leakage. IVS is invalid in Sleep or Deep power down mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IVS. true - enable CORE VDD IVS in Power Down mode. false - disable CORE VDD IVS in Power Down mode.

static inline void SPC_SetLowPowerWakeUpDelay(SPC_Type *base, uint16_t delay)

Sets the delay when exit the low power modes.

Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles that the SPC waits on exit from low power modes.

status_t SPC_SetLowPowerModeRegulatorsConfig(SPC_Type *base, const *spc_lowpower_mode_regulators_config_t* *config)

Configs all settings of regulators in Low power mode at a time.

Note: This function is used to overwrite all settings of regulators(including bandgap mode, regulators' drive strength and voltage level) in low power mode at a time.

Note: Enable/disable LVDs/HVDs before invoking this function.

Note: This function will check input parameters based on hardware restrictions before setting registers, if input parameters do not satisfy hardware restrictions the specific error will be reported.

Note: Some hardware restrictions not covered, application should be aware of this and follow this hardware restrictions otherwise some unkown issue may occur:

a. If Core LDO's drive strength are set to same value in both Active mode and low power mode, the voltage level should also set to same value.

b. When switching Core LDO's drive strength from low to normal, ensure the LDO_CORE high voltage level is set to same level that was set prior to switching to the LDO_CORE drive strength. Otherwise, if the LVDs are enabled, an unexpected LVD can occur.

Note: If this function can not satisfy some tricky settings, please invoke other APIs in low-level function group.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_lowpower_mode_regulators_config_t structure.

Return values

- kStatus_Success – Config regulators in Low power mode successful.
- kStatus_SPC_BandgapModeWrong – The bandgap should not be disabled based on input settings.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.
- kStatus_SPC_CORELDOVoltageWrong – Core LDO and System LDO do not have same voltage level.

static inline void SPC_DisableLowPowerModeVddCoreGlitchDetect(SPC_Type *base, bool disable)

Disable/Enable VDD Core Glitch Detect in low power mode.

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
  - **true** Disable VDD Core Low Voltage detect;
  - **false** Enable VDD Core Low Voltage detect.

static inline bool SPC_CheckLowPowerModeVddCoreGlitchDetectEnabled(SPC_Type *base)

Check if Glitch detect hardware is enabled in low power mode.

Parameters

- base – SPC peripheral base address.

Returns

Indicate if Glitch detector is enabled.

static inline void SPC_EnableLowPowerModeAnalogModules(SPC_Type *base, uint32_t maskValue)

Enables analog modules in low power modes.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to enable in low power modes, should be OR'ed value of spc_analog_module_control.

static inline void SPC_DisableLowPowerModeAnalogModules(SPC_Type *base, uint32_t maskValue)

Disables analog modules in low power modes.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to disable in low power modes, should be OR'ed value of spc_analog_module_control.

static inline uint32_t SPC_GetLowPowerModeEnabledAnalogModules(SPC_Type *base)

Gets enabled analog modules that enabled in low power modes.

Parameters

- base – SPC peripheral base address.

Returns

The mask of enabled analog modules that enabled in low power modes.

static inline uint32_t SPC_GetVoltageDetectStatusFlag(SPC_Type *base)

Get Voltage Detect Status Flags.

Parameters

- base – SPC peripheral base address.

Returns

Voltage Detect Status Flags. See _spc_voltage_detect_flags for details.

static inline void SPC_ClearVoltageDetectStatusFlag(SPC_Type *base, uint8_t mask)

Clear Voltage Detect Status Flags.

Parameters

- base – SPC peripheral base address.
- mask – The mask of the voltage detect status flags. See _spc_voltage_detect_flags for details.

void SPC_SetCoreVoltageDetectConfig(SPC_Type *base, const *spc_core_voltage_detect_config_t* *config)

Configs CORE voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_core_voltage_detect_config_t structure.

static inline void SPC_LockCoreVoltageDetectResetSetting(SPC_Type *base)

Locks Core voltage detect reset setting.

This function locks core voltage detect reset setting. After invoking this function any configuration of Core voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockCoreVoltageDetectResetSetting(SPC_Type *base)

Unlocks Core voltage detect reset setting.

This function unlocks core voltage detect reset setting. If locks the Core voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

status_t SPC_EnableActiveModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Active mode.

Note: If the CORE_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core LVD. true - Enable Core Low voltage detector in active mode. false - Disable Core Low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Low Power mode.

This function enables/disables the Core Low Voltage Detector. If enabled the Core Low Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core Low voltage detector in low power mode. false - Disable Core Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect in low power mode successfully.

status_t SPC_EnableActiveModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core High Voltage Detector in Active mode.

Note: If the CORE_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in active mode. false - Disable Core High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable Core High Voltage Detect successfully.

status_t SPC_EnableLowPowerModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core High Voltage Detector in Low Power mode.

This function enables/disables the Core High Voltage Detector. If enabled the Core High Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in low power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in low power mode. false - Disable Core High voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core High Voltage Detect in low power mode successfully.

void SPC_SetSystemVDDLowVoltageLevel(SPC_Type *base, *spc_low_voltage_level_select_t* level)

Set system VDD Low-voltage level selection.

This function selects the system VDD low-voltage level. Changing system VDD low-voltage level must be done after disabling the System VDD low voltage reset and interrupt.

Deprecated:

In latest RM, reserved for all devices, will removed in next release.

Parameters

- base – SPC peripheral base address.
- level – System VDD Low-Voltage level selection.

void SPC_SetSystemVoltageDetectConfig(SPC_Type *base, const *spc_system_voltage_detect_config_t* *config)

Configs SYS voltage detect options.

This function config SYS voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_system_voltage_detect_config_t structure.

static inline void SPC_LockSystemVoltageDetectResetSetting(SPC_Type *base)

Lock System voltage detect reset setting.

This function locks system voltage detect reset setting. After invoking this function any configuration of System Voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockSystemVoltageDetectResetSetting(SPC_Type *base)

Unlock System voltage detect reset setting.

This function unlocks system voltage detect reset setting. If locks the System voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

status_t SPC_EnableActiveModeSystemHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System High Voltage Detector in Active mode.

Note: If the System_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in active mode. false - Disable System High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable System High Voltage Detect successfully.

status_t SPC_EnableActiveModeSystemLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disable the System Low Voltage Detector in Active mode.

Note: If the System_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System LVD. true - Enable System Low voltage detector in active mode. false - Disable System Low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable the System Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeSystemHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System High Voltage Detector in Low Power mode.

Note: If the System_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in low power mode. false - Disable System High voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable System High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeSystemLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System Low Voltage Detector in Low Power mode.

Note: If the System_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System Low voltage detector in low power mode. false - Disable System Low voltage detector in low power mode.

Return values

kStatus_Success – Enables System Low Voltage Detect in low power mode successfully.

void SPC_SetIOVDDLowVoltageLevel(SPC_Type *base, *spc_low_voltage_level_select_t* level)

Set IO VDD Low-Voltage level selection.

This function selects the IO VDD Low-voltage level. Changing IO VDD low-voltage level must be done after disabling the IO VDD low voltage reset and interrupt.

Parameters

- base – SPC peripheral base address.
- level – IO VDD Low-voltage level selection.

void SPC_SetIOVoltageDetectConfig(SPC_Type *base, const *spc_io_voltage_detect_config_t* *config)

Configs IO voltage detect options.

This function config IO voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_voltage_detect_config_t structure.

static inline void SPC_LockIOVoltageDetectResetSetting(SPC_Type *base)

Lock IO Voltage detect reset setting.

This function locks IO voltage detect reset setting. After invoking this function any configuration of system voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockIOVoltageDetectResetSetting(SPC_Type *base)

Unlock IO voltage detect reset setting.

This function unlocks IO voltage detect reset setting. If locks the IO voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

status_t SPC_EnableActiveModeIOHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO High Voltage Detector in Active mode.

Note: If the IO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in active mode. false - Disable IO High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable IO High Voltage Detect successfully.

status_t SPC_EnableActiveModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO Low Voltage Detector in Active mode.

Note: If the IO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in active mode. false - Disable IO Low voltage detector in active mode.

Return values

kStatus_Success – Enable IO Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeIOHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO High Voltage Detector in Low Power mode.

Note: If the IO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in low power mode. false - Disable IO High voltage detector in low power mode.

Return values

kStatus_Success – Enable IO High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO Low Voltage Detector in Low Power mode.

Note: If the IO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in low power mode. false - Disable IO Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable IO Low Voltage Detect in low power mode successfully.

void SPC_SetExternalVoltageDomainsConfig(SPC_Type *base, uint8_t lowPowerIsoMask, uint8_t IsoMask)

Configs external voltage domains.

This function configs external voltage domains isolation.

Parameters

- base – SPC peripheral base address.
- lowPowerIsoMask – The mask of external domains isolate enable during low power mode. Please read the Reference Manual for the Bitmap.
- IsoMask – The mask of external domains isolate. Please read the Reference Manual for the Bitmap.

static inline uint8_t SPC_GetExternalDomainsStatus(SPC_Type *base)

Gets External Domains status.

Parameters

- base – SPC peripheral base address.

Returns

The status of each external domain.

static inline void SPC_EnableCoreLDORegulator(SPC_Type *base, bool enable)

Enable/Disable Core LDO regulator.

Note: The CORE LDO enable bit is write-once.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CORE LDO Regulator. true - Enable CORE LDO Regulator. false - Disable CORE LDO Regulator.

static inline void SPC_PullDownCoreLDORegulator(SPC_Type *base, bool pulldown)

Enable/Disable the CORE LDO Regulator pull down in Deep Power Down.

Note: This function only useful when enabled the CORE LDO Regulator.

Parameters

- base – SPC peripheral base address.
- pulldown – Enable/Disable CORE LDO pulldown in Deep Power Down mode. true - CORE LDO Regulator will discharge in Deep Power Down mode. false - CORE LDO Regulator will not discharge in Deep Power Down mode.

status_t SPC_SetActiveModeCoreLDORegulatorConfig(SPC_Type *base, const *spc_active_mode_core_ldo_option_t* *option)

Configs Core LDO Regulator in Active mode.

Note: The bandgap must be enabled before invoking this function.

Note: To set Core LDO as low drive strength, all HVDs/LVDs must be disabled previously.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_core_ldo_option_t structure.

Return values

- kStatus_Success – Config Core LDO regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_BandgapModeWrong – Bandgap should be enabled before invoking this function.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – To set Core LDO as low drive strength, all LVDs/HVDs must be disabled before invoking this function.

status_t SPC_SetActiveModeCoreLDORegulatorVoltageLevel(SPC_Type *base, *spc_core_ldo_voltage_level_t* voltageLevel)

Set Core LDO Regulator Voltage level in Active mode.

Note: In active mode, the Core LDO voltage level should only be changed when the Core LDO is in normal drive strength.

Note: Update Core LDO voltage level will set Busy flag, this function return only when busy flag is cleared by hardware

---

Parameters

- base – SPC peripheral base address.
- voltageLevel – Specify the voltage level of CORE LDO Regulator in Active mode, please refer to spc_core_ldo_voltage_level_t.

Return values

- kStatus_SPC_CORELDOVoltageSetFail – The drive strength of Core LDO is not normal.
- kStatus_Success – Set Core LDO regulator voltage level in Active power mode successful.

static inline *spc_core_ldo_voltage_level_t* SPC_GetActiveModeCoreLDOVDDVoltageLevel(SPC_Type *base)

Gets CORE LDO Regulator Voltage level.

This function returns the voltage level of CORE LDO Regulator in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

Voltage level of CORE LDO in type of spc_core_ldo_voltage_level_t enumeration.

status_t SPC_SetActiveModeCoreLDORegulatorDriveStrength(SPC_Type *base, *spc_core_ldo_drive_strength_t* driveStrength)

Set Core LDO VDD Regulator Drive Strength in Active mode.

Parameters

- base – SPC peripheral base address.

- driveStrength – Specify the drive strength of CORE LDO Regulator in Active mode, please refer to spc_core_ldo_drive_strength_t.

Return values

- kStatus_Success – Set Core LDO regulator drive strength in Active power mode successful.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – If any voltage detect enabled, core_ldo's drive strength can not set to low.
- kStatus_SPC_BandgapModeWrong – The selected bandgap mode is not allowed.

static inline *spc_core_ldo_drive_strength_t* SPC_GetActiveModeCoreLDODriveStrength(SPC_Type *base)

Gets CORE LDO VDD Regulator Drive Strength in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

Drive Strength of CORE LDO regulator in Active mode, please refer to spc_core_ldo_drive_strength_t.

status_t SPC_SetLowPowerModeCoreLDORegulatorConfig(SPC_Type *base, const *spc_lowpower_mode_core_ldo_option_t* *option)

Configs CORE LDO Regulator in low power mode.

This function configs CORE LDO Regulator in Low Power mode. If CORE LDO VDD Drive Strength is set to Normal, the CORE LDO VDD regulator voltage level in Active mode must be equal to the voltage level in Low power mode. And the Bandgap must be programmed to select bandgap enabled. Core VDD voltage levels for the Core LDO low power regulator can only be changed when the CORE LDO Drive Strength set as Normal.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_lowpower_mode_core_ldo_option_t structure.

Return values

- kStatus_Success – Config Core LDO regulator in power mode successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.
- #kStatus_SPC_CORELDOVoltageSetFail. – Fail to change Core LDO voltage level.

status_t SPC_SetLowPowerModeCoreLDORegulatorVoltageLevel(SPC_Type *base, *spc_core_ldo_voltage_level_t* voltageLevel)

Set Core LDO VDD Regulator Voltage level in Low power mode.

Note: If CORE LDO's drive strength is set to Normal, the CORE LDO VDD regulator voltage in active mode and low power mode must be same.

Note: Voltage level for the CORE LDO in low power mode can only be changed when the CORE LDO Drive Strength set as Normal.

Parameters

- base – SPC peripheral base address.
- voltageLevel – Voltage level of CORE LDO Regulator in Low power mode, please refer to spc_core_ldo_voltage_level_t.

Return values

- kStatus_SPC_CORELDOVoltageWrong – Voltage level in active mode and low power mode is not same.
- kStatus_Success – Set Core LDO regulator voltage level in Low power mode successful.
- kStatus_SPC_CORELDOVoltageSetFail – Fail to update voltage level because drive strength is incorrect.

static inline *spc_core_ldo_voltage_level_t* SPC_GetLowPowerCoreLDOVDDVoltageLevel(SPC_Type *base)

Gets the CORE LDO VDD Regulator Voltage Level for Low Power modes.

Parameters

- base – SPC peripheral base address.

Returns
The CORE LDO VDD Regulator's voltage level.

status_t SPC_SetLowPowerModeCoreLDORegulatorDriveStrength(SPC_Type *base, *spc_core_ldo_drive_strength_t* driveStrength)

Set Core LDO VDD Regulator Drive Strength in Low power mode.

Parameters

- base – SPC peripheral base address.
- driveStrength – Specify drive strength of CORE LDO in low power mode.

Return values

- kStatus_SPC_CORELDOLowDriveStrengthIgnore – Some voltage detect enabled, CORE LDO's drive strength can not set as low.
- kStatus_Success – Set Core LDO regulator drive strength in Low power mode successful.
- kStatus_SPC_BandgapModeWrong – Bandgap is disabled when attempt to set CORE LDO work as normal drive strength.

static inline *spc_core_ldo_drive_strength_t* SPC_GetLowPowerCoreLDOVDDDriveStrength(SPC_Type *base)

Gets CORE LDO VDD Drive Strength for Low Power modes.

Parameters

- base – SPC peripheral base address.

Returns
The CORE LDO's VDD Drive Strength.

static inline void SPC_EnableSystemLDORegulator(SPC_Type *base, bool enable)

Enable/Disable System LDO regulator.

Note: The SYSTEM LDO enable bit is write-once.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System LDO Regulator. true - Enable System LDO Regulator. false - Disable System LDO Regulator.

static inline void SPC_EnableSystemLDOSinkFeature(SPC_Type *base, bool sink)

Enable/Disable current sink feature of System LDO Regulator.

Parameters

- base – SPC peripheral base address.
- sink – Enable/Disable current sink feature. true - Enable current sink feature of System LDO Regulator. false - Disable current sink feature of System LDO Regulator.

status_t SPC_SetActiveModeSystemLDORegulatorConfig(SPC_Type *base, const *spc_active_mode_sys_ldo_option_t* *option)

Configs System LDO VDD Regulator in Active mode.

Note: If System LDO VDD Drive Strength is set to Normal, the Bandgap mode in Active mode must be programmed to a value that enables the bandgap.

Note: If any voltage detects are kept enabled, configuration to set System LDO VDD drive strength to low will be ignored.

Note: If select System LDO VDD Regulator voltage level to Over Drive Voltage, the Drive Strength of System LDO VDD Regulator must be set to Normal otherwise the regulator Drive Strength will be forced to Normal.

Note: If select System LDO VDD Regulator voltage level to Over Drive Voltage, the High voltage detect must be disabled. Otherwise it will be fail to regulator to Over Drive Voltage.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_sys_ldo_option_t structure.

Return values

- kStatus_Success – Config System LDO regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.

- kStatus_SPC_BandgapModeWrong – The bandgap is not enabled before invoking this function.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – HVD of System VDD is not disable before setting to Over Drive voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set System LDO VDD regulator's driver strength to Low will be ignored.

status_t SPC_SetActiveModeSystemLDORegulatorVoltageLevel(SPC_Type *base, *spc_sys_ldo_voltage_level_t* voltageLevel)

Set System LDO Regulator voltage level in Active mode.

Note: The system LDO regulator can only operate at the overdrive voltage level for a limited amount of time for the life of chip.

Parameters

- base – SPC peripheral base address.
- voltageLevel – Specify the voltage level of System LDO Regulator in Active mode.

Return values

- kStatus_Success – Set System LDO Regulator voltage level in Active mode successfully.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Must disable system LDO high voltage detector before specifing overdrive voltage.

static inline *spc_sys_ldo_voltage_level_t* SPC_GetActiveModeSystemLDORegulatorVoltageLevel(SPC_Type *base)

Get System LDO Regulator voltage level in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

System LDO Regulator voltage level in Active mode, please refer to spc_sys_ldo_voltage_level_t.

status_t SPC_SetActiveModeSystemLDORegulatorDriveStrength(SPC_Type *base, *spc_sys_ldo_drive_strength_t* driveStrength)

Set System LDO Regulator Drive Strength in Active mode.

Parameters

- base – SPC peripheral base address.
- driveStrength – Specify the drive strength of System LDO Regulator in Active mode.

Return values

- kStatus_Success – Set System LDO Regulator drive strength in Active mode successfully.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Attempt to specify low drive strength is ignored due to any voltage detect feature is enabled in active mode.

- kStatus_SPC_BandgapModeWrong – Bandgap mode in Active mode must be programmed to a value that enables the bandgap if attempt to specify normal drive strength.

static inline *spc_sys_ldo_drive_strength_t* SPC_GetActiveModeSystemLDORegulatorDriveStrength(SPC_Type *base)

Get System LDO Regulator Drive Strength in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

System LDO regulator drive strength in Active mode, please refer to spc_sys_ldo_drive_strength_t.

status_t SPC_SetLowPowerModeSystemLDORegulatorConfig(SPC_Type *base, const *spc_lowpower_mode_sys_ldo_option_t* *option)

Configs System LDO regulator in low power modes.

This function configs System LDO regulator in low power modes. If System LDO VDD Regulator Drive strength is set to normal, bandgap mode in low power mode must be programmed to a value that enables the Bandgap. If any High voltage detectors or Low Voltage detectors are kept enabled, configuration to set System LDO Regulator drive strength as Low will be ignored.

Parameters

- base – SPC peripheral base address.
- option – Pointer to spc_lowpower_mode_sys_ldo_option_t structure.

Return values

- kStatus_Success – Config System LDO regulator in Low Power Mode successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.

status_t SPC_SetLowPowerModeSystemLDORegulatorDriveStrength(SPC_Type *base, *spc_sys_ldo_drive_strength_t* driveStrength)

Set System LDO Regulator drive strength in Low Power Mode.

Parameters

- base – SPC peripheral base address.
- driveStrength – Specify the drive strength of System LDO Regulator in Low Power Mode.

Return values

- kStatus_Success – Set System LDO Regulator drive strength in Low Power Mode successfully.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Attempt to specify low drive strength is ignored due to any voltage detect feature is enabled in low power mode.
- kStatus_SPC_BandgapModeWrong – Bandgap mode in low power mode must be programmed to a value that enables the bandgap if attempt to specify normal drive strength.

static inline *spc_sys_ldo_drive_strength_t* SPC_GetLowPowerModeSystemLDORegulatorDriveStrength(SPC_Type *base)

Get System LDO Regulator drive strength in Low Power Mode.

Parameters

- base – SPC peripheral base address.

Returns

System LDO regulator drive strength in Low Power Mode, please refer to spc_sys_ldo_drive_strength_t.

static inline void SPC_EnableDCDCRegulator(SPC_Type *base, bool enable)

Enable/Disable DCDC Regulator.

Note: The DCDC enable bit is write-once, settings only reset after a POR, LVD, or HVD event.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable DCDC Regulator. true - Enable DCDC Regulator. false - Disable DCDC Regulator.

void SPC_SetDCDCBurstConfig(SPC_Type *base, *spc_dcdc_burst_config_t* *config)

Config DCDC Burst options.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_dcdc_burst_config_t structure.

static inline void SPC_TriggerDCDCBurstRequest(SPC_Type *base)

Trigger a software burst request to DCDC.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_CheckDCDCBurstAck(SPC_Type *base)

Check if burst acknowlege flag is asserted.

Parameters

- base – SPC peripheral base address.

Return values

- false – DCDC burst not complete.
- true – DCDC burst complete.

static inline void SPC_ClearDCDCBurstAckFlag(SPC_Type *base)

Clear DCDC busrt acknowledge flag.

Parameters

- base – SPC periphral base address.

void SPC_SetDCDCRefreshCount(SPC_Type *base, uint16_t count)

Set the count value of the reference clock to configure the period of DCDC not active.

Note: This function is only useful when DCDC's drive strength is set as pulse refresh.

Note: The pulse duration(time between on and off) is: reference clock period * (count + 2).

Parameters

- base – SPC peripheral base address.
- count – The count value, 16 bit width.

static inline void SPC_EnableDCDCBleedResistor(SPC_Type *base, bool enable)

Enable a bleed resistor to discharge DCDC output when DCDC is disabled.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable bleed resistor.

status_t SPC_SetActiveModeDCDCRegulatorConfig(SPC_Type *base, const *spc_active_mode_dcdc_option_t* *option)

Configs DCDC_CORE Regulator in Active mode.

Note: When changing the DCDC output voltage level, take care to change the CORE LDO voltage level.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_dcdc_option_t structure.

Return values

- kStatus_Success – Config DCDC regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_BandgapModeWrong – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

static inline void SPC_SetActiveModeDCDCRegulatorVoltageLevel(SPC_Type *base, *spc_dcdc_voltage_level_t* voltageLevel)

Set DCDC_CORE Regulator voltage level in Active mode.

Note: When changing the DCDC output voltage level, take care to change the CORE LDO voltage level.

Parameters

- base – SPC peripheral base address.
- voltageLevel – Specify the DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

static inline *spc_dcdc_voltage_level_t* SPC_GetActiveModeDCDCRegulatorVoltageLevel(SPC_Type *base)

Get DCDC_CORE Regulator voltage level in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

status_t SPC_SetActiveModeDCDCRegulatorDriveStrength(SPC_Type *base, *spc_dcdc_drive_strength_t* driveStrength)

Set DCDC_CORE Regulator drive strength in Active mode.

Note: To set DCDC drive strength as Normal, the bandgap must be enabled.

Parameters

- base – SPC peripheral base address.
- driveStrength – Specify the DCDC_CORE regulator drive strength, please refer to spc_dcdc_drive_strength_t.

Return values

- kStatus_Success – Set DCDC_CORE Regulator drive strength in Active mode successfully.
- kStatus_SPC_BandgapModeWrong – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

static inline *spc_dcdc_drive_strength_t* SPC_GetActiveModeDCDCRegulatorDriveStrength(SPC_Type *base)

Get DCDC_CORE Regulator drive strength in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator drive strength, please refer to spc_dcdc_drive_strength_t.

status_t SPC_SetLowPowerModeDCDCRegulatorConfig(SPC_Type *base, const *spc_lowpower_mode_dcdc_option_t* *option)

Configs DCDC_CORE Regulator in Low power modes.

Note: If DCDC_CORE Drive Strength is set to Normal, the Bandgap mode in Low Power mode must be programmed to a value that enables the Bandgap.

Note: In Deep Power Down mode, DCDC regulator is always turned off.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_lowpower_mode_dcdc_option_t structure.

Return values

- kStatus_Success – Config DCDC regulator in low power mode successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.

- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in Low Power mode is wrong.

status_t SPC_SetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type *base, *spc_dcdc_drive_strength_t* driveStrength)

Set DCDC_CORE Regulator drive strength in Low power mode.

Note: To set drive strength as normal, the bandgap must be enabled.

Parameters

- base – SPC peripheral base address.
- driveStrength – Specify the DCDC_CORE Regulator drive strength, please refer to spc_dcdc_drive_strength_t.

Return values

- kStatus_Success – Set DCDC_CORE Regulator drive strength in Low power mode successfully.
- kStatus_SPC_BandgapModeWrong – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

static inline *spc_dcdc_drive_strength_t* SPC_GetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type *base)

Get DCDC_CORE Regulator drive strength in Low power mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator drive strength, please refer to spc_dcdc_drive_strength_t.

static inline void SPC_SetLowPowerModeDCDCRegulatorVoltageLevel(SPC_Type *base, *spc_dcdc_voltage_level_t* voltageLevel)

Set DCDC_CORE Regulator voltage level in Low power mode.

a. Configure ACTIVE_CFG[DCDC_VDD_LVL] to same level programmed in #1.

Note: To change DCDC level in Low-Power mode:

a. Configure LP_CFG[DCDC_VDD_LVL] to desired level;

b. Configure LP_CFG[DCDC_VDD_DS] to low driver strength;

Note: After invoking this function, the voltage level in active mode(wakeup from low power modes) also changed, if it is necessary, please invoke SPC_SetActiveModeDCDCRegulatorVoltageLevel() to change to desried voltage level.

Parameters

- base – SPC peripheral base address.

- voltageLevel – Specify the DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

static inline *spc_dcdc_voltage_level_t* SPC_GetLowPowerModeDCDCRegulatorVoltageLevel(SPC_Type *base)

Get DCDC_CORE Regulator voltage level in Low power mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

FSL_SPC_DRIVER_VERSION

SPC driver version 2.10.0.

SPC status enumeration.

Note: Some device(such as MCXA family) do not equip DCDC or System LDO, please refer to the reference manual to check.

Values:

enumerator kStatus_SPC_Busy

The SPC instance is busy executing any type of power mode transition.

enumerator kStatus_SPC_DCDCLowDriveStrengthIgnore

DCDC Low drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_DCDCPulseRefreshModeIgnore

DCDC Pulse Refresh Mode drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_SYSLDOOverDriveVoltageFail

SYS LDO regulate to Over drive voltage failed for SYS LDO HVD must be disabled.

enumerator kStatus_SPC_SYSLDOLowDriveStrengthIgnore

SYS LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_CORELDOLowDriveStrengthIgnore

CORE LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_BandgapModeWrong

Selected Bandgap Mode wrong.

enumerator kStatus_SPC_CORELDOVoltageWrong

Core LDO voltage is wrong.

enumerator kStatus_SPC_CORELDOVoltageSetFail

Core LDO voltage set fail.

enumerator kStatus_SPC_CORELDOVoltageDetectWrong

Settings of CORE_LDO voltage detection is not allowed.

enumerator kStatus_SPC_DCDCCoreLdoVoltageMisMatch

Target voltage level of DCDC not equal to CORE_LDO.

enum _spc_voltage_detect_flags

Voltage Detect Status Flags.

Values:

enumerator kSPC_IOVDDHighVoltageDetectFlag
IO VDD High-Voltage detect flag.

enumerator kSPC_IOVDDLowVoltageDetectFlag
IO VDD Low-Voltage detect flag.

enumerator kSPC_SystemVDDHighVoltageDetectFlag
System VDD High-Voltage detect flag.

enumerator kSPC_SystemVDDLowVoltageDetectFlag
System VDD Low-Voltage detect flag.

enumerator kSPC_CoreVDDHighVoltageDetectFlag
Core VDD High-Voltage detect flag.

enumerator kSPC_CoreVDDLowVoltageDetectFlag
Core VDD Low-Voltage detect flag.

enum _spc_power_domains
SPC power domain isolation status.

Note: Some devices(such as MCXA family) do not contain WAKE Power Domain, please refer to the reference manual to check.

Values:

enumerator kSPC_MAINPowerDomainRetain
Peripherals and IO pads retain in MAIN Power Domain.

enumerator kSPC_WAKEPowerDomainRetain
Peripherals and IO pads retain in WAKE Power Domain.

enum _spc_analog_module_control
The enumeration of all analog module that can be controlled by SPC in active or low-power modes.

Note: Enumerations may not suitable for all devices, please check the specific device's RM for supported analog modules.

Values:

enumerator kSPC_controlVref
Enable/disable VREF in active or low-power modes.

enumerator kSPC_controlUsb3vDet
Enable/disable USB3V_Det in active or low-power modes.

enumerator kSPC_controlDac0
Enable/disable DAC0 in active or low-power modes.

enumerator kSPC_controlDac1
Enable/disable DAC1 in active or low-power modes.

enumerator kSPC_controlDac2
Enable/disable DAC2 in active or low-power modes.

enumerator kSPC_controlOpamp0
Enable/disable OPAMP0 in active or low-power modes.

enumerator kSPC_controlOpamp1
Enable/disable OPAMP1 in active or low-power modes.

enumerator kSPC_controlOpamp2
Enable/disable OPAMP2 in active or low-power modes.

enumerator kSPC_controlOpamp3
Enable/disable OPAMP3 in active or low-power modes.

enumerator kSPC_controlCmp0
Enable/disable CMP0 in active or low-power modes.

enumerator kSPC_controlCmp1
Enable/disable CMP1 in active or low-power modes.

enumerator kSPC_controlCmp2
Enable/disable CMP2 in active or low-power modes.

enumerator kSPC_controlCmp0Dac
Enable/disable CMP0_DAC in active or low-power modes.

enumerator kSPC_controlCmp1Dac
Enable/disable CMP1_DAC in active or low-power modes.

enumerator kSPC_controlCmp2Dac
Enable/disable CMP2_DAC in active or low-power modes.

enumerator kSPC_controlAllModules
Enable/disable all modules in active or low-power modes.

enum _spc_power_domain_id
The enumeration of spc power domain, the connected power domain is chip specfic, please refer to chip's RM for details.

Values:

enumerator kSPC_PowerDomain0
Power domain0, the connected power domain is chip specific.

enumerator kSPC_PowerDomain1
Power domain1, the connected power domain is chip specific.

enum _spc_power_domain_low_power_mode
The enumeration of Power domain's low power mode.

Values:

enumerator kSPC_SleepWithSYSClockRunning
Power domain request SLEEP mode with SYS clock running.

enumerator kSPC_DeepSleepWithSysClockOff
Power domain request deep sleep mode with system clock off.

enumerator kSPC_PowerDownWithSysClockOff
Power domain request power down mode with system clock off.

enumerator kSPC_DeepPowerDownWithSysClockOff
Power domain request deep power down mode with system clock off.

enum _spc_lowPower_request_pin_polarity
SPC low power request output pin polarity.

Values:

enumerator kSPC_HighTruePolarity
Control the High Polarity of the Low Power Reqest Pin.

enumerator kSPC_LowTruePolarity
Control the Low Polarity of the Low Power Reqest Pin.

enum _spc_lowPower_request_output_override
SPC low power request output override.

Values:

enumerator kSPC_LowPowerRequestNotForced
Not Forced.

enumerator kSPC_LowPowerRequestReserved
Reserved.

enumerator kSPC_LowPowerRequestForcedLow
Forced Low (Ignore LowPower request output polarity setting.)

enumerator kSPC_LowPowerRequestForcedHigh
Forced High (Ignore LowPower request output polarity setting.)

enum _spc_bandgap_mode
SPC Bandgap mode enumeration in Active mode or Low Power mode.

Values:

enumerator kSPC_BandgapDisabled
Bandgap disabled.

enumerator kSPC_BandgapEnabledBufferDisabled
Bandgap enabled with Buffer disabled.

enumerator kSPC_BandgapEnabledBufferEnabled
Bandgap enabled with Buffer enabled.

enumerator kSPC_BandgapReserved
Reserved.

enum _spc_dcdc_voltage_level
DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

Note: kSPC_DCDC_RetentionVoltage not supported for all power modes.

Values:

enumerator kSPC_DCDC_RetentionVoltage
DCDC_CORE Regulator regulate to retention Voltage(Only supportedin low power modes)

enumerator kSPC_DCDC_MidVoltage
DCDC_CORE Regulator regulate to Mid Voltage(1.0V).

enumerator kSPC_DCDC_NormalVoltage
DCDC_CORE Regulator regulate to Normal Voltage(1.1V).

enumerator kSPC_DCDC_OverdriveVoltage
DCDC_CORE Regulator regulate to Safe-Mode Voltage(1.2V).

enum _spc_dcdc_drive_strength

DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

Note: Different drive strength differ in these DCDC characterstics: Maximum load current Quiescent current Transient response.

Values:

enumerator kSPC_DCDC_PulseRefreshMode

DCDC_CORE Regulator Drive Strength set to Pulse Refresh Mode, This enum member is only useful for Low Power Mode config, please note that pluse refresh mode is invalid in SLEEP mode.

enumerator kSPC_DCDC_LowDriveStrength

DCDC_CORE regulator Drive Strength set to low.

enumerator kSPC_DCDC_NormalDriveStrength

DCDC_CORE regulator Drive Strength set to Normal.

enum _spc_sys_ldo_voltage_level

SYS LDO regulator voltage level enumeration in Active mode.

Values:

enumerator kSPC_SysLDO_NormalVoltage

SYS LDO VDD Regulator regulate to Normal Voltage(1.8V).

enumerator kSPC_SysLDO_OverDriveVoltage

SYS LDO VDD Regulator regulate to Over Drive Voltage(2.5V).

enum _spc_sys_ldo_drive_strength

SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.

Values:

enumerator kSPC_SysLDO_LowDriveStrength

SYS LDO VDD regulator Drive Strength set to low.

enumerator kSPC_SysLDO_NormalDriveStrength

SYS LDO VDD regulator Drive Strength set to Normal.

enum _spc_core_ldo_voltage_level

Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

Values:

enumerator kSPC_CoreLDO_UnderDriveVoltage

Deprecated:

, to align with description of latest RM, please use kSPC_Core_LDO_RetentionVoltage as instead.

enumerator kSPC_Core_LDO_RetentionVoltage

Core LDO VDD regulator regulate to retention voltage, please note that only useful in low power modes and not all devices support this options please refer to devices' RM for details.

enumerator kSPC_CoreLDO_MidDriveVoltage

Core LDO VDD regulator regulate to Mid Drive Voltage.

enumerator kSPC_CoreLDO_NormalVoltage

Core LDO VDD regulator regulate to Normal Voltage.

enumerator kSPC_CoreLDO_OverDriveVoltage
Core LDO VDD regulator regulate to overdrive Voltage.

enum _spc_core_ldo_drive_strength
CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

Values:

enumerator kSPC_CoreLDO_LowDriveStrength
Core LDO VDD regulator Drive Strength set to low.

enumerator kSPC_CoreLDO_NormalDriveStrength
Core LDO VDD regulator Drive Strength set to Normal.

enum _spc_low_voltage_level_select
IO VDD Low-Voltage Level Select.

Values:

enumerator kSPC_LowVoltageNormalLevel

Deprecated:
, please use kSPC_LowVoltageHighRange as instead.

enumerator kSPC_LowVoltageSafeLevel

Deprecated:
, please use kSPC_LowVoltageLowRange as instead.

enumerator kSPC_LowVoltageHighRange
High range LVD threshold.

enumerator kSPC_LowVoltageLowRange
Low range LVD threshold.

enum _spc_vdd_core_glitch_ripple_counter_select
Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

Values:

enumerator kSPC_selectBit0Of4bitRippleCounter
Select bit-0 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit1Of4bitRippleCounter
Select bit-1 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit2Of4bitRippleCounter
Select bit-2 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit3Of4bitRippleCounter
Select bit-3 of 4-bit Ripple Counter to detect glitch on VDD Core.

enum _spc_sram_operate_voltage
The list of the operating voltage for the SRAM's read/write timing margin.

Values:

enumerator kSPC_sramOperateAt1P0V
SRAM configured for 1.0V operation.

enumerator kSPC_sramOperateAt1P1V
SRAM configured for 1.1V operation.

enumerator kSPC_sramOperateAt1P2V
SRAM configured for 1.2V operation.

typedef enum *_spc_power_domain_id* spc_power_domain_id_t

The enumeration of spc power domain, the connected power domain is chip specfic, please refer to chip's RM for details.

typedef enum *_spc_power_domain_low_power_mode* spc_power_domain_low_power_mode_t

The enumeration of Power domain's low power mode.

typedef enum *_spc_lowPower_request_pin_polarity* spc_lowpower_request_pin_polarity_t

SPC low power request output pin polarity.

typedef enum *_spc_lowPower_request_output_override* spc_lowpower_request_output_override_t

SPC low power request output override.

typedef enum *_spc_bandgap_mode* spc_bandgap_mode_t

SPC Bandgap mode enumeration in Active mode or Low Power mode.

typedef enum *_spc_dcdc_voltage_level* spc_dcdc_voltage_level_t

DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

Note: kSPC_DCDC_RetentionVoltage not supported for all power modes.

---

typedef enum *_spc_dcdc_drive_strength* spc_dcdc_drive_strength_t

DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

Note: Different drive strength differ in these DCDC characterstics: Maximum load current Quiescent current Transient response.

---

typedef enum *_spc_sys_ldo_voltage_level* spc_sys_ldo_voltage_level_t

SYS LDO regulator voltage level enumeration in Active mode.

typedef enum *_spc_sys_ldo_drive_strength* spc_sys_ldo_drive_strength_t

SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.

typedef enum *_spc_core_ldo_voltage_level* spc_core_ldo_voltage_level_t

Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

typedef enum *_spc_core_ldo_drive_strength* spc_core_ldo_drive_strength_t

CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

typedef enum *_spc_low_voltage_level_select* spc_low_voltage_level_select_t

IO VDD Low-Voltage Level Select.

typedef enum *_spc_vdd_core_glitch_ripple_counter_select*
spc_vdd_core_glitch_ripple_counter_select_t

Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

typedef enum *_spc_sram_operate_voltage* spc_sram_operate_voltage_t

The list of the operating voltage for the SRAM's read/write timing margin.

typedef struct *_spc_vdd_core_glitch_detector_config* spc_vdd_core_glitch_detector_config_t

The configuration of VDD Core glitch detector.

typedef struct *_spc_sram_voltage_config* spc_sram_voltage_config_t

typedef struct *_spc_lowpower_request_config* spc_lowpower_request_config_t

Low Power Request output pin configuration.

typedef struct *_spc_active_mode_core_ldo_option* spc_active_mode_core_ldo_option_t

Core LDO regulator options in Active mode.

typedef struct *_spc_active_mode_sys_ldo_option* spc_active_mode_sys_ldo_option_t
System LDO regulator options in Active mode.

typedef struct *_spc_active_mode_dcdc_option* spc_active_mode_dcdc_option_t
DCDC regulator options in Active mode.

typedef struct *_spc_lowpower_mode_core_ldo_option* spc_lowpower_mode_core_ldo_option_t
Core LDO regulator options in Low Power mode.

typedef struct *_spc_lowpower_mode_sys_ldo_option* spc_lowpower_mode_sys_ldo_option_t
System LDO regulator options in Low Power mode.

typedef struct *_spc_lowpower_mode_dcdc_option* spc_lowpower_mode_dcdc_option_t
DCDC regulator options in Low Power mode.

typedef struct *_spc_dcdc_burst_config* spc_dcdc_burst_config_t
DCDC Burst configuration.

Deprecated:
Do not recommend to use this structure.

typedef struct *_spc_voltage_detect_option* spc_voltage_detect_option_t
CORE/SYS/IO VDD Voltage Detect options.

typedef struct *_spc_core_voltage_detect_config* spc_core_voltage_detect_config_t
Core Voltage Detect configuration.

typedef struct *_spc_system_voltage_detect_config* spc_system_voltage_detect_config_t
System Voltage Detect Configuration.

typedef struct *_spc_io_voltage_detect_config* spc_io_voltage_detect_config_t
IO Voltage Detect Configuration.

typedef struct *_spc_active_mode_regulators_config* spc_active_mode_regulators_config_t
Active mode configuration.

typedef struct *_spc_lowpower_mode_regulators_config* spc_lowpower_mode_regulators_config_t
Low Power Mode configuration.

SPC_EVD_CFG_REG_EVDISO_SHIFT

SPC_EVD_CFG_REG_EVDLPISO_SHIFT

SPC_EVD_CFG_REG_EVDSTAT_SHIFT

SPC_EVD_CFG_REG_EVDISO(**x**)

SPC_EVD_CFG_REG_EVDLPISO(**x**)

SPC_EVD_CFG_REG_EVDSTAT(**x**)

struct _spc_vdd_core_glitch_detector_config
#include <fsl_spc.h> The configuration of VDD Core glitch detector.

Public Members

spc_vdd_core_glitch_ripple_counter_select_t rippleCounterSelect
Used to set ripple counter.

uint8_t resetTimeoutValue
The timeout value used to reset glitch detect/compare logic after an initial glitch is detected.

bool enableReset
Used to enable/disable POR/LVD reset that caused by CORE VDD glitch detect error.

bool enableInterrupt
Used to enable/disable hardware interrupt if CORE VDD glitch detect error.

struct _spc_sram_voltage_config
#include <fsl_spc.h>

Public Members

spc_sram_operate_voltage_t operateVoltage
Specifies the operating voltage for the SRAM's read/write timing margin.

bool requestVoltageUpdate
Used to control whether request an SRAM trim value change.

struct _spc_lowpower_request_config
#include <fsl_spc.h> Low Power Request output pin configuration.

Public Members

bool enable
Low Power Request Output enable.

spc_lowpower_request_pin_polarity_t polarity
Low Power Request Output pin polarity select.

spc_lowpower_request_output_override_t override
Low Power Request Output Override.

struct _spc_active_mode_core_ldo_option
#include <fsl_spc.h> Core LDO regulator options in Active mode.

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage
Core LDO Regulator Voltage Level selection in Active mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength
Core LDO Regulator Drive Strength selection in Active mode

struct _spc_active_mode_sys_ldo_option
#include <fsl_spc.h> System LDO regulator options in Active mode.

Public Members

spc_sys_ldo_voltage_level_t SysLDOVoltage
System LDO Regulator Voltage Level selection in Active mode.

spc_sys_ldo_drive_strength_t SysLDODriveStrength
System LDO Regulator Drive Strength selection in Active mode.

struct _spc_active_mode_dcdc_option
#include <fsl_spc.h> DCDC regulator options in Active mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage
DCDC Regulator Voltage Level selection in Active mode.

spc_dcdc_drive_strength_t DCDCDriveStrength
DCDC_CORE Regulator Drive Strength selection in Active mode.

struct _spc_lowpower_mode_core_ldo_option
#include <fsl_spc.h> Core LDO regulator options in Low Power mode.

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage
Core LDO Regulator Voltage Level selection in Low Power mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength
Core LDO Regulator Drive Strength selection in Low Power mode

struct _spc_lowpower_mode_sys_ldo_option
#include <fsl_spc.h> System LDO regulator options in Low Power mode.

Public Members

spc_sys_ldo_drive_strength_t SysLDODriveStrength
System LDO Regulator Drive Strength selection in Low Power mode.

struct _spc_lowpower_mode_dcdc_option
#include <fsl_spc.h> DCDC regulator options in Low Power mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage
DCDC Regulator Voltage Level selection in Low Power mode.

spc_dcdc_drive_strength_t DCDCDriveStrength
DCDC_CORE Regulator Drive Strength selection in Low Power mode.

struct _spc_dcdc_burst_config
#include <fsl_spc.h> DCDC Burst configuration.

Deprecated:
Do not recommend to use this structure.

Public Members

bool sofwareBurstRequest
Enable/Disable DCDC Software Burst Request.

bool externalBurstRequest
Enable/Disable DCDC External Burst Request.

bool stabilizeBurstFreq
Enable/Disable DCDC frequency stabilization.

uint8_t freq
The frequency of the current burst.

struct __spc__voltage__detect__option
#include <fsl_spc.h> CORE/SYS/IO VDD Voltage Detect options.

Public Members

bool HVDInterruptEnable
CORE/SYS/IO VDD High Voltage Detect interrupt enable.

bool HVDResetEnable
CORE/SYS/IO VDD High Voltage Detect reset enable.

bool LVDInterruptEnable
CORE/SYS/IO VDD Low Voltage Detect interrupt enable.

bool LVDResetEnable
CORE/SYS/IO VDD Low Voltage Detect reset enable.

struct __spc__core__voltage__detect__config
#include <fsl_spc.h> Core Voltage Detect configuration.

Public Members

spc_voltage_detect_option_t option
Core VDD Voltage Detect option.

struct __spc__system__voltage__detect__config
#include <fsl_spc.h> System Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option
System VDD Voltage Detect option.

spc_low_voltage_level_select_t level

Deprecated:
, reserved for all devices, will removed in next release.

struct __spc__io__voltage__detect__config
#include <fsl_spc.h> IO Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option
IO VDD Voltage Detect option.

spc_low_voltage_level_select_t level
IO VDD Low-voltage level selection.

struct __spc__active__mode__regulators__config
#include <fsl_spc.h> Active mode configuration.

Public Members

spc_bandgap_mode_t bandgapMode
Specify bandgap mode in active mode.

bool lpBuff
Enable/disable CMP bandgap buffer.

spc_active_mode_dcdc_option_t DCDCOption
Specify DCDC configurations in active mode.

spc_active_mode_sys_ldo_option_t SysLDOOption
Specify System LDO configurations in active mode.

spc_active_mode_core_ldo_option_t CoreLDOOption
Specify Core LDO configurations in active mode.

struct _spc_lowpower_mode_regulators_config
#include <fsl_spc.h> Low Power Mode configuration.

Public Members

bool lpIREF
Enable/disable low power IREF in low power modes.

spc_bandgap_mode_t bandgapMode
Specify bandgap mode in low power modes.

bool lpBuff
Enable/disable CMP bandgap buffer in low power modes.

bool CoreIVS
Enable/disable CORE VDD internal voltage scaling.

spc_lowpower_mode_dcdc_option_t DCDCOption
Specify DCDC configurations in low power modes.

spc_lowpower_mode_sys_ldo_option_t SysLDOOption
Specify system LDO configurations in low power modes.

spc_lowpower_mode_core_ldo_option_t CoreLDOOption
Specify core LDO configurations in low power modes.

2.75 MCX_VBAT: Smart Power Switch

The enumeration of VBAT module status.

Values:

enumerator kStatus_VBAT_Fro16kNotEnabled
Internal 16kHz free running oscillator not enabled.

enumerator kStatus_VBAT_BandgapNotEnabled
Bandgap not enabled.

enumerator kStatus_VBAT_WrongCapacitanceValue
Wrong capacitance for selected oscillator mode.

enumerator kStatus_VBAT_ClockMonitorLocked
Clock monitor locked.

enumerator kStatus_VBAT_OSC32KNotReady
OSC32K not ready.

enumerator kStatus_VBAT_LDONotReady
LDO not ready.

enumerator kStatus_VBAT_TamperLocked
Tamper locked.

enum _vbat_status_flag
The enumeration of VBAT status flags.

Values:

enumerator kVBAT_StatusFlagPORDetect
VBAT domain has been reset

enumerator kVBAT_StatusFlagWakeupPin
A falling edge is detected on the wakeup pin.

enumerator kVBAT_StatusFlagBandgapTimer0
Bandgap Timer0 period reached.

enumerator kVBAT_StatusFlagBandgapTimer1
Bandgap Timer1 period reached.

enumerator kVBAT_StatusFlagLdoReady
LDO is enabled and ready.

enumerator kVBAT_StatusFlagOsc32kReady
OSC32k is enabled and clock is ready.

enumerator kVBAT_StatusFlagInterrupt0Detect
Interrupt 0 asserted.

enumerator kVBAT_StatusFlagInterrupt1Detect
Interrupt 1 asserted.

enumerator kVBAT_StatusFlagInterrupt2Detect
Interrupt 2 asserted.

enumerator kVBAT_StatusFlagInterrupt3Detect
Interrupt 2 asserted.

enum _vbat_interrupt_enable
The enumeration of VBAT interrupt enable.

Values:

enumerator kVBAT_InterruptEnablePORDetect
Enable POR detect interrupt.

enumerator kVBAT_InterruptEnableWakeupPin
Enable the interrupt when a falling edge is detected on the wakeup pin.

enumerator kVBAT_InterruptEnableBandgapTimer0
Enable the interrupt if Bandgap Timer0 period reached.

enumerator kVBAT_InterruptEnableBandgapTimer1
Enable the interrupt if Bandgap Timer1 period reached.

enumerator kVBAT_InterruptEnableLdoReady
Enable LDO ready interrupt.

enumerator kVBAT_InterruptEnableOsc32kReady
Enable OSC32K ready interrupt.

enumerator kVBAT_InterruptEnableInterrupt0
Enable the interrupt0.

enumerator kVBAT_InterruptEnableInterrupt1
Enable the interrupt1.

enumerator kVBAT_InterruptEnableInterrupt2
Enable the interrupt2.

enumerator kVBAT_InterruptEnableInterrupt3
Enable the interrupt3.

enumerator kVBAT_AllInterruptsEnable
Enable all interrupts.

enum _vbat_wakeup_enable
The enumeration of VBAT wakeup enable.

Values:

enumerator kVBAT_WakeupEnablePORDetect
Enable POR detect wakeup.

enumerator kVBAT_WakeupEnableWakeupPin
Enable wakeup feature when a falling edge is detected on the wakeup pin.

enumerator kVBAT_WakeupEnableBandgapTimer0
Enable wakeup feature when bandgap timer0 period reached.

enumerator kVBAT_WakeupEnableBandgapTimer1
Enable wakeup feature when bandgap timer1 period reached.

enumerator kVBAT_WakeupEnableLdoReady
Enable wakeup when LDO ready.

enumerator kVBAT_WakeupEnableOsc32kReady
Enable wakeup when OSC32k ready.

enumerator kVBAT_WakeupEnableInterrupt0
Enable wakeup when interrupt0 asserted.

enumerator kVBAT_WakeupEnableInterrupt1
Enable wakeup when interrupt1 asserted.

enumerator kVBAT_WakeupEnableInterrupt2
Enable wakeup when interrupt2 asserted.

enumerator kVBAT_WakeupEnableInterrupt3
Enable wakeup when interrupt3 asserted.

enumerator kVBAT_AllWakeupsEnable
Enable all wakeup.

enum _vbat_tamper_enable
The enumeration of VBAT tamper enable.

Values:

enumerator kVBAT_TamperEnablePOR
Enable tamper if POR asserted in STATUS register.

enumerator kVBAT_TamperEnableClockDetect
Enable tamper if clock monitor detect an error.

enumerator kVBAT_TamperEnableConfigDetect
Enable tamper if configuration error detected.

enumerator kVBAT_TamperEnableVoltageDetect
Enable tamper if voltage monitor detect an error.

enumerator kVBAT_TamperEnableTemperatureDetect
Enable tamper if temperature monitor detect an error.

enumerator kVBAT_TamperEnableSec0Detect
Enable tamper if security input 0 detect an error.

enum _vbat_bandgap_timer_id
The enumeration of bandgap timer id, VBAT support two bandgap timers.

Values:

enumerator kVBAT_BandgapTimer0
Bandgap Timer0.

enumerator kVBAT_BandgapTimer1
Bandgap Timer1.

enum _vbat_clock_enable
The enumeration of connections for OSC32K/FRO32K output clock to other modules.

Values:

enumerator kVBAT_EnableClockToDomain0
Enable clock to power domain0.

enumerator kVBAT_EnableClockToDomain1
Enable clock to power domain1.

enumerator kVBAT_EnableClockToDomain2
Enable clock to power domain2.

enumerator kVBAT_EnableClockToDomain3
Enable clock to power domain3.

enum _vbat_ram_array
The enumeration of SRAM arrays that controlled by VBAT. .

Values:

enumerator kVBAT_SramArray0
Specify SRAM array0 that controlled by VBAT.

enumerator kVBAT_SramArray1
Specify SRAM array1 that controlled by VBAT.

enumerator kVBAT_SramArray2
Specify SRAM array2 that controlled by VBAT.

enumerator kVBAT_SramArray3
Specify SRAM array3 that controlled by VBAT.

enum _vbat_bandgap_refresh_period

The enumeration of bandgap refresh period.

Values:

enumerator kVBAT_BandgapRefresh7P8125ms

Bandgap refresh every 7.8125ms.

enumerator kVBAT_BandgapRefresh15P625ms

Bandgap refresh every 15.625ms.

enumerator kVBAT_BandgapRefresh31P25ms

Bandgap refresh every 31.25ms.

enumerator kVBAT_BandgapRefresh62P5ms

Bandgap refresh every 62.5ms.

enum _vbat_bandgap_timer0_timeout_period

The enumeration of bandgap timer0 timeout period.

Values:

enumerator kVBAT_BangapTimer0Timeout1s

Bandgap timer0 timerout every 1s.

enumerator kVBAT_BangapTimer0Timeout500ms

Bandgap timer0 timerout every 500ms.

enumerator kVBAT_BangapTimer0Timeout250ms

Bandgap timer0 timerout every 250ms.

enumerator kVBAT_BangapTimer0Timeout125ms

Bandgap timer0 timerout every 125ms.

enumerator kVBAT_BangapTimer0Timeout62P5ms

Bandgap timer0 timerout every 62.5ms.

enumerator kVBAT_BangapTimer0Timeout31P25ms

Bandgap timer0 timerout every 31.25ms.

enum _vbat_osc32k_operate_mode

The enumeration of osc32k operate mode, including Bypass mode, low power switched mode and so on.

Values:

enumerator kVBAT_Osc32kEnabledToTransconductanceMode

Set to transconductance mode.

enumerator kVBAT_Osc32kEnabledToLowPowerBackupMode

Set to low power backup mode.

enumerator kVBAT_Osc32kEnabledToLowPowerSwitchedMode

Set to low power switched mode.

enum _vbat_osc32k_load_capacitance_select

The enumeration of OSC32K load capacitance.

Values:

enumerator kVBAT_Osc32kCrystalLoadCap0pF

Internal capacitance bank is enabled, set the internal capacitance to 0 pF.

enumerator kVBAT_Osc32kCrystalLoadCap2pF
Internal capacitance bank is enabled, set the internal capacitance to 2 pF.

enumerator kVBAT_Osc32kCrystalLoadCap4pF
Internal capacitance bank is enabled, set the internal capacitance to 4 pF.

enumerator kVBAT_Osc32kCrystalLoadCap6pF
Internal capacitance bank is enabled, set the internal capacitance to 6 pF.

enumerator kVBAT_Osc32kCrystalLoadCap8pF
Internal capacitance bank is enabled, set the internal capacitance to 8 pF.

enumerator kVBAT_Osc32kCrystalLoadCap10pF
Internal capacitance bank is enabled, set the internal capacitance to 10 pF.

enumerator kVBAT_Osc32kCrystalLoadCap12pF
Internal capacitance bank is enabled, set the internal capacitance to 12 pF.

enumerator kVBAT_Osc32kCrystalLoadCap14pF
Internal capacitance bank is enabled, set the internal capacitance to 14 pF.

enumerator kVBAT_Osc32kCrystalLoadCap16pF
Internal capacitance bank is enabled, set the internal capacitance to 16 pF.

enumerator kVBAT_Osc32kCrystalLoadCap18pF
Internal capacitance bank is enabled, set the internal capacitance to 18 pF.

enumerator kVBAT_Osc32kCrystalLoadCap20pF
Internal capacitance bank is enabled, set the internal capacitance to 20 pF.

enumerator kVBAT_Osc32kCrystalLoadCap22pF
Internal capacitance bank is enabled, set the internal capacitance to 22 pF.

enumerator kVBAT_Osc32kCrystalLoadCap24pF
Internal capacitance bank is enabled, set the internal capacitance to 24 pF.

enumerator kVBAT_Osc32kCrystalLoadCap26pF
Internal capacitance bank is enabled, set the internal capacitance to 26 pF.

enumerator kVBAT_Osc32kCrystalLoadCap28pF
Internal capacitance bank is enabled, set the internal capacitance to 28 pF.

enumerator kVBAT_Osc32kCrystalLoadCap30pF
Internal capacitance bank is enabled, set the internal capacitance to 30 pF.

enumerator kVBAT_Osc32kCrystalLoadCapBankDisabled
Internal capacitance bank is disabled.

enum _vbat_osc32k_start_up_time
The enumeration of start-up time of the oscillator.

Values:

enumerator kVBAT_Osc32kStartUpTime8Sec
Configure the start-up time as 8 seconds.

enumerator kVBAT_Osc32kStartUpTime4Sec
Configure the start-up time as 4 seconds.

enumerator kVBAT_Osc32kStartUpTime2Sec
Configure the start-up time as 2 seconds.

enumerator kVBAT_Osc32kStartUpTime1Sec
Configure the start-up time as 1 seconds.

enumerator kVBAT_Osc32kStartUpTime0P5Sec
Configure the start-up time as 0.5 seconds.

enumerator kVBAT_Osc32kStartUpTime0P25Sec
Configure the start-up time as 0.25 seconds.

enumerator kVBAT_Osc32kStartUpTime0P125Sec
Configure the start-up time as 0.125 seconds.

enumerator kVBAT_Osc32kStartUpTime0P5MSec
Configure the start-up time as 0.5 milliseconds.

enum _vbat_internal_module_supply
The enumeration of VBAT module supplies.

Values:

enumerator kVBAT_ModuleSuppliedByVddBat
VDD_BAT supplies VBAT modules.

enumerator kVBAT_ModuleSuppliedByVddSys
VDD_SYS supplies VBAT modules.

enum _vbat_clock_monitor_divide_trim
The enumeration of VBAT clock monitor divide trim value.

Values:

enumerator kVBAT_ClockMonitorOperateAt1kHz
Clock monitor operates at 1 kHz.

enumerator kVBAT_ClockMonitorOperateAt64Hz
Clock monitor operates at 64 Hz.

enum _vbat_clock_monitor_freq_trim
The enumeration of VBAT clock monitor frequency trim value used to adjust the clock monitor assert.

Values:

enumerator kVBAT_ClockMonitorAssert2Cycle
Clock monitor assert 2 cycles after expected edge.

enumerator kVBAT_ClockMonitorAssert4Cycle
Clock monitor assert 4 cycles after expected edge.

enumerator kVBAT_ClockMonitorAssert6Cycle
Clock monitor assert 8 cycles after expected edge.

enumerator kVBAT_ClockMonitorAssert8Cycle
Clock monitor assert 8 cycles after expected edge.

typedef enum *_vbat_bandgap_refresh_period* vbat_bandgap_refresh_period_t
The enumeration of bandgap refresh period.

typedef enum *_vbat_bandgap_timer0_timeout_period* vbat_bandgap_timer0_timeout_period_t
The enumeration of bandgap timer0 timeout period.

typedef enum *_vbat_osc32k_operate_mode* vbat_osc32k_operate_mode_t
The enumeration of osc32k operate mode, including Bypass mode, low power switched mode and so on.

typedef enum *_vbat_osc32k_load_capacitance_select* vbat_osc32k_load_capacitance_select_t

The enumeration of OSC32K load capacitance.

typedef enum *_vbat_osc32k_start_up_time* vbat_osc32k_start_up_time_t

The enumeration of start-up time of the oscillator.

typedef enum *_vbat_internal_module_supply* vbat_internal_module_supply_t

The enumeration of VBAT module supplies.

typedef enum *_vbat_clock_monitor_divide_trim* vbat_clock_monitor_divide_trim_t

The enumeration of VBAT clock monitor divide trim value.

typedef enum *_vbat_clock_monitor_freq_trim* vbat_clock_monitor_freq_trim_t

The enumeration of VBAT clock monitor frequency trim value used to adjust the clock monitor assert.

typedef struct *_vbat_fro16k_config* vbat_fro16k_config_t

The structure of internal 16kHz free running oscillator attributes.

typedef struct *_vbat_clock_monitor_config* vbat_clock_monitor_config_t

The structure of internal clock monitor, including divide trim and frequency trim.

typedef struct *_vbat_tamper_config* vbat_tamper_config_t

The structure of Tamper configuration.

FSL_VBAT_DRIVER_VERSION

VBAT driver version 2.5.0.

VBAT_LDORAMC_RET_MASK

VBAT_LDORAMC_RET_SHIFT

VBAT_LDORAMC_RET(x)

kVBAT_EnableClockToVddBat

kVBAT_EnableClockToVddSys

kVBAT_EnableClockToVddWake

kVBAT_EnableClockToVddMain

void VBAT_ConfigFRO16k(VBAT_Type *base, const *vbat_fro16k_config_t* *config)

Configure internal 16kHz free running oscillator, including enabel FRO16k, gate FRO16k output.

Parameters

- base – VBAT peripheral base address.
- config – Pointer to vbat_fro16k_config_t structure.

static inline void VBAT_EnableFRO16k(VBAT_Type *base, bool enable)

Enable/disable internal 16kHz free running oscillator.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable 16kHz FRO.
  - **true** Enable internal 16kHz free running oscillator.
  - **false** Disable internal 16kHz free running oscillator.

static inline bool VBAT_CheckFRO16kEnabled(VBAT_Type *base)

Check if internal 16kHz free running oscillator is enabled.

Parameters

- base – VBAT peripheral base address.

Return values

- true – The internal 16kHz Free running oscillator is enabled.
- false – The internal 16kHz Free running oscillator is enabled.

static inline void VBAT_UngateFRO16k(VBAT_Type *base, uint8_t connectionsMask)

Enable FRO16kHz output clock to selected modules.

Parameters

- base – VBAT peripheral base address.
- connectionsMask – The mask of modules that FRO16k is connected, should be the OR'ed value of vbat_clock_enable_t.

static inline void VBAT_GateFRO16k(VBAT_Type *base, uint8_t connectionsMask)

Disable FRO16kHz output clock to selected modules.

Parameters

- base – VBAT peripheral base address.
- connectionsMask – The OR'ed value of vbat_clock_enable_t.

static inline void VBAT_LockFRO16kSettings(VBAT_Type *base)

Lock settings of internal 16kHz free running oscillator, please note that if locked 16kHz FRO's settings can not be updated until the next POR.

Note: Please note that the operation to ungate/gate FRO 16kHz output clock can not be locked by this function.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT_CheckFRO16kSettingsLocked(VBAT_Type *base)

Check if FRO16K settings are locked.

Parameters

- base – VBAT peripheral base address.

Returns

true in case of FRO16k settings are locked, false in case of FRO16k settings are not locked.

static inline void VBAT_EnableCrystalOsc32k(VBAT_Type *base, bool enable)

Enable/disable 32K Crystal Oscillator.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable 32k Crystal Oscillator:
  - **true** Enable crystal oscillator and polling status register to check clock is ready.
  - **false** Disable crystal oscillator.

static inline void VBAT_BypassCrystalOsc32k(VBAT_Type *base, bool enableBypass)

Bypass 32k crystal oscillator, the clock is still output by oscillator but this clock is the same as clock provided on EXTAL pin.

Note: In bypass mode, oscillator must be enabled; To exit bypass mode, oscillator must be disabled.

Parameters

- base – VBAT peripheral base address.
- enableBypass – Used to enter/exit bypass mode:
  - **true** Enter into bypass mode;
  - **false** Exit bypass mode.

static inline void VBAT_AdjustCrystalOsc32kAmplifierGain(VBAT_Type *base, uint8_t coarse, uint8_t fine)

Adjust 32k crystal oscillator amplifier gain.

Parameters

- base – VBAT peripheral base address.
- coarse – Specify amplifier coarse trim value.
- fine – Specify amplifier fine trim value.

status_t VBAT_SetCrystalOsc32kModeAndLoadCapacitance(VBAT_Type *base, *vbat_osc32k_operate_mode_t* operateMode, *vbat_osc32k_load_capacitance_select_t* xtalCap, *vbat_osc32k_load_capacitance_select_t* extalCap)

Set 32k crystal oscillator mode and load capacitance for the XTAL/EXTAL pin.

Parameters

- base – VBAT peripheral base address.
- operateMode – Specify the crystal oscillator mode, please refer to vbat_osc32k_operate_mode_t.
- xtalCap – Specify the internal capacitance for the XTAL pin from the capacitor bank.
- extalCap – Specify the internal capacitance for the EXTAL pin from the capacitor bank.

Return values

- kStatus_VBAT_WrongCapacitanceValue – The load capacitance value to set is not align with operate mode's requirements.
- kStatus_Success – Success to set operate mode and load capacitance.

static inline void VBAT_TrimCrystalOsc32kStartupTime(VBAT_Type *base, *vbat_osc32k_start_up_time_t* startupTime)

Trim 32k crystal oscillator startup time.

Parameters

- base – VBAT peripheral base address.

- startupTime – Specify the startup time of the oscillator.

static inline void VBAT_SetOsc32kSwitchModeComparatorTrimValue(VBAT_Type *base, uint8_t comparatorTrimValue)

Set crystal oscillator comparator trim value when oscillator is set as low power switch mode.

Parameters

- base – VBAT peripheral base address.
- comparatorTrimValue – Comparator trim value, ranges from 0 to 7.

static inline void VBAT_SetOsc32kSwitchModeDelayTrimValue(VBAT_Type *base, uint8_t delayTrimValue)

Set crystal oscillator delay trim value when oscillator is set as low power switch mode.

Parameters

- base – VBAT peripheral base address.
- delayTrimValue – Delay trim value, ranges from 0 to 15.

static inline void VBAT_SetOsc32kSwitchModeCapacitorTrimValue(VBAT_Type *base, uint8_t capacitorTrimValue)

Set crystal oscillator capacitor trim value when oscillator is set as low power switch mode.

Parameters

- base – VBAT peripheral base address.
- capacitorTrimValue – Capacitor value to trim, ranges from 0 to 3.

static inline void VBAT_LookOsc32kSettings(VBAT_Type *base)

Lock Osc32k settings, after locked all writes to the Oscillator registers are blocked.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_UnlockOsc32kSettings(VBAT_Type *base)

Unlock Osc32k settings.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT_CheckOsc32kSettingsLocked(VBAT_Type *base)

Check if osc32k settings are locked.

Parameters

- base – VBAT peripheral base address.

Returns

true in case of osc32k settings are locked, false in case of osc32k settings are not locked.

static inline void VBAT_UngateOsc32k(VBAT_Type *base, uint8_t connectionsMask)

Enable OSC32k output clock to selected modules.

Parameters

- base – VBAT peripheral base address.
- connectionsMask – The OR'ed value of vbat_clock_enable_t.

static inline void VBAT_GateOsc32k(VBAT_Type *base, uint8_t connectionsMask)

Disable OSC32k output clock to selected modules.

Parameters

- base – VBAT peripheral base address.
- connectionsMask – The OR'ed value of vbat_clock_enable_t.

status_t VBAT_EnableBandgap(VBAT_Type *base, bool enable)

Enable/disable Bandgap.

Note: The FRO16K must be enabled before enabling the bandgap.

Note: This setting can be locked by VBAT_LockRamLdoSettings() function.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap.
  - **true** Enable the bandgap.
  - **false** Disable the bandgap.

Return values

- kStatus_Success – Success to enable/disable the bandgap.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable the bandgap due to FRO16k is not enabled previously.

static inline bool VBAT_CheckBandgapEnabled(VBAT_Type *base)

Check if bandgap is enabled.

Parameters

- base – VBAT peripheral base address.

Return values

- true – The bandgap is enabled.
- false – The bandgap is disabled.

static inline void VBAT_EnableBandgapRefreshMode(VBAT_Type *base, bool enableRefreshMode)

Enable/disable bandgap low power refresh mode.

Note: For lowest power consumption, refresh mode must be enabled.

Note: This setting can be locked by VBAT_LockRamLdoSettings() function.

Parameters

- base – VBAT peripheral base address.
- enableRefreshMode – Used to enable/disable bandgap low power refresh mode.
  - **true** Enable bandgap low power refresh mode.

– **false** Disable bandgap low power refresh mode.

status_t VBAT_EnableBackupSRAMRegulator(VBAT_Type *base, bool enable)

Enable/disable Backup RAM Regulator(RAM_LDO).

Note: This setting can be locked by VBAT_LockRamLdoSettings() function.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable RAM_LDO.
  - **true** Enable backup SRAM regulator.
  - **false** Disable backup SRAM regulator.

Return values

- kStatusSuccess – Success to enable/disable backup SRAM regulator.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable backup SRAM regulator due to FRO16k is not enabled previously.
- kStatus_VBAT_BandgapNotEnabled – Fail to enable backup SRAM regulator due to the bandgap is not enabled previously.

static inline void VBAT_LockRamLdoSettings(VBAT_Type *base)

Lock settings of RAM_LDO, please note that if locked then RAM_LDO's settings can not be updated until the next POR.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT_CheckRamLdoSettingsLocked(VBAT_Type *base)

Check if RAM_LDO settings is locked.

Parameters

- base – VBAT peripheral base address.

Returns

true in case of RAM_LDO settings are locked, false in case of RAM_LDO settings are unlocked.

status_t VBAT_SwitchSRAMPowerByLDOSRAM(VBAT_Type *base)

Switch the SRAM to be powered by LDO_RAM.

Note: This function can be used to switch the SRAM to the VBAT retention supply at any time, but please note that the SRAM must not be accessed during this time.

Note: Invoke this function to switch power supply before switching off external power.

Note: RAM_LDO must be enabled before invoking this function.

Note: To access the SRAM arrays retained by the LDO_RAM, please invoke VBAT_SwitchSRAMPowerBySocSupply(), after external power is switched back on.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatusSuccess – Success to Switch SRAM powered by VBAT.
- kStatus_VBAT_Fro16kNotEnabled – Fail to switch SRAM powered by VBAT due to FRO16K not enabled previously.

static inline void VBAT_SwitchSRAMPowerBySocSupply(VBAT_Type *base)

Switch the RAM to be powered by Soc Supply in software mode.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_PowerOffSRAMsInLowPowerModes(VBAT_Type *base, uint8_t sramMask)

Power off selected SRAM array in low power modes.

Parameters

- base – VBAT peripheral base address.
- sramMask – The mask of SRAM array to power off, should be the OR'ed value of vbat_ram_array_t.

static inline void VBAT_RetainSRAMsInLowPowerModes(VBAT_Type *base, uint8_t sramMask)

Retain selected SRAM array in low power modes.

Parameters

- base – VBAT peripheral base address.
- sramMask – The mask of SRAM array to retain, should be the OR'ed value of vbat_ram_array_t.

static inline void VBAT_EnableSRAMIsolation(VBAT_Type *base, bool enable)

Enable/disable SRAM isolation.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable SRAM violation.
  - **true** SRAM will be isolated.
  - **false** SRAM state follows the SoC power modes.

status_t VBAT_EnableBandgapTimer(VBAT_Type *base, bool enable, uint8_t timerIdMask)

Enable/disable Bandgap timer.

Note: The bandgap timer is available when the bandgap is enabled and are clocked by the FRO16k.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap timer.
- timerIdMask – The mask of bandgap timer Id, should be the OR'ed value of vbat_bandgap_timer_id_t.

Return values

- kStatus_Success – Success to enable/disable selected bandgap timer.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable/disable selected bandgap timer due to FRO16k not enabled previously.
- kStatus_VBAT_BandgapNotEnabled – Fail to enable/disable selected bandgap timer due to bandgap not enabled previously.

void VBAT_SetBandgapTimer0TimeoutValue(VBAT_Type *base, *vbat_bandgap_timer0_timeout_period_t* timeoutPeriod)

Set bandgap timer0 timeout value.

Note: The timeout value can only be changed when the timer is disabled.

Parameters

- base – VBAT peripheral base address.
- timeoutPeriod – Bandgap timer timeout value, please refer to vbat_bandgap_timer0_timeout_period_t.

void VBAT_SetBandgapTimer1TimeoutValue(VBAT_Type *base, uint32_t timeoutPeriod)

Set bandgap timer1 timeout value.

Note: The timeout value can only be changed when the timer is disabled.

Parameters

- base – VBAT peripheral base address.
- timeoutPeriod – The bandgap timerout 1 period, in number of seconds, ranging from 0 to 65535s.

static inline void VBAT_SwitchVBATModuleSupplyActiveMode(VBAT_Type *base, *vbat_internal_module_supply_t* supply)

Control the VBAT internal switch in active mode, VBAT modules can be suppiled by VDD_BAT and VDD_SYS.

Parameters

- base – VBAT peripheral base address.
- supply – Used to control the VBAT internal switch.

static inline *vbat_internal_module_supply_t* VBAT_GetVBATModuleSupply(VBAT_Type *base)

Get VBAT module supply in active mode.

Parameters

- base – VBAT peripheral base address.

Returns

VDD_SYS supplies VBAT modules or VDD_BAT supplies VBAT modules, in type of vbat_internal_module_supply_t.

static inline void VBAT_SwitchVBATModuleSupplyLowPowerMode(VBAT_Type *base, *vbat_internal_module_supply_t* supply)

Control the VBAT internal switch in low power modes.

Note: If VBAT modules are supplied by VDD_SYS in low power modes, VBAT module will also supplied by VDD_SYS in active mode.

Parameters

- base – VBAT peripheral base address.
- supply – Used to specify which voltage input supply VBAT modules in low power mode.

static inline void VBAT_LockSwitchControl(VBAT_Type *base)

Lock switch control, if locked all writes to the switch registers will be blocked.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_UnlockSwitchControl(VBAT_Type *base)

Unlock switch control.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT_CheckSwitchControlLocked(VBAT_Type *base)

Check if switch control is locked.

Parameters

- base – VBAT peripheral base address.

Return values

- false – switch control is not locked.
- true – switch control is locked, any writes to related registers are blocked.

status_t VBAT_InitClockMonitor(VBAT_Type *base, const *vbat_clock_monitor_config_t* *config)

Initialize the VBAT clock monitor, enable clock monitor and set the clock monitor configuration.

Note: Both FRO16K and OSC32K should be enabled and stable before invoking this function.

Parameters

- base – VBAT peripheral base address.
- config – Pointer to vbat_clock_monitor_config_t structure.

Return values

- kStatus_Success – Clock monitor is initialized successfully.
- kStatus_VBAT_Fro16kNotEnabled – FRO16K is not enabled.
- kStatus_VBAT_Osc32kNotReady – OSC32K is not ready.
- kStatus_VBAT_ClockMonitorLocked – Clock monitor is locked.

status_t VBAT_DeinitMonitor(VBAT_Type *base)

Deinitialize the VBAT clock monitor.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatus_Success – Clock monitor is de-initialized successfully.
- kStatus_VBAT_ClockMonitorLocked – Control of Clock monitor is locked.

static inline void VBAT_EnableClockMonitor(VBAT_Type *base, bool enable)

Enable/disable clock monitor.

- false: disable clock monitor.

Parameters

- base – VBAT peripheral base address.
- enable – Switcher to enable/disable clock monitor:
  - true: enable clock monitor;

static inline void VBAT_SetClockMonitorDivideTrim(VBAT_Type *base, *vbat_clock_monitor_divide_trim_t* divideTrim)

Set clock monitor's divide trim, avaiable value is kVBAT_ClockMonitorOperateAt1kHz and kVBAT_ClockMonitorOperateAt64Hz.

Parameters

- base – VBAT peripheral base address.
- divideTrim – Specify divide trim value, please refer to vbat_clock_monitor_divide_trim_t.

static inline void VBAT_SetClockMonitorFrequencyTrim(VBAT_Type *base, *vbat_clock_monitor_freq_trim_t* freqTrim)

Set clock monitor's frequency trim, avaiable value is kVBAT_ClockMonitorAssert2Cycle, kVBAT_ClockMonitorAssert4Cycle, kVBAT_ClockMonitorAssert6Cycle and kVBAT_ClockMonitorAssert8Cycle.

Parameters

- base – VBAT peripheral base address.
- freqTrim – Specify frequency trim value, please refer to vbat_clock_monitor_freq_trim_t.

static inline void VBAT_LockClockMonitorControl(VBAT_Type *base)

Lock clock monitor enable/disable control.

Note: If locked, it is not allowed to change clock monitor enable/disable control.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_UnlockClockMonitorControl(VBAT_Type *base)

Unlock clock monitor enable/disable control.

Parameters

- base – VBTA peripheral base address.

static inline bool VBAT_CheckClockMonitorControlLocked(VBAT_Type *base)

Check if clock monitor enable/disable control is locked.

Note: If locked, it is not allowed to change clock monitor enable/disable control.

Parameters

- base – VBAT peripheral base address.

Return values

- false – clock monitor enable/disable control is not locked.
- true – clock monitor enable/disable control is locked, any writes to related registers are blocked.

status_t VBAT_InitTamper(VBAT_Type *base, const *vbat_tamper_config_t* *config)

Initialize tamper control.

Note: Both FRO16K and bandgap should be enabled before calling this function.

Parameters

- base – VBAT peripheral base address.
- config – Pointer to vbat_tamper_config_t structure.

Return values

- kStatus_Success – Tamper is initialized successfully.
- kStatus_VBAT_TamperLocked – Tamper control is locked.
- kStatus_VBAT_BandgapNotEnabled – Bandgap is not enabled.
- kStatus_VBAT_Fro16kNotEnabled – FRO 16K is not enabled.

status_t VBAT_DeinitTamper(VBAT_Type *base)

De-initialize tamper control.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatus_Success – Tamper is de-initialized successfully.
- kStatus_VBAT_TamperLocked – Tamper control is locked.

static inline void VBAT_EnableTamper(VBAT_Type *base, uint32_t tamperEnableMask)

Enable tampers for VBAT.

Parameters

- base – VBAT peripheral base address.
- tamperEnableMask – Mask of tamper to be enabled, should be the OR'ed value of _vbat_tamper_enable.

static inline void VBAT_DisableTamper(VBAT_Type *base, uint32_t tamperEnableMask)

Disable tampers for VBAT.

Parameters

- base – VBAT peripheral base address.

- tamperEnableMask – Mask of tamper to be disabled, should be the OR'ed value of _vbat_tamper_enable.

static inline uint32_t VBAT_GetTamperEnableInfo(VBAT_Type *base)

Get tamper enable information.

Parameters

- base – VBAT peripheral base address.

Returns

Mask of tamper enable information, should be the OR'ed value of _vbat_tamper_enable.

static inline void VBAT_LockTamperControl(VBAT_Type *base)

Lock tamper control, if locked, it is not allowed to change tamper control.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_UnlockTamperControl(VBAT_Type *base)

Unlock tamper control.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT_CheckTamperControlLocked(VBAT_Type *base)

Check if tamper control is locked.

Parameters

- base – VBAT peripheral base address.

Return values

- false – Tamper control is not locked.
- true – Tamper control is locked, any writes to related registers are blocked.

static inline uint32_t VBAT_GetStatusFlags(VBAT_Type *base)

Get VBAT status flags.

Parameters

- base – VBAT peripheral base address.

Returns

The asserted status flags, should be the OR'ed value of vbat_status_flag_t.

static inline void VBAT_ClearStatusFlags(VBAT_Type *base, uint32_t mask)

Clear VBAT status flags.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of status flags to be cleared, should be the OR'ed value of vbat_status_flag_t except kVBAT_StatusFlagLdoReady, kVBAT_StatusFlagOsc32kReady, kVBAT_StatusFlagInterrupt0Detect, kVBAT_StatusFlagInterrupt1Detect, kVBAT_StatusFlagInterrupt2Detect, kVBAT_StatusFlagInterrupt3Detect.

static inline void VBAT_EnableInterrupts(VBAT_Type *base, uint32_t mask)

Enable interrupts for the VBAT module, such as POR detect interrupt, Wakeup Pin interrupt and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be enabled, should be the OR'ed value of vbat_interrupt_enable_t.

static inline void VBAT_DisableInterrupts(VBAT_Type *base, uint32_t mask)

Disable interrupts for the VBAT module, such as POR detect interrupt, wakeup pin interrupt and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be disabled, should be the OR'ed value of vbat_interrupt_enable_t.

static inline void VBAT_EnableWakeup(VBAT_Type *base, uint32_t mask)

Enable wakeup for the VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of enumerators in vbat_wakeup_enable_t.

static inline void VBAT_DisableWakeup(VBAT_Type *base, uint32_t mask)

Disable wakeup for VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of enumerators in vbat_wakeup_enable_t.

static inline void VBAT_LockInterruptWakeupSettings(VBAT_Type *base)

Lock VBAT interrupt and wakeup settings, please note that if locked the interrupt and wakeup settings can not be updated until the next POR.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_SetWakeupPinDefaultState(VBAT_Type *base, bool assert)

Set the default state of the WAKEUP_b pin output when no enabled wakeup source is asserted.

Parameters

- base – VBAT peripheral base address.
- assert – Used to set default state of the WAKEUP_b pin output:
  - **true** WAKEUP_b output state is logic one;
  - **false** WAKEUP_b output state is logic zero.

struct __vbat_fro16k_config

#include <fsl_vbat.h> The structure of internal 16kHz free running oscillator attributes.

Public Members

bool enableFRO16k

Enable/disable internal 16kHz free running oscillator.

uint8_t enabledConnectionsMask

The mask of connected modules to enable FRO16k clock output.

struct _vbat_clock_monitor_config

#include <fsl_vbat.h> The structure of internal clock monitor, including divide trim and frequency trim.

Public Members

vbat_clock_monitor_freq_trim_t freqTrim

Frequency trim value used to adjust the clock monitor assert, please refer to vbat_clock_monitor_freq_trim_t.

bool lock

Lock the clock monitor control after enabled.

struct _vbat_tamper_config

#include <fsl_vbat.h> The structure of Tamper configuration.

Public Members

bool enableVoltageDetect

Enable/disable voltage detection.

bool enableTemperatureDetect

Enable/disable temperature detection.

bool lock

Lock the tamper control after enabled.

2.76 Memory_interface

Memory group definition.

Values:

enumerator kMemoryGroup_Internal

Memory belongs internal 4G memory region.

enumerator kMemoryGroup_External

Memory belongs external memory region.

Memory device ID definition.

Values:

enumerator kMemoryInternal

Internal memory (include all on chip memory)

enumerator kMemoryQuadSpi0

Qsuad SPI memory 0

enumerator kMemoryIFR0

Nonvolatile information register 0. Only used by SB loader.

enumerator kMemoryFFR
LPCc040hd flash FFR region.

enumerator kMemorySemcNor
SEMC Nor memory

enumerator kMemoryFlexSpiNor
Flex SPI Nor memory

enumerator kMemorySpifiNor
SPIFI Nor memory

enumerator kMemoryFlashExecuteOnly
Execute-only region on internal Flash

enumerator kMemorySemcNand
SEMC NAND memory

enumerator kMemorySpiNand
SPI NAND memory

enumerator kMemorySpiNorEeprom
SPI NOR/EEPROM memory

enumerator kMemoryI2cNorEeprom
I2C NOR/EEPROM memory

enumerator kMemorySDCard
eSD, SD, SDHC, SDXC memory Card

enumerator kMemoryMMCCard
MMC, eMMC memory Card

Bootloader status group numbers.

Values:

enumerator kStatusGroup_Bootloader
Bootloader status group number (100).

enumerator kStatusGroup_MemoryInterface
Memory interface status group number (102).

Memory interface status codes.

Values:

enumerator kStatusMemoryRangeInvalid

enumerator kStatusMemoryReadFailed

enumerator kStatusMemoryWriteFailed

enumerator kStatusMemoryCumulativeWrite

enumerator kStatusMemoryAppOverlapWithExecuteOnlyRegion

enumerator kStatusMemoryNotConfigured

enumerator kStatusMemoryAlignmentError

enumerator kStatusMemoryVerifyFailed

enumerator kStatusMemoryWriteProtected

enumerator kStatusMemoryAddressError

enumerator kStatusMemoryBlankCheckFailed

enumerator kStatusMemoryBlankPageReadDisallowed

enumerator kStatusMemoryProtectedPageReadDisallowed

enumerator kStatusMemoryFfrSpecRegionWriteBroken

enumerator kStatusMemoryUnsupportedCommand

Bootloader status codes.

Values:

enumerator kStatus_UnknownCommand

enumerator kStatus_SecurityViolation

enumerator kStatus_AbortDataPhase

enumerator kStatus_Ping

enumerator kStatus_NoResponse

enumerator kStatus_NoResponseExpected

enumerator kStatus_CommandUnsupported

typedef union *StandardVersion* standard_version_t

Structure of version property.

typedef struct *kb_api_parameter_struct* kp_api_init_param_t

API initialization data structure.

standard_version_t API_Version(void)

status_t API_Init(*api_core_context_t* *coreCtx, const *kp_api_init_param_t* *param)

Initialize the IAP API runtime environment.

status_t API_Deinit(*api_core_context_t* *coreCtx)

Deinitialize the IAP API runtime environment.

status_t MEM_Init(*api_core_context_t* *coreCtx)

Initialize memory interface.

Return values

- kStatus_Fail –
- kStatus_Success –

status_t MEM_Config(*api_core_context_t* *coreCtx, uint32_t *config, uint32_t memoryId)

Configure memory interface.

Parameters

- config – A pointer to the storage for the driver runtime state.
- memoryId – Indicates the index of the memory type. Please refer to "Memory group definition"

Return values

- kStatus_Success –
- kStatus_CommandUnsupported –
- kStatus_InvalidArgument –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatusMemoryRangeInvalid –
- kStatus_Fail –
- kStatus_OutOfRange –
- #kStatus_SPI_BaudrateNotSupport –

status_t MEM_Write(*api_core_context_t* *coreCtx, uint32_t start, uint32_t lengthInBytes, const uint8_t *buf, uint32_t memoryId)

Write memory.

Parameters

- address – The start address of the desired flash memory to be programmed. For internal flash the address need to be 512bytes-aligned.
- length – Number of bytes to be programmed.
- buffer – A pointer to the source buffer of data that is to be programmed into the flash.
- memoryId – Indicates the index of the memory type. Please refer to "Memory group definition"

Return values

- kStatus_Success –
- kStatus_Fail –
- kStatusMemoryRangeInvalid –
- kStatus_CommandUnsupported –
- kStatus_FLASH_AlignmentError –
- kStatusMemoryCumulativeWrite –
- kStatus_FLASH_InvalidArgument –
- kStatus_FLASH_AddressError –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatus_FLASH_CommandFailure –
- kStatus_FLASH_CommandNotSupported –
- kStatus_FLASH_EccError –
- kStatus_FLASH_RegulationLoss –
- kStatus_FLASH_Success –
- kStatus_FLASH_ReadHidingAreaDisallowed –
- kStatus_FLASH_CompareError –
- kStatusMemoryNotConfigured –
- kStatusMemoryVerifyFailed –

status_t MEM_Fill(*api_core_context_t* *coreCtx, uint32_t start, uint32_t lengthInBytes, uint32_t pattern, uint32_t memoryId)

Fill memory with a word pattern.

Parameters

- address – The start address of the desired flash memory to be programmed. For internal flash the address need to be 512bytes-aligned.
- length – Number of bytes to be programmed.
- pattern – The data to be written into the specified memory area.

Return values

- kStatus_CommandUnsupported –
- kStatus_Success –
- kStatus_FLASH_AlignmentError –
- kStatusMemoryCumulativeWrite –
- kStatus_Fail –
- kStatus_FLASH_InvalidArgument –
- kStatus_FLASH_AddressError –
- kStatus_FLASH_Success –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatus_FLASH_CommandFailure –
- kStatus_FLASH_CommandNotSupported –
- kStatus_FLASH_EccError –
- kStatus_FLASH_RegulationLoss –
- kStatus_FLASH_ReadHidingAreaDisallowed –

status_t MEM_Flush(*api_core_context_t* *coreCtx)

Flush memory.

Return values

- kStatus_Success –
- kStatus_Fail –
- kStatusMemoryCumulativeWrite –
- kStatus_FLASH_InvalidArgument –
- kStatus_FLASH_AlignmentError –
- kStatus_FLASH_Success –
- kStatus_FLASH_AddressError –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatus_FLASH_CommandFailure –
- kStatus_FLASH_CommandNotSupported –
- kStatus_FLASH_EccError –
- kStatus_FLASH_RegulationLoss –
- kStatus_FLASH_ReadHidingAreaDisallowed –
- kStatusMemoryVerifyFailed –

status_t MEM_Erase(*api_core_context_t* *coreCtx, uint32_t start, uint32_t lengthInBytes, uint32_t memoryId)

Erase memory.

Parameters

- address – The start address of the desired flash memory to be erased.
- length – Number of bytes to be read.
- memoryId – Indicates the index of the memory type. Please refer to "Memory group definition"

Return values

- kStatus_Success –
- kStatusMemoryRangeInvalid –
- kStatusMemoryAddressError –
- kStatus_FLASH_InvalidArgument –
- kStatus_FLASH_AlignmentError –
- kStatus_FLASH_Success –
- kStatus_FLASH_AddressError –
- kStatus_FLASH_EraseKeyError –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatus_Fail –
- kStatus_FLASH_CommandFailure –
- kStatus_FLASH_CommandNotSupported –
- kStatus_FLASH_EccError –
- kStatus_FLASH_RegulationLoss –
- kStatusMemoryNotConfigured –
- kStatusMemoryVerifyFailed –

status_t MEM_EraseAll(*api_core_context_t* *coreCtx, uint32_t memoryId)

Erase entire memory based on memoryId.

Parameters

- memoryId – Indicates the index of the memory type. Please refer to "Memory group definition"

Return values

- kStatus_Success –
- kStatus_Fail –
- kStatus_CommandUnsupported –
- kStatus_FLASH_InvalidArgument –
- kStatus_FLASH_AlignmentError –
- kStatus_FLASH_Success –
- kStatus_FLASH_AddressError –
- kStatus_FLASH_EraseKeyError –
- kStatus_FLASH_CommandFailure –

- kStatus_FLASH_CommandNotSupported –
- kStatus_FLASH_EccError –
- kStatus_FLASH_RegulationLoss –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatusMemoryVerifyFailed –
- kStatusMemoryNotConfigured –
- kStatus_InvalidArgument –

FSL_ROMAPI_MEM_INTERFACE_DRIVER_VERSION
: ROMAPI_MEM_INTERFACE driver version 2.0.0.

DEVICE_ID_MASK
: Bit mask for device ID.

DEVICE_ID_SHIFT
: Bit position of device ID.

GROUP_ID_MASK
: Bit mask for group ID.

GROUP_ID_SHIFT
: Bit position of group ID.

MAKE_MEMORYID(group, device)
: Construct a memory ID from a given group ID and device ID.

GROUPID(memoryId)
: Get group ID from a given memory ID.

DEVICEID(memoryId)
: Get device ID from a given memory ID.

status_t (*init)(void)

status_t (*read)(uint32_t address, uint32_t length, uint8_t *buffer, uint32_t memoryId)

status_t (*write)(uint32_t address, uint32_t length, const uint8_t *buffer, uint32_t memoryId)

status_t (*fill)(uint32_t address, uint32_t length, uint32_t pattern)

status_t (*flush)(void)

status_t (*finalize)(void)

status_t (*erase)(uint32_t address, uint32_t length, uint32_t memoryId)

status_t (*init)(void)

status_t (*read)(uint32_t address, uint32_t length, uint8_t *buffer)

status_t (*write)(uint32_t address, uint32_t length, const uint8_t *buffer)

status_t (*fill)(uint32_t address, uint32_t length, uint32_t pattern)

status_t (*flush)(void)

status_t (*erase)(uint32_t address, uint32_t length)

status_t (*config)(uint32_t *buffer)

status_t (*erase_all)(void)

uint32_t startAddress

uint32_t endAddress

uint32_t memoryProperty

uint32_t memoryId

const *memory_region_interface_t* *memoryInterface

uint8_t bugfix
: bugfix version [7:0]

uint8_t minor
: minor version [15:8]

uint8_t major
: major version [23:16]

char name
: name [31:24]

struct StandardVersion

uint32_t version
: combined version numbers

uint32_t allocStart

uint32_t allocSize

struct memory_interface_t
: *#include <fsl_mem_interface.h>* Interface to memory operations.

  This is the main abstract interface to all memory operations.

struct memory_region_interface_t
: *#include <fsl_mem_interface.h>* Interface to memory operations for one region of memory.

struct memory_map_entry_t
: *#include <fsl_mem_interface.h>* Structure of a memory map entry.

union StandardVersion
: *#include <fsl_mem_interface.h>* Structure of version property.

struct kb_api_parameter_struct
: *#include <fsl_mem_interface.h>* API initialization data structure.

struct ___unnamed41___

Public Members

uint8_t bugfix
: bugfix version [7:0]

uint8_t minor
: minor version [15:8]

uint8_t major
: major version [23:16]

char name
: name [31:24]

2.77 MRT: Multi-Rate Timer

void MRT_Init(MRT_Type *base, const *mrt_config_t* *config)

Ungates the MRT clock and configures the peripheral for basic operation.

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

Parameters

- base – Multi-Rate timer peripheral base address
- config – Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MODCFG reigster, param config is useless.

void MRT_Deinit(MRT_Type *base)

Gate the MRT clock.

Parameters

- base – Multi-Rate timer peripheral base address

static inline void MRT_GetDefaultConfig(*mrt_config_t* *config)

Fill in the MRT config struct with the default settings.

The default values are:

```
config->enableMultiTask = false;
```

Parameters

- config – Pointer to user's MRT config structure.

static inline void MRT_SetupChannelMode(MRT_Type *base, *mrt_chnl_t* channel, const *mrt_timer_mode_t* mode)

Sets up an MRT channel mode.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Channel that is being configured.
- mode – Timer mode to use for the channel.

static inline void MRT_EnableInterrupts(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Enables the MRT interrupt.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration mrt_interrupt_enable_t

static inline void MRT_DisableInterrupts(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Disables the selected MRT interrupt.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

- mask – The interrupts to disable. This is a logical OR of members of the enumeration mrt_interrupt_enable_t

static inline uint32_t MRT_GetEnabledInterrupts(MRT_Type *base, *mrt_chnl_t* channel)

Gets the enabled MRT interrupts.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns

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

static inline uint32_t MRT_GetStatusFlags(MRT_Type *base, *mrt_chnl_t* channel)

Gets the MRT status flags.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns

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

static inline void MRT_ClearStatusFlags(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Clears the MRT status flags.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration mrt_status_flags_t

void MRT_UpdateTimerPeriod(MRT_Type *base, *mrt_chnl_t* channel, uint32_t count, bool immediateLoad)

Used to update the timer period in units of count.

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks

---

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- count – Timer period in units of ticks
- immediateLoad – true: Load the new value immediately into the TIMER register; false: Load the new value at the end of current timer interval

static inline uint32_t MRT_GetCurrentTimerCount(MRT_Type *base, *mrt_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in fsl_common.h to convert ticks to usec or msec

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns

Current timer counting value in ticks

static inline void MRT_StartTimer(MRT_Type *base, *mrt_chnl_t* channel, uint32_t count)

Starts the timer counting.

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.
- count – Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

static inline void MRT_StopTimer(MRT_Type *base, *mrt_chnl_t* channel)

Stops the timer counting.

This function stops the timer from counting.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

static inline uint32_t MRT_GetIdleChannel(MRT_Type *base)

Find the available channel.

This function returns the lowest available channel number.

Parameters

- base – Multi-Rate timer peripheral base address

static inline void MRT_ReleaseChannel(MRT_Type *base, *mrt_chnl_t* channel)

Release the channel when the timer is using the multi-task mode.

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling MRT_GetIdleChannel() for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

FSL_MRT_DRIVER_VERSION

enum _mrt_chnl

List of MRT channels.

Values:

enumerator kMRT_Channel_0

MRT channel number 0

enumerator kMRT_Channel_1

MRT channel number 1

enumerator kMRT_Channel_2

MRT channel number 2

enumerator kMRT_Channel_3

MRT channel number 3

enum _mrt_timer_mode

List of MRT timer modes.

Values:

enumerator kMRT_RepeatMode

Repeat Interrupt mode

enumerator kMRT_OneShotMode

One-shot Interrupt mode

enumerator kMRT_OneShotStallMode

One-shot stall mode

enum _mrt_interrupt_enable

List of MRT interrupts.

Values:

enumerator kMRT_TimerInterruptEnable

Timer interrupt enable

enum _mrt_status_flags

List of MRT status flags.

Values:

enumerator kMRT_TimerInterruptFlag

Timer interrupt flag

enumerator kMRT_TimerRunFlag

Indicates state of the timer

typedef enum *_mrt_chnl* mrt_chnl_t

List of MRT channels.

typedef enum *_mrt_timer_mode* mrt_timer_mode_t

List of MRT timer modes.

typedef enum *_mrt_interrupt_enable* mrt_interrupt_enable_t

List of MRT interrupts.

typedef enum *_mrt_status_flags* mrt_status_flags_t

List of MRT status flags.

typedef struct *_mrt_config* mrt_config_t

MRT configuration structure.

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

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

struct _mrt_config

#include <fsl_mrt.h> MRT configuration structure.

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

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

Public Members

bool enableMultiTask

true: Timers run in multi-task mode; false: Timers run in hardware status mode

2.78 Nboot

nboot status codes.

Values:

enumerator kStatus_NBOOT_Success

Operation completed successfully.

enumerator kStatus_NBOOT_Fail

Operation failed.

enumerator kStatus_NBOOT_InvalidArgument

Invalid argument passed to the function.

enumerator kStatus_NBOOT_RequestTimeout

Operation timed out.

enumerator kStatus_NBOOT_KeyNotLoaded

The requested key is not loaded.

enumerator kStatus_NBOOT_AuthFail

Authentication failed.

enumerator kStatus_NBOOT_OperationNotAvaialable

Operation not available on this HW.

enumerator kStatus_NBOOT_KeyNotAvailable

Key is not avaialble.

enumerator kStatus_NBOOT_IvCounterOverflow

Overflow of IV counter (PRINCE/IPED).

enumerator kStatus_NBOOT_SelftestFail

FIPS self-test failure.

enumerator kStatus_NBOOT_InvalidDataFormat
Invalid data format for example antipole

enumerator kStatus_NBOOT_IskCertUserDataTooBig
Size of User data in ISK certificate is greater than 96 bytes

enumerator kStatus_NBOOT_IskCertSignatureOffsetTooSmall
Signature offset in ISK certificate is smaller than expected

enumerator kStatus_NBOOT_MemcpyFail
Unexpected error detected during nboot_memcpy()

enum _nboot_bool
Boolean type for the NBOOT functions.

This type defines boolean values used by NBOOT functions that are not easily disturbed by Fault Attacks

Values:

enumerator kNBOOT_TRUE
Value for TRUE.

enumerator kNBOOT_TRUE256
Value for TRUE when P256 was used to sign the image.

enumerator kNBOOT_TRUE384
Value for TRUE when P384 was used to sign the image.

enumerator kNBOOT_FALSE
Value for FALSE.

enumerator kNBOOT_OperationAllowed

enumerator kNBOOT_OperationDisallowed

typedef int romapi_status_t

typedef uint32_t nboot_root_key_usage_t

typedef uint32_t nboot_root_key_revocation_t

typedef uint32_t nboot_root_key_type_and_length_t

typedef uint32_t nboot_soc_lifecycle_t

typedef uint32_t nboot_status_t
Type for nboot status codes.

typedef uint64_t nboot_status_protected_t
Type for nboot protected status codes.

typedef struct *_nboot_secure_counter* nboot_secure_counter_t
Data structure holding secure counter value used by nboot library.

typedef struct *_nboot_context* nboot_context_t
NBOOT context type.

This type defines the NBOOT context

typedef struct *_nboot_rot_auth_parms* nboot_rot_auth_parms_t
NBOOT type for the root of trust parameters.

This type defines the NBOOT root of trust parameters

typedef struct *_nboot_sb3_load_manifest_parms* nboot_sb3_load_manifest_parms_t

manifest loading parameters

This type defines the NBOOT SB3.1 manifest loading parameters

typedef struct *_nboot_img_auth_ecdsa_parms* nboot_img_auth_ecdsa_parms_t

Data structure holding input arguments to POR secure boot (authentication) algorithm. Shall be read from SoC trusted NVM or SoC fuses.

typedef struct *_nboot_cmac_authenticate_parms* nboot_img_authenticate_cmac_parms_t

Data structure holding input arguments for CMAC authentication.

typedef enum *_nboot_bool* nboot_bool_t

Boolean type for the NBOOT functions.

This type defines boolean values used by NBOOT functions that are not easily disturbed by Fault Attacks

status_t NBOOT_GenerateRandom(uint8_t *output, size_t outputByteLen)

This API function is used to generate random number with specified length.

Parameters

- output – Pointer to random number buffer
- outputByteLen – length of generated random number in bytes. Length has to be in range <1, 2^16>

Return values

- kStatus_NBOOT_InvalidArgument – Invalid input parameters (Input pointers points to NULL or length is invalid)
- kStatus_NBOOT_Success – Operation successfully finished
- kStatus_NBOOT_Fail – Error occured during operation

nboot_status_t NBOOT_ContextInit(*nboot_context_t* *context)

The function is used for initializing of the nboot context data structure. It should be called prior to any other calls of nboot API.

Parameters

- nbootCtx – Pointer to nboot_context_t structure.

Return values

- kStatus_NBOOT_Success – Operation successfully finished
- kStatus_NBOOT_Fail – Error occured during operation

nboot_status_t NBOOT_ContextDeinit(*nboot_context_t* *context)

The function is used to deinitialize nboot context data structure. Its contents are overwritten with random data so that any sensitive data does not remain in memory.

Parameters

- context – Pointer to nboot_context_t structure.

Return values

- kStatus_NBOOT_Success – Operation successfully finished
- kStatus_NBOOT_Fail – Error occured during operation

nboot_status_protected_t NBOOT_Sb3LoadManifest(*nboot_context_t* *context, uint32_t *manifest, *nboot_sb3_load_manifest_parms_t* *parms)

Verify NBOOT SB3.1 manifest (header message)

This function verifies the NBOOT SB3.1 manifest (header message), initializes the context and loads keys into the CSS key store so that they can be used by nboot_sb3_load_block function. The NBOOT context has to be initialized by the function nboot_context_init before calling this function. Please note that this API is intended to be used only by users who needs to split FW update process (loading of SB3.1 file) to partial steps to customize whole operation. For regular SB3.1 processing, please use API described in chapter ��SBloader APIs��.

Parameters

- nbootCtx – Pointer to nboot_context_t structure.
- manifest – Pointer to the input manifest buffer
- params – additional input parameters. Please refer to nboot_sb3_load_manifest_parms_t definition for details.

Return values

- kStatus_NBOOT_Success – Operation successfully finished
- kStatus_NBOOT_Fail – Error occured during operation

nboot_status_protected_t NBOOT_Sb3LoadBlock(*nboot_context_t* *context, uint32_t *block)

Verify NBOOT SB3.1 block.

This function verifies and decrypts an NBOOT SB3.1 block. Decryption is performed in-place. The NBOOT context has to be initialized by the function nboot_context_init before calling this function. Please note that this API is intended to be used only by users who needs to split FW update process (loading of SB3.1 file) to partial steps to customize whole operation. For regular SB3.1 processing, please use API described in chapter ��SBloader APIs��.

Parameters

- context – Pointer to nboot_context_t structure.
- block – Pointer to the input SB3.1 data block

Return values

- kStatus_NBOOT_Success – successfully finished
- kStatus_NBOOT_Fail – occured during operation

nboot_status_protected_t NBOOT_ImgAuthenticateEcdsa(*nboot_context_t* *context, uint8_t imageStartAddress[], *nboot_bool_t* *isSignatureVerified, *nboot_img_auth_ecdsa_parms_t* *parms)

This function authenticates image with asymmetric cryptography. The NBOOT context has to be initialized by the function nboot_context_init before calling this function.

Parameters

- context – Pointer to nboot_context_t structure.
- imageStartAddress – Pointer to start of the image in memory.
- isSignatureVerified – Pointer to memory holding function call result. After the function returns, the value will be set to kNBOOT_TRUE when the image is authentic. Any other value means the authentication does not pass.
- parms – Pointer to a data structure in trusted memory, holding input parameters for the algorithm. The data structure shall be correctly filled before the function call.

Return values

- kStatus_NBOOT_Success – Operation successfully finished
- kStatus_NBOOT_Fail – Returned in all other cases. Doesn't always mean invalid image, it could also mean transient error caused by short time environmental conditions.

nboot_status_protected_t NBOOT_ImgAuthenticateCmac(*nboot_context_t* *context, uint8_t imageStartAddress[], *nboot_bool_t* *isSignatureVerified, *nboot_img_authenticate_cmac_parms_t* *parms)

This function calculates the CMAC over the given image and compares it to the expected value. To be more resistant against SPA, it is recommended that imageStartAddress is word aligned. The NBOOT context has to be initialized by the nboot_context_init() before calling this function.

Parameters

- context – Pointer to nboot_context_t structure.
- imageStartAddress – Pointer to start of the image in memory.
- isSignatureVerified – Pointer to memory holding function call result. After the function returns, the value will be set to
- parms – Pointer to a data structure in trusted memory, holding the reference MAC. The data structure shall be correctly filled before the function call.

Return values

- kStatus_NBOOT_Success –
- kStatus_NBOOT_Fail –

FSL_ROMAPI_NBOOT_DRIVER_VERSION

ROMAPI_NBOOT driver version 2.0.0.

NXPCLHASH_WA_SIZE_MAX

Define the max workarea size required for this component.

NBOOT_ROOT_CERT_COUNT

NXPCLCSS_HASH_RTF_OUTPUT_SIZE_HAL

Size of RTF appendix to hash output buffer, in bytes.

NBOOT_KEYINFO_WORDLEN

NBOOT_CONTEXT_BYTELEN

NBOOT_CONTEXT_WORDLEN

kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey

NBOOT type for the root key usage.

This type defines the NBOOT root key usage; any other value means the root key is not valid (treat as if revoked).

kNBOOT_RootKeyUsage_DebugCA

kNBOOT_RootKeyUsage_ImageCA_FwCA

kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA

kNBOOT_RootKeyUsage_ImageKey_FwKey

kNBOOT_RootKeyUsage_ImageKey

kNBOOT_RootKeyUsage_FwKey

kNBOOT_RootKeyUsage_Unused

kNBOOT_RootKey_Enabled
: NBOOT type for the root key revocation.

  This type defines the NBOOT root key revocation; any other value means the root key is revoked.

kNBOOT_RootKey_Revoked

kNBOOT_RootKey_Ecdsa_P256
: NBOOT type specifying the elliptic curve to be used.

  This type defines the elliptic curve type and length

kNBOOT_RootKey_Ecdsa_P384

nboot_lc_nxpBlank
: Enumeration for SoC Lifecycle.

nboot_lc_nxpFab

nboot_lc_nxpDev

nboot_lc_nxpProvisioned

nboot_lc_oemOpen

nboot_lc_oemSecureWorld

nboot_lc_oemClosed

nboot_lc_oemLocked

nboot_lc_oemFieldReturn

nboot_lc_nxpFieldReturn

nboot_lc_shredded

uint32_t sc

uint32_t scAp

uint32_t totalBlocks
: holds number of SB3 blocks. Initialized by nboot_sb3_load_header().

uint32_t processData
: flag, initialized by nboot_sb3_load_header(). SB3 related flag set by NBOOT in case the nboot_sb3_load_block() provides plain data to output buffer (for processing by ROM SB3 loader

uint32_t timeout
: timeout value for css operation. In case it is 0, infinite wait is performed

uint32_t keyinfo[(23U)]
: data for NBOOT key management.

uint32_t context[((192U + (128U + 64U)) / sizeof(uint32_t))]
: work area for NBOOT lib.

uint32_t uuid[4]
: holds UUID value from NMPA

uint32_t prngReadyFlag
: flag, used by nboot_rng_generate_lq_random() to determine whether CSS is ready to generate rnd number

uint32_t multipartMacBuffer[1024 / sizeof(uint32_t)]

uint32_t oemShareValidFlag
: flag, used during TP to determine whether valid oemShare was set by nboot_tp_isp_gen_oem_master_share()

uint32_t oemShare[4]
: buffer to store OEM_SHARE computed by nxpCLTrustProv_nboot_isp_gen_oem_master_share()

nboot_secure_counter_t secureCounter
: Secure counter used by nboot

uint32_t rtf[((size_t)32U) / sizeof(uint32_t)]

uint32_t imageHash[48 / sizeof(uint32_t)]

uint32_t authStatus

nboot_root_key_revocation_t soc_rootKeyRevocation[(4U)]
: Provided by caller based on NVM information in CFPA: ROTKH_REVOKE

uint32_t soc_imageKeyRevocation
: Provided by caller based on NVM information in CFPA: IMAGE_KEY_REVOKE

uint32_t soc_rkh[12]
: Provided by caller based on NVM information in CMPA: ROTKH (hash of hashes) In case of kNBOOT_RootKey_Ecdsa_P384, sock_rkh[0..11] are used In case of kNBOOT_RootKey_Ecdsa_P256, sock_rkh[0..7] are used

uint32_t soc_numberOfRootKeys
: unsigned int, between minimum = 1 and maximum = 4;

nboot_root_key_usage_t soc_rootKeyUsage[(4U)]
: CMPA

nboot_root_key_type_and_length_t soc_rootKeyTypeAndLength
: static selection between ECDSA P-256 or ECDSA P-384 based root keys

nboot_soc_lifecycle_t soc_lifecycle

nboot_rot_auth_parms_t soc_RoTNVM
: trusted information originated from CFPA and NMPA

uint32_t soc_trustedFirmwareVersion
: Provided by caller based on NVM information in CFPA: Secure_FW_Version

uint8_t pckBlob[48]

nboot_rot_auth_parms_t soc_RoTNVM
: trusted information originated from CFPA and NMPA

uint32_t soc_trustedFirmwareVersion
: Provided by caller based on NVM information in CFPA: Secure_FW_Version

uint32_t expectedMAC[4]
expected MAC result

struct __nboot__secure__counter
#include <fsl_nboot.h> Data structure holding secure counter value used by nboot library.

struct __nboot__context
#include <fsl_nboot.h> NBOOT context type.

This type defines the NBOOT context

struct __nboot__rot__auth__parms
#include <fsl_nboot.h> NBOOT type for the root of trust parameters.

This type defines the NBOOT root of trust parameters

struct __nboot__sb3__load__manifest__parms
#include <fsl_nboot.h> manifest loading parameters

This type defines the NBOOT SB3.1 manifest loading parameters

struct __nboot__img__auth__ecdsa__parms
#include <fsl_nboot.h> Data structure holding input arguments to POR secure boot (authentication) algorithm. Shall be read from SoC trusted NVM or SoC fuses.

struct __nboot__cmac__authenticate__parms
#include <fsl_nboot.h> Data structure holding input arguments for CMAC authentication.

2.79 Nbot_hal

enum __nboot__hash__algo__t
Algorithm used for nboot HASH operation.

Values:

enumerator kHASH__Sha1
SHA_1

enumerator kHASH__Sha256
SHA_256

enumerator kHASH__Sha512
SHA_512

enumerator kHASH__Aes
AES

enumerator kHASH__AesIcb
AES_ICB

typedef uint32_t nboot__timestamp__t[2]
NBOOT type for a timestamp.

This type defines the NBOOT timestamp

typedef struct *_nboot_sb3_header* nboot__sb3__header__t
NBOOT SB3.1 header type.

This type defines the header used in the SB3.1 manifest

typedef struct *_nboot_certificate_header_block* nboot_certificate_header_block_t

NBOOT type for the header of the certificate block.

This type defines the NBOOT header of the certificate block, it is part of the nboot_certificate_block_t

typedef uint8_t nboot_ctrk_hash_t[(48u)]

typedef struct *_nboot_ctrk_hash_table* nboot_ctrk_hash_table_t

NBOOT type for the hash table.

This type defines the NBOOT hash table

typedef uint8_t nboot_ecc_coordinate_t[(48u)]

NBOOT type for an ECC coordinate.

This type defines the NBOOT ECC coordinate type ECC point coordinate, up to 384-bits. big endian.

typedef struct *_nboot_root_certificate_block* nboot_root_certificate_block_t

NBOOT type for the root certificate block.

This type defines the NBOOT root certificate block, it is part of the nboot_certificate_block_t

typedef struct *_nboot_certificate_block* nboot_certificate_block_t

NBOOT type for the certificate block.

This type defines the constant length part of an NBOOT certificate block

typedef enum *_nboot_hash_algo_t* nboot_hash_algo_t

Algorithm used for nboot HASH operation.

NBOOT_UUID_SIZE_IN_WORD

The size of the UUID.

NBOOT_UUID_SIZE_IN_BYTE

NBOOT_PUF_AC_SIZE_IN_BYTE

The size of the PUF activation code.

NBOOT_PUF_KC_SIZE_IN_BYTE

The size of the PUF key code.

NBOOT_KEY_STORE_SIZE_IN_BYTE

The size of the key store.

NBOOT_ROOT_ROTKH_SIZE_IN_WORD

The size of the root of trust key table hash.

NBOOT_ROOT_ROTKH_SIZE_IN_BYTE

NBOOT_KEY_BLOB_SIZE_IN_BYTE_256

The size of the blob with Key Blob.

NBOOT_KEY_BLOB_SIZE_IN_BYTE_384

NBOOT_KEY_BLOB_SIZE_IN_BYTE_MAX

NBOOT_DBG_AUTH_DBG_STATE_MASK

The mask of the value of the debug state .

NBOOT_DBG_AUTH_DBG_STATE_SHIFT

The shift inverted value of the debug state.

NBOOT_DBG_AUTH_DBG_STATE_ALL_DISABLED
The value with all debug feature disabled.

NBOOT_ROOT_OF_TRUST_HASH_SIZE_IN_BYTES

NBOOT_EC_COORDINATE_384_SIZE_IN_BYTES

NBOOT_EC_COORDINATE_MAX_SIZE

NBOOT_SB3_CHUNK_SIZE_IN_BYTES

NBOOT_SB3_BLOCK_HASH256_SIZE_IN_BYTES

NBOOT_SB3_BLOCK_HASH384_SIZE_IN_BYTES

NBOOT_SB3_MANIFEST_MAX_SIZE_IN_BYTES

NBOOT_SB3_BLOCK_MAX_SIZE_IN_BYTES

NBOOT_DICE_CSR_SIZE_IN_WORD
The size of the DICE certificate.

NBOOT_DICE_CSR_SIZE_IN_BYTES

NBOOT_DICE_CSR_ADDRESS
The physical address to put the DICE certificate.

NBOOT_IPED_IV_OFFSET
The offset for the PRCINE/IPED erase region return by nboot mem checker.

NBOOT_IMAGE_CMAC_UPDATE_NONE

NBOOT_IMAGE_CMAC_UPDATE_INDEX0

NBOOT_IMAGE_CMAC_UPDATE_INDEX1

NBOOT_IMAGE_CMAC_UPDATE_BOTH

NBOOT_IMAGE_CMAC_UPDATE_MASK

NBOOT_CMPA_CMAC_UPDATE_MASK

NBOOT_CMPA_CMAC_UPDATE_SHIFT

NBOOT_CMPA_UPDATE_CMAC_PFR

NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_SECURE

NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_CLOSE

NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_LOCKED

struct _nboot_sb3_header
#include <fsl_nboot_hal.h> NBOOT SB3.1 header type.

This type defines the header used in the SB3.1 manifest

Public Members

uint32_t magic
offset 0x00: Fixed 4-byte string of ‘sbv3’ without the trailing NULL

uint32_t formatVersion

offset 0x04: (major = 3, minor = 1); The format version determines the manifest (block0) size.

uint32_t flags

offset 0x08: not defined yet, keep zero for future compatibility

uint32_t blockCount

offset 0x0C: Number of blocks not including the manifest (block0).

uint32_t blockSize

offset 0x10: Size in bytes of data block (repeated blockCount times for SB3 data stream).

nboot_timestamp_t timeStamp

offset 0x14: 64-bit value used as key derivation data.

uint32_t firmwareVersion

offset 0x1c: Version number of the included firmware

uint32_t imageTotalLength

offset 0x20: Total manifest length in bytes, including signatures etc.

uint32_t imageType

offset 0x24: image type and flags

uint32_t certificateBlockOffset

offset 0x28: Offset from start of header block to the certificate block.

uint8_t description[16]

offset 0x32: This field provides description of the file. It is an arbitrary string injected by the signing tool, which helps to identify the file.

struct __nboot__certificate__header__block

#include <fsl_nboot_hal.h> NBOOT type for the header of the certificate block.

This type defines the NBOOT header of the certificate block, it is part of the nboot_certificate_block_t

Public Members

uint32_t magic

magic number.

uint32_t formatMajorMinorVersion

format major minor version

uint32_t certBlockSize

Size of the full certificate block

struct __nboot__ctrk__hash__table

#include <fsl_nboot_hal.h> NBOOT type for the hash table.

This type defines the NBOOT hash table

struct nboot__ecdsa__public__key__t

#include <fsl_nboot_hal.h> NBOOT type for an ECC point.

This type defines the NBOOT ECC point type

Public Members

nboot_ecc_coordinate_t x
: x portion of the ECDSA public key, up to 384-bits. big endian.

nboot_ecc_coordinate_t y
: y portion of the ECDSA public key, up to 384-bits. big endian.

struct __nboot__root__certificate__block
: *#include <fsl_nboot_hal.h>* NBOOT type for the root certificate block.

This type defines the NBOOT root certificate block, it is part of the nboot_certificate_block_t

Public Members

uint32_t flags
: root certificate flags

nboot_ctrk_hash_table_t ctrkHashTable
: hash table

nboot_ecdsa_public_key_t rootPublicKey
: root public key

struct nboot__ecdsa__signature__t
: *#include <fsl_nboot_hal.h>* NBOOT type for an ECC signature.

This type defines the NBOOT ECC signature type

Public Members

nboot_ecc_coordinate_t r
: r portion of the ECDSA signature, up to 384-bits. big endian.

nboot_ecc_coordinate_t s
: s portion of the ECDSA signature, up to 384-bits. big endian.

struct nboot__isk__block__t
: *#include <fsl_nboot_hal.h>* NBOOT type for the isk block.

This type defines the constant length part of an NBOOT isk block

Public Members

uint32_t signatureOffset
: Offset of signature in ISK block.

uint32_t constraints
: Version number of signing certificate.

uint32_t iskFlags
: Reserved for definiton of ISK certificate flags.

nboot_ecdsa_public_key_t iskPubKey
: Public key of signing certificate. Variable length; only used to determine start address

nboot_ecdsa_public_key_t userData
: Space for at lest one addition public key

nboot_ecdsa_signature_t iskSign
ISK signature

struct _nboot_certificate_block
#include <fsl_nboot_hal.h> NBOOT type for the certificate block.

This type defines the constant length part of an NBOOT certificate block

Public Members

nboot_isk_block_t iskBlock
Details of selected root certificate (root certificate which will be used for ISK signing/SB3 header signing)

2.80 OPAMP: Operational Amplifier

void OPAMP_Init(OPAMP_Type *base, const *opamp_config_t* *config)
Initialize OPAMP instance.

Parameters

- base – OPAMP peripheral base address.
- config – The pointer to opamp_config_t.

void OPAMP_Deinit(OPAMP_Type *base)
De-initialize OPAMP instance.

Parameters

- base – OPAMP peripheral base address.

void OPAMP_GetDefaultConfig(*opamp_config_t* *config)
Get default configuration of OPAMP.

```
config->enable        = false;
config->mode          = kOPAMP_LowNoiseMode;
config->trimOption    = kOPAMP_TrimOptionDefault;
config->intRefVoltage = kOPAMP_IntRefVoltVddaDiv2;
config->enablePosADCSw = false;
config->posRefVoltage = kOPAMP_PosRefVoltVrefh3;
config->posGain       = kOPAMP_PosGainReserved;
config->negGain       = kOPAMP_NegGainBufferMode;
```

Parameters

- config – The pointer to opamp_config_t.

static inline void OPAMP_DoPosGainConfig(OPAMP_Type *base, *opamp_positive_gain_t* option)
Configure OPAMP positive port gain.

Parameters

- base – OPAMP peripheral base address.
- option – OPAMP positive port gain.

static inline void OPAMP_DoNegGainConfig(OPAMP_Type *base, *opamp_negative_gain_t* option)
Configure OPAMP negative port gain.

Parameters

- base – OPAMP peripheral base address.
- option – OPAMP negative port gain.

static inline void OPAMP_EnableRefBuffer(OPAMP_Type *base, bool enable)

Enable reference buffer.

Parameters

- base – OPAMP peripheral base address.
- enable – true to enable and false to disable.

static inline void OPAMP_EnableTriggerMode(OPAMP_Type *base, bool enable)

Enable OPAMP trigger mode.

Parameters

- base – OPAMP peripheral base address.
- enable – true to enable and false to disable.

FSL_OPAMP_DRIVER_VERSION

OPAMP driver version.

enum _opamp_mode

The enumeration of OPAMP mode, including low noise mode and high speed mode.

Values:

enumerator kOPAMP_LowNoiseMode

Set opamp mode as low noise mode.

enumerator kOPAMP_HighSpeedMode

Set opamp mode as high speed mode.

enum _opamp_bias_current_trim_option

The enumeration of bias current trim option.

Values:

enumerator kOPAMP_TrimOptionDefault

Default Bias current trim option.

enumerator kOPAMP_TrimOptionIncreaseCurrent

Trim option selected as increase current.

enumerator kOPAMP_TrimOptionDecreaseCurrent

Trim option selected as decrease current.

enumerator kOPAMP_TrimOptionFurtherDecreaseCurrent

Trim option selected as further decrease current.

enum _opamp_internal_ref_voltage

The enumeration of internal reference voltage.

Values:

enumerator kOPAMP_IntRefVoltVddaDiv2

Internal reference voltage selected as Vdda/2.

enumerator kOPAMP_IntRefVoltVdda3V

Internal reference voltage selected as Vdda_3V.

enumerator kOPAMP_IntRefVoltVssa3V

Internal reference voltage selected as Vssa_3V.

enumerator kOPAMP_IntRefVoltNotAllowed
Internal reference voltage not allowed.

enum _opamp_positive_ref_voltage
The enumeration of positive reference voltage(please refer to manual use).

Values:

enumerator kOPAMP_PosRefVoltVrefh3
Positive part reference voltage select Vrefh3, connected from DAC output.

enumerator kOPAMP_PosRefVoltVrefh0
Positive part reference voltage select Vrefh0, connected from VDDA supply.

enumerator kOPAMP_PosRefVoltVrefh1
Positive part reference voltage select Vrefh1, connected from Voltage reference output.

enumerator kOPAMP_PosRefVoltVrefh4
Positive part reference voltage select 520mv or reserved.

enum _opamp_positive_gain
The enumeration of positive programmable gain (please refer to manual use).

Values:

enumerator kOPAMP_PosGainReserved
Positive Gain reserved.

enumerator kOPAMP_PosGainNonInvert1X
Positive non-inverting gain application 1X.

enumerator kOPAMP_PosGainNonInvert2X
Positive non-inverting gain application 2X.

enumerator kOPAMP_PosGainNonInvert4X
Positive non-inverting gain application 4X.

enumerator kOPAMP_PosGainNonInvert8X
Positive non-inverting gain application 8X.

enumerator kOPAMP_PosGainNonInvert16X
Positive non-inverting gain application 16X.

enumerator kOPAMP_PosGainNonInvert33X
Positive non-inverting gain application 33X.

enumerator kOPAMP_PosGainNonInvert64X
Positive non-inverting gain application 64X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer2X
Positive non-inverting gain application 2X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer3X
Positive non-inverting gain application 3X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer5X
Positive non-inverting gain application 5X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer9X
Positive non-inverting gain application 9X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer17X
Positive non-inverting gain application 17X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer34X
Positive non-inverting gain application 34X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer65X
Positive non-inverting gain application 65X.

enum _opamp_negative_gain
The enumeration of negative programmable gain.

Values:

enumerator kOPAMP_NegGainBufferMode
Negative Buffer Mode.

enumerator kOPAMP_NegGainInvert1X
Negative inverting gain application -1X.

enumerator kOPAMP_NegGainInvert2X
Negative inverting gain application -2X.

enumerator kOPAMP_NegGainInvert4X
Negative inverting gain application -4X.

enumerator kOPAMP_NegGainInvert8X
Negative inverting gain application -8X.

enumerator kOPAMP_NegGainInvert16X
Negative inverting gain application -16X.

enumerator kOPAMP_NegGainInvert33X
Negative inverting gain application -33X.

enumerator kOPAMP_NegGainInvert64X
Negative inverting gain application -64X.

enum _opamp_positive_input_channel_selection
The enumeration of positive input channel selection.

Values:

enumerator kOPAMP_PosInputChannel0
When OPAMP not in trigger mode, select positive input 0 (INP0).

enumerator kOPAMP_PosInputChannel1
When OPAMP not in trigger mode, select positive input 1 (INP1).

typedef enum *_opamp_mode* opamp_mode_t
The enumeration of OPAMP mode, including low noise mode and high speed mode.

typedef enum *_opamp_bias_current_trim_option* opamp_bias_current_trim_option_t
The enumeration of bias current trim option.

typedef enum *_opamp_internal_ref_voltage* opamp_internal_ref_voltage_t
The enumeration of internal reference voltage.

typedef enum *_opamp_positive_ref_voltage* opamp_positive_ref_voltage_t
The enumeration of positive reference voltage(please refer to manual use).

typedef enum *_opamp_positive_gain* opamp_positive_gain_t
The enumeration of positive programmable gain (please refer to manual use).

typedef enum *_opamp_negative_gain* opamp_negative_gain_t
The enumeration of negative programmable gain.

typedef enum *_opamp_positive_input_channel_selection* opamp_positive_input_channel_selection_t

The enumeration of positive input channel selection.

typedef struct *_opamp_config* opamp_config_t

OPAMP configuraion, including mode, internal reference voltage, positive gain, negative gain and so on.

struct _opamp_config

#include <fsl_opamp.h> OPAMP configuraion, including mode, internal reference voltage, positive gain, negative gain and so on.

Public Members

bool enable

Enable/disable OPAMP.

opamp_mode_t mode

Opamp mode, available values are kOPAMP_LowNoiseMode and kOPAMP_HighSpeedMode.

opamp_bias_current_trim_option_t trimOption

Bias current trim option, please refer to opamp_bias_current_trim_option_t.

opamp_internal_ref_voltage_t intRefVoltage

Internal reference voltage, please refer to opamp_internal_ref_voltage_t.

opamp_positive_ref_voltage_t posRefVoltage

Positive part reference voltage, please refer to opamp_positive_ref_voltage_t.

opamp_positive_gain_t posGain

Positive part programmable gain, please refer to opamp_positive_gain_t.

opamp_negative_gain_t negGain

Negative part programmable gain, please refer to opamp_negative_gain_t.

bool enableOutputSwitch

OPAMP out to negative gain resistor ladder switch.

bool enablePosADCSw1

Positive part reference voltage switch to ADC channel or not.

- **true** Positive part reference voltage switch to ADC channel.
- **false** Positive part reference voltage do not switch to ADC channel.

bool enablePosADCSw2

Positive part reference voltage switch to ADC channel or not.

- **true** Positive part reference voltage switch to ADC channel.
- **false** Positive part reference voltage do not switch to ADC channel.

bool enableRefBuffer

Reference buffer enable.

opamp_positive_input_channel_selection_t PosInputChannelSelection

Positive Input Channel Selection

bool enableTriggerMode

Trigger Mode Enable.

2.81 OSTIMER: OS Event Timer Driver

void OSTIMER_Init(OSTIMER_Type *base)

Initializes an OSTIMER by turning its bus clock on.

void OSTIMER_Deinit(OSTIMER_Type *base)

Deinitializes a OSTIMER instance.

This function shuts down OSTIMER bus clock

Parameters

- base – OSTIMER peripheral base address.

uint64_t OSTIMER_GrayToDecimal(uint64_t gray)

Translate the value from gray-code to decimal.

Parameters

- gray – The gray value input.

Returns

The decimal value.

static inline uint64_t OSTIMER_DecimalToGray(uint64_t dec)

Translate the value from decimal to gray-code.

Parameters

- dec – The decimal value.

Returns

The gray code of the input value.

uint32_t OSTIMER_GetStatusFlags(OSTIMER_Type *base)

Get OSTIMER status Flags.

This returns the status flag. Currently, only match interrupt flag can be got.

Parameters

- base – OSTIMER peripheral base address.

Returns

status register value

void OSTIMER_ClearStatusFlags(OSTIMER_Type *base, uint32_t mask)

Clear Status Interrupt Flags.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

- base – OSTIMER peripheral base address.
- mask – Clear bit mask.

Returns

none

status_t OSTIMER_SetMatchRawValue(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match raw value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format may be gray-code, if so, please using OSTIMER_SetMatchValue().

Parameters

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value may be gray-code format)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success – - Set match raw value and enable interrupt Successfully.

status_t OSTIMER_SetMatchValue(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

Parameters

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code in API if the IP counter is gray encoded.)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success – - Set match raw value and enable interrupt Successfully.

static inline void OSTIMER_SetMatchRegister(OSTIMER_Type *base, uint64_t value)

Set value to OSTIMER MATCH register directly.

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER_DecimalToGray could convert decimal value to gray code.

Parameters

- base – OSTIMER peripheral base address.
- value – OSTIMER timer match value (Value is gray-code format).

static inline uint64_t OSTIMER_GetMatchRegister(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get the match value from OSTIMER. The value of timer match is gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register, data format is gray code.

static inline uint64_t OSTIMER_GetMatchValue(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get a match value from OSTIMER.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register.

static inline void OSTIMER_EnableMatchInterrupt(OSTIMER_Type *base)

Enable the OSTIMER counter match interrupt.

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline void OSTIMER_DisableMatchInterrupt(OSTIMER_Type *base)

Disable the OSTIMER counter match interrupt.

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline uint64_t OSTIMER_GetCurrentTimerRawValue(OSTIMER_Type *base)

Get current timer raw count value from OSTIMER.

This function will get the timer count value from OS timer register. The raw value of timer count may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, may be gray code format.

uint64_t OSTIMER_GetCurrentTimerValue(OSTIMER_Type *base)

Get current timer count value from OSTIMER.

This function will get a decimal timer count value. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formated to decimal value.

static inline uint64_t OSTIMER_GetCaptureRawValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a captured value from OSTIMER. The Raw value of timer capture may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of capture register, data format may be gray code.

uint64_t OSTIMER_GetCaptureValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a capture decimal-value from OSTIMER. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns
Value of capture register, data format is decimal.

void OSTIMER_HandleIRQ(OSTIMER_Type *base, *ostimer_callback_t* cb)

OS timer interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – OS timer peripheral base address.
- cb – callback scheduled for this instance of OS timer

Returns
none

FSL_OSTIMER_DRIVER_VERSION

OSTIMER driver version.

enum _ostimer_flags

OSTIMER status flags.

Values:

enumerator kOSTIMER_MatchInterruptFlag
Match interrupt flag bit, sets if the match value was reached.

typedef void (*ostimer_callback_t)(void)

ostimer callback function.

2.82 OTP: One-Time Programmable memory and API

FSL_OTP_DRIVER_VERSION

OTP driver version 2.0.1.

Current version: 2.0.1

Change log:

- Version 2.0.1
  - Fixed MISRA-C 2012 violations.
- Version 2.0.0
  - Initial version.

enum _otp_bank

Bank bit flags.

Values:

enumerator kOTP_Bank0
Bank 0.

enumerator kOTP_Bank1
Bank 1.

enumerator kOTP_Bank2
Bank 2.

enumerator kOTP_Bank3
Bank 3.

enum _otp_word
Bank word bit flags.

Values:

enumerator kOTP_Word0
Word 0.

enumerator kOTP_Word1
Word 1.

enumerator kOTP_Word2
Word 2.

enumerator kOTP_Word3
Word 3.

enum _otp_lock
Lock modifications of a read or write access to a bank register.

Values:

enumerator kOTP_LockDontLock
Do not lock.

enumerator kOTP_LockLock
Lock till reset.

enum _otp_status
OTP error codes.

Values:

enumerator kStatus_OTP_WrEnableInvalid
Write enable invalid.

enumerator kStatus_OTP_SomeBitsAlreadyProgrammed
Some bits already programmed.

enumerator kStatus_OTP_AllDataOrMaskZero
All data or mask zero.

enumerator kStatus_OTP_WriteAccessLocked
Write access locked.

enumerator kStatus_OTP_ReadDataMismatch
Read data mismatch.

enumerator kStatus_OTP_UsbIdEnabled
USB ID enabled.

enumerator kStatus_OTP_EthMacEnabled
Ethernet MAC enabled.

enumerator kStatus_OTP_AesKeysEnabled
AES keys enabled.

enumerator kStatus_OTP_IllegalBank
Illegal bank.

enumerator kStatus_OTP_ShufflerConfigNotValid
Shuffler config not valid.

enumerator kStatus_OTP_ShufflerNotEnabled
Shuffler not enabled.

enumerator kStatus_OTP_ShufflerCanOnlyProgSingleKey
Shuffler can only program single key.

enumerator kStatus_OTP_IllegalProgramData
Illegal program data.

enumerator kStatus_OTP_ReadAccessLocked
Read access locked.

typedef enum *_otp_bank* otp_bank_t
Bank bit flags.

typedef enum *_otp_word* otp_word_t
Bank word bit flags.

typedef enum *_otp_lock* otp_lock_t
Lock modifications of a read or write access to a bank register.

static inline *status_t* OTP_Init(void)
Initializes OTP controller.

Returns
kStatus_Success upon successful execution, error status otherwise.

static inline *status_t* OTP_EnableBankWriteMask(*otp_bank_t* bankMask)
Unlock one or more OTP banks for write access.

Parameters
- bankMask – bit flag that specifies which banks to unlock.

Returns
kStatus_Success upon successful execution, error status otherwise.

static inline *status_t* OTP_DisableBankWriteMask(*otp_bank_t* bankMask)
Lock one or more OTP banks for write access.

Parameters
- bankMask – bit flag that specifies which banks to lock.

Returns
kStatus_Success upon successful execution, error status otherwise.

static inline *status_t* OTP_EnableBankWriteLock(uint32_t bankIndex, *otp_word_t* regEnableMask, *otp_word_t* regDisableMask, *otp_lock_t* lockWrite)
Locks or unlocks write access to a register of an OTP bank and possibly lock un/locking of it.

Parameters
- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regEnableMask – bit flag that specifies for which words to enable writing.
- regDisableMask – bit flag that specifies for which words to disable writing.
- lockWrite – specifies if access set can be modified or is locked till reset.

Returns
kStatus_Success upon successful execution, error status otherwise.

static inline *status_t* OTP_EnableBankReadLock(uint32_t bankIndex, *otp_word_t* regEnableMask, *otp_word_t* regDisableMask, *otp_lock_t* lockWrite)

Locks or unlocks read access to a register of an OTP bank and possibly lock un/locking of it.

Parameters

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regEnableMask – bit flag that specifies for which words to enable reading.
- regDisableMask – bit flag that specifies for which words to disable reading.
- lockWrite – specifies if access set can be modified or is locked till reset.

Returns
kStatus_Success upon successful execution, error status otherwise.

static inline *status_t* OTP_ProgramRegister(uint32_t bankIndex, uint32_t regIndex, uint32_t value)

Program a single register in an OTP bank.

Parameters

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regIndex – OTP register index.
- value – value to write.

Returns
kStatus_Success upon successful execution, error status otherwise.

static inline uint32_t OTP_GetDriverVersion(void)

Returns the version of the OTP driver in ROM.

Returns
version.

FSL_COMPONENT_ID

_OTP_ERR_BASE

_OTP_MAKE_STATUS(errorCode)

2.83 PDM: Microphone Interface

2.84 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_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

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

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 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.

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_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_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

2.85 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

2.86 PINT: Pin Interrupt and Pattern Match Driver

FSL_PINT_DRIVER_VERSION

enum _pint_pin_enable
PINT Pin Interrupt enable type.

Values:

enumerator kPINT_PinIntEnableNone
Do not generate Pin Interrupt

enumerator kPINT_PinIntEnableRiseEdge
Generate Pin Interrupt on rising edge

enumerator kPINT_PinIntEnableFallEdge
Generate Pin Interrupt on falling edge

enumerator kPINT_PinIntEnableBothEdges
Generate Pin Interrupt on both edges

enumerator kPINT_PinIntEnableLowLevel
Generate Pin Interrupt on low level

enumerator kPINT_PinIntEnableHighLevel
Generate Pin Interrupt on high level

enum _pint_int
PINT Pin Interrupt type.

Values:

enumerator kPINT_PinInt0
Pin Interrupt 0

enum _pint_pmatch_input_src
PINT Pattern Match bit slice input source type.

Values:

enumerator kPINT_PatternMatchInp0Src
Input source 0

enumerator kPINT_PatternMatchInp1Src
Input source 1

enumerator kPINT_PatternMatchInp2Src
Input source 2

enumerator kPINT_PatternMatchInp3Src
Input source 3

enumerator kPINT_PatternMatchInp4Src
Input source 4

enumerator kPINT_PatternMatchInp5Src
Input source 5

enumerator kPINT_PatternMatchInp6Src
Input source 6

enumerator kPINT_PatternMatchInp7Src
Input source 7

enumerator kPINT_SecPatternMatchInp0Src
Input source 0

enumerator kPINT_SecPatternMatchInp1Src
Input source 1

enum _pint_pmatch_bslice
PINT Pattern Match bit slice type.

Values:

enumerator kPINT_PatternMatchBSlice0
Bit slice 0

enum _pint_pmatch_bslice_cfg
PINT Pattern Match configuration type.

Values:

enumerator kPINT_PatternMatchAlways
Always Contributes to product term match

enumerator kPINT_PatternMatchStickyRise
Sticky Rising edge

enumerator kPINT_PatternMatchStickyFall
Sticky Falling edge

enumerator kPINT_PatternMatchStickyBothEdges
Sticky Rising or Falling edge

enumerator kPINT_PatternMatchHigh
High level

enumerator kPINT_PatternMatchLow
Low level

enumerator kPINT_PatternMatchNever
Never contributes to product term match

enumerator kPINT_PatternMatchBothEdges
Either rising or falling edge

typedef enum *_pint_pin_enable* pint_pin_enable_t
PINT Pin Interrupt enable type.

typedef enum *_pint_int* pint_pin_int_t
PINT Pin Interrupt type.

typedef enum *_pint_pmatch_input_src* pint_pmatch_input_src_t
PINT Pattern Match bit slice input source type.

typedef enum *_pint_pmatch_bslice* pint_pmatch_bslice_t
PINT Pattern Match bit slice type.

typedef enum *_pint_pmatch_bslice_cfg* pint_pmatch_bslice_cfg_t
PINT Pattern Match configuration type.

typedef struct *_pint_status* pint_status_t
PINT event status.

typedef void (*pint_cb_t)(*pint_pin_int_t* pintr, *pint_status_t* *status)
PINT Callback function.

typedef struct *_pint_pmatch_cfg* pint_pmatch_cfg_t

void PINT_Init(PINT_Type *base)
Initialize PINT peripheral.

This function initializes the PINT peripheral and enables the clock.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_SetCallback(PINT_Type *base, *pint_cb_t* callback)
Set PINT callback.

This function set the callback for PINT interupt handler.

Parameters

- base – Base address of the PINT peripheral.
- callback – Callback.

Return values
None. –

void PINT_PinInterruptConfig(PINT_Type *base, *pint_pin_int_t* intr, *pint_pin_enable_t* enable)
Configure PINT peripheral pin interrupt.

This function configures a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- intr – Pin interrupt.
- enable – Selects detection logic.

Return values
None. –

void PINT_PinInterruptGetConfig(PINT_Type *base, *pint_pin_int_t* pintr, *pint_pin_enable_t* *enable)
Get PINT peripheral pin interrupt configuration.

This function returns the configuration of a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.
- enable – Pointer to store the detection logic.

Return values
None. –

void PINT_PinInterruptClrStatus(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.

This function clears the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetStatus(PINT_Type *base, *pint_pin_int_t* pintr)

Get Selected pin interrupt status.

This function returns the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
status – = 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

void PINT_PinInterruptClrStatusAll(PINT_Type *base)

Clear all pin interrupts status only when pins were triggered by edge-sensitive.

This function clears the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetStatusAll(PINT_Type *base)

Get all pin interrupts status.

This function returns the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
status – Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active.

static inline void PINT_PinInterruptClrFallFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt fall flag.

This function clears the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetFallFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Get selected pin interrupt fall flag.

This function returns the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
flag – = 0 Falling edge has not been detected. = 1 Falling edge has been detected.

static inline void PINT_PinInterruptClrFallFlagAll(PINT_Type *base)

Clear all pin interrupt fall flags.

This function clears the fall flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetFallFlagAll(PINT_Type *base)

Get all pin interrupt fall flags.

This function returns the fall flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
flags – Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

static inline void PINT_PinInterruptClrRiseFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt rise flag.

This function clears the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetRiseFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Get selected pin interrupt rise flag.

This function returns the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
flag – = 0 Rising edge has not been detected. = 1 Rising edge has been detected.

static inline void PINT_PinInterruptClrRiseFlagAll(PINT_Type *base)

Clear all pin interrupt rise flags.

This function clears the rise flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline uint32_t PINT_PinInterruptGetRiseFlagAll(PINT_Type *base)

Get all pin interrupt rise flags.

This function returns the rise flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

flags – Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

void PINT_PatternMatchConfig(PINT_Type *base, *pint_pmatch_bslice_t* bslice, *pint_pmatch_cfg_t* *cfg)

Configure PINT pattern match.

This function configures a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

void PINT_PatternMatchGetConfig(PINT_Type *base, *pint_pmatch_bslice_t* bslice, *pint_pmatch_cfg_t* *cfg)

Get PINT pattern match configuration.

This function returns the configuration of a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

static inline uint32_t PINT_PatternMatchGetStatus(PINT_Type *base, *pint_pmatch_bslice_t* bslice)

Get pattern match bit slice status.

This function returns the status of selected bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.

Return values
status – = 0 Match has not been detected. = 1 Match has been detected.

static inline uint32_t PINT_PatternMatchGetStatusAll(PINT_Type *base)

Get status of all pattern match bit slices.

This function returns the status of all bit slices.

Parameters

- base – Base address of the PINT peripheral.

Return values
status – Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected.

uint32_t PINT_PatternMatchResetDetectLogic(PINT_Type *base)

Reset pattern match detection logic.

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

- base – Base address of the PINT peripheral.

Return values
pmstatus – Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.

static inline void PINT_PatternMatchEnable(PINT_Type *base)

Enable pattern match function.

This function enables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline void PINT_PatternMatchDisable(PINT_Type *base)

Disable pattern match function.

This function disables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline void PINT_PatternMatchEnableRXEV(PINT_Type *base)

Enable RXEV output.

This function enables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline void PINT_PatternMatchDisableRXEV(PINT_Type *base)

Disable RXEV output.

This function disables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_EnableCallback(PINT_Type *base)

Enable callback.

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_DisableCallback(PINT_Type *base)

Disable callback.

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

- base – Base address of the peripheral.

Return values
None. –

void PINT_Deinit(PINT_Type *base)

Deinitialize PINT peripheral.

This function disables the PINT clock.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_EnableCallbackByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)

enable callback by pin index.

This function enables callback by pin index instead of enabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values
None. –

void PINT_DisableCallbackByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)

disable callback by pin index.

This function disables callback by pin index instead of disabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values
None. –

PINT_USE_LEGACY_CALLBACK

PININT_BITSLICE_SRC_START

PININT_BITSLICE_SRC_MASK

PININT_BITSLICE_CFG_START

PININT_BITSLICE_CFG_MASK

PININT_BITSLICE_ENDP_MASK

PINT_PIN_INT_LEVEL

PINT_PIN_INT_EDGE

PINT_PIN_INT_FALL_OR_HIGH_LEVEL

PINT_PIN_INT_RISE

PINT_PIN_RISE_EDGE

PINT_PIN_FALL_EDGE

PINT_PIN_BOTH_EDGE

PINT_PIN_LOW_LEVEL

PINT_PIN_HIGH_LEVEL

struct _pint_status
#include <fsl_pint.h> PINT event status.

struct _pint_pmatch_cfg
#include <fsl_pint.h>

2.87 PLU: Programmable Logic Unit

void PLU_Init(PLU_Type *base)
Enable the PLU clock and reset the module.

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

Parameters

- base – PLU peripheral base address

void PLU_Deinit(PLU_Type *base)
Gate the PLU clock.

Parameters

- base – PLU peripheral base address

static inline void PLU_SetLutInputSource(PLU_Type *base, *plu_lut_index_t* lutIndex, *plu_lut_in_index_t* lutInIndex, *plu_lut_input_source_t* inputSrc)
Set Input source of LUT.

Note: An external clock must be applied to the PLU_CLKIN input when using FFs. For each LUT, the slot associated with the output from LUTn itself is tied low.

Parameters

- base – PLU peripheral base address.
- lutIndex – LUT index (see plu_lut_index_t typedef enumeration).
- lutInIndex – LUT input index (see plu_lut_in_index_t typedef enumeration).
- inputSrc – LUT input source (see plu_lut_input_source_t typedef enumeration).

static inline void PLU_SetOutputSource(PLU_Type *base, *plu_output_index_t* outputIndex, *plu_output_source_t* outputSrc)

Set Output source of PLU.

Note: An external clock must be applied to the PLU_CLKIN input when using FFs.

Parameters

- base – PLU peripheral base address.
- outputIndex – PLU output index (see plu_output_index_t typedef enumeration).
- outputSrc – PLU output source (see plu_output_source_t typedef enumeration).

static inline void PLU_SetLutTruthTable(PLU_Type *base, *plu_lut_index_t* lutIndex, uint32_t truthTable)

Set Truth Table of LUT.

Parameters

- base – PLU peripheral base address.
- lutIndex – LUT index (see plu_lut_index_t typedef enumeration).
- truthTable – Truth Table value.

static inline uint32_t PLU_ReadOutputState(PLU_Type *base)

Read the current state of the 8 designated PLU Outputs.

Note: The PLU bus clock must be re-enabled prior to reading the Outpus Register if PLU bus clock is shut-off.

Parameters

- base – PLU peripheral base address.

Returns

Current PLU output state value.

void PLU_GetDefaultWakeIntConfig(*plu_wakeint_config_t* *config)

Gets an available pre-defined settings for wakeup/interrupt control.

This function initializes the initial configuration structure with an available settings. The default values are:

```
config->filterMode = kPLU_WAKEINT_FILTER_MODE_BYPASS;
config->clockSource = kPLU_WAKEINT_FILTER_CLK_SRC_1MHZ_LPOSC;
```

Parameters

- config – Pointer to configuration structure.

void PLU_EnableWakeIntRequest(PLU_Type *base, uint32_t interruptMask, const *plu_wakeint_config_t* *config)

Enable PLU outputs wakeup/interrupt request.

This function enables Any of the eight selected PLU outputs to contribute to an asynchronous wake-up or an interrupt request.

Note: If a PLU_CLKIN is provided, the raw wake-up/interrupt request will be set on the rising-edge of the PLU_CLKIN whenever the raw request signal is high. This registered signal will be glitch-free and just use the default wakeint config by PLU_GetDefaultWakeIntConfig(). If not, have to specify the filter mode and clock source to eliminate the glitches caused by long and widely disparate delays through the network of LUTs making up the PLU. This way may increase power consumption in low-power operating modes and inject delay before the wake-up/interrupt request is generated.

Parameters

- base – PLU peripheral base address.
- interruptMask – PLU interrupt mask (see _plu_interrupt_mask enumeration).
- config – Pointer to configuration structure (see plu_wakeint_config_t typedef enumeration)

static inline void PLU_LatchInterrupt(PLU_Type *base)

Latch an interrupt.

This function latches the interrupt and then it can be cleared with PLU_ClearLatchedInterrupt().

Note: This mode is not compatible with use of the glitch filter. If this bit is set, the FILTER MODE should be set to kPLU_WAKEINT_FILTER_MODE_BYPASS (Bypass Mode) and PLU_CLKIN should be provided. If this bit is set, the wake-up/interrupt request will be set on the rising-edge of PLU_CLKIN whenever the raw wake-up/interrupt signal is high. The request must be cleared by software.

Parameters

- base – PLU peripheral base address.

void PLU_ClearLatchedInterrupt(PLU_Type *base)

Clear the latched interrupt.

This function clears the wake-up/interrupt request flag latched by PLU_LatchInterrupt()

Note: It is not necessary for the PLU bus clock to be enabled in order to write-to or read-back this bit.

Parameters

- base – PLU peripheral base address.

FSL_PLU_DRIVER_VERSION

Version 2.2.1

enum _plu_lut_index

Index of LUT.

Values:

enumerator kPLU_LUT_0

5-input Look-up Table 0

enumerator kPLU_LUT_1

5-input Look-up Table 1

enumerator kPLU_LUT_2
5-input Look-up Table 2

enumerator kPLU_LUT_3
5-input Look-up Table 3

enumerator kPLU_LUT_4
5-input Look-up Table 4

enumerator kPLU_LUT_5
5-input Look-up Table 5

enumerator kPLU_LUT_6
5-input Look-up Table 6

enumerator kPLU_LUT_7
5-input Look-up Table 7

enumerator kPLU_LUT_8
5-input Look-up Table 8

enumerator kPLU_LUT_9
5-input Look-up Table 9

enumerator kPLU_LUT_10
5-input Look-up Table 10

enumerator kPLU_LUT_11
5-input Look-up Table 11

enumerator kPLU_LUT_12
5-input Look-up Table 12

enumerator kPLU_LUT_13
5-input Look-up Table 13

enumerator kPLU_LUT_14
5-input Look-up Table 14

enumerator kPLU_LUT_15
5-input Look-up Table 15

enumerator kPLU_LUT_16
5-input Look-up Table 16

enumerator kPLU_LUT_17
5-input Look-up Table 17

enumerator kPLU_LUT_18
5-input Look-up Table 18

enumerator kPLU_LUT_19
5-input Look-up Table 19

enumerator kPLU_LUT_20
5-input Look-up Table 20

enumerator kPLU_LUT_21
5-input Look-up Table 21

enumerator kPLU_LUT_22
5-input Look-up Table 22

enumerator kPLU_LUT_23
5-input Look-up Table 23

enumerator kPLU_LUT_24
5-input Look-up Table 24

enumerator kPLU_LUT_25
5-input Look-up Table 25

enum _plu_lut_in_index
Inputs of LUT. 5 input present for each LUT.

Values:

enumerator kPLU_LUT_IN_0
LUT input 0

enumerator kPLU_LUT_IN_1
LUT input 1

enumerator kPLU_LUT_IN_2
LUT input 2

enumerator kPLU_LUT_IN_3
LUT input 3

enumerator kPLU_LUT_IN_4
LUT input 4

enum _plu_lut_input_source
Available sources of LUT input.

Values:

enumerator kPLU_LUT_IN_SRC_PLU_IN_0
Select PLU input 0 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_1
Select PLU input 1 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_2
Select PLU input 2 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_3
Select PLU input 3 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_4
Select PLU input 4 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_5
Select PLU input 5 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_0
Select LUT output 0 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_1
Select LUT output 1 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_2
Select LUT output 2 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_3
Select LUT output 3 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_4
Select LUT output 4 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_5
Select LUT output 5 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_6
Select LUT output 6 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_7
Select LUT output 7 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_8
Select LUT output 8 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_9
Select LUT output 9 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_10
Select LUT output 10 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_11
Select LUT output 11 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_12
Select LUT output 12 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_13
Select LUT output 13 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_14
Select LUT output 14 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_15
Select LUT output 15 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_16
Select LUT output 16 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_17
Select LUT output 17 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_18
Select LUT output 18 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_19
Select LUT output 19 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_20
Select LUT output 20 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_21
Select LUT output 21 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_22
Select LUT output 22 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_23
Select LUT output 23 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_24
Select LUT output 24 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_25
Select LUT output 25 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_0
Select Flip-Flops state 0 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_1
Select Flip-Flops state 1 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_2
Select Flip-Flops state 2 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_3
Select Flip-Flops state 3 to be connected to LUTn Input x

enum _plu_output_index
PLU output multiplexer registers.

Values:

enumerator kPLU_OUTPUT_0
PLU OUTPUT 0

enumerator kPLU_OUTPUT_1
PLU OUTPUT 1

enumerator kPLU_OUTPUT_2
PLU OUTPUT 2

enumerator kPLU_OUTPUT_3
PLU OUTPUT 3

enumerator kPLU_OUTPUT_4
PLU OUTPUT 4

enumerator kPLU_OUTPUT_5
PLU OUTPUT 5

enumerator kPLU_OUTPUT_6
PLU OUTPUT 6

enumerator kPLU_OUTPUT_7
PLU OUTPUT 7

enum _plu_output_source
Available sources of PLU output.

Values:

enumerator kPLU_OUT_SRC_LUT_0
Select LUT0 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_1
Select LUT1 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_2
Select LUT2 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_3
Select LUT3 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_4
Select LUT4 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_5
Select LUT5 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_6
Select LUT6 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_7
Select LUT7 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_8
Select LUT8 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_9
Select LUT9 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_10
Select LUT10 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_11
Select LUT11 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_12
Select LUT12 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_13
Select LUT13 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_14
Select LUT14 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_15
Select LUT15 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_16
Select LUT16 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_17
Select LUT17 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_18
Select LUT18 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_19
Select LUT19 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_20
Select LUT20 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_21
Select LUT21 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_22
Select LUT22 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_23
Select LUT23 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_24
Select LUT24 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_25
Select LUT25 output to be connected to PLU output

enumerator kPLU_OUT_SRC_FLIPFLOP_0
Select Flip-Flops state(0) to be connected to PLU output

enumerator kPLU_OUT_SRC_FLIPFLOP_1
Select Flip-Flops state(1) to be connected to PLU output

enumerator kPLU_OUT_SRC_FLIPFLOP_2
Select Flip-Flops state(2) to be connected to PLU output

enumerator kPLU_OUT_SRC_FLIPFLOP_3
Select Flip-Flops state(3) to be connected to PLU output

enum _plu_interrupt_mask
The enumerator of PLU Interrupt.

Values:

enumerator kPLU_OUTPUT_0_INTERRUPT_MASK
Select PLU output 0 contribute to interrupt/wake-up generation

enumerator kPLU_OUTPUT_1_INTERRUPT_MASK
Select PLU output 1 contribute to interrupt/wake-up generation

enumerator kPLU_OUTPUT_2_INTERRUPT_MASK
Select PLU output 2 contribute to interrupt/wake-up generation

enumerator kPLU_OUTPUT_3_INTERRUPT_MASK
Select PLU output 3 contribute to interrupt/wake-up generation

enumerator kPLU_OUTPUT_4_INTERRUPT_MASK
Select PLU output 4 contribute to interrupt/wake-up generation

enumerator kPLU_OUTPUT_5_INTERRUPT_MASK
Select PLU output 5 contribute to interrupt/wake-up generation

enumerator kPLU_OUTPUT_6_INTERRUPT_MASK
Select PLU output 6 contribute to interrupt/wake-up generation

enumerator kPLU_OUTPUT_7_INTERRUPT_MASK
Select PLU output 7 contribute to interrupt/wake-up generation

enum _plu_wakeint_filter_mode
Control input of the PLU, add filtering for glitch.

Values:

enumerator kPLU_WAKEINT_FILTER_MODE_BYPASS
Select Bypass mode

enumerator kPLU_WAKEINT_FILTER_MODE_1_CLK_PERIOD
Filter 1 clock period

enumerator kPLU_WAKEINT_FILTER_MODE_2_CLK_PERIOD
Filter 2 clock period

enumerator kPLU_WAKEINT_FILTER_MODE_3_CLK_PERIOD
Filter 3 clock period

enum _plu_wakeint_filter_clock_source
Clock source for filter mode.

Values:

enumerator kPLU_WAKEINT_FILTER_CLK_SRC_1MHZ_LPOSC
Select the 1MHz low-power oscillator as the filter clock

enumerator kPLU_WAKEINT_FILTER_CLK_SRC_12MHZ_FRO
Select the 12MHz FRO as the filer clock

enumerator kPLU_WAKEINT_FILTER_CLK_SRC_ALT
Select a third clock source

typedef enum *_plu_lut_index* plu_lut_index_t
Index of LUT.

typedef enum *_plu_lut_in_index* plu_lut_in_index_t
Inputs of LUT. 5 input present for each LUT.

typedef enum *_plu_lut_input_source* plu_lut_input_source_t
Available sources of LUT input.

typedef enum *_plu_output_index* plu_output_index_t
PLU output multiplexer registers.

typedef enum *_plu_output_source* plu_output_source_t
Available sources of PLU output.

typedef enum *_plu_wakeint_filter_mode* plu_wakeint_filter_mode_t
Control input of the PLU, add filtering for glitch.

typedef enum *_plu_wakeint_filter_clock_source* plu_wakeint_filter_clock_source_t
Clock source for filter mode.

typedef struct *_plu_wakeint_config* plu_wakeint_config_t
Wake configuration.

struct _plu_wakeint_config
#include <fsl_plu.h> Wake configuration.

Public Members

plu_wakeint_filter_mode_t filterMode
Filter Mode.

plu_wakeint_filter_clock_source_t clockSource
The clock source for filter mode.

2.88 PORT: Port Control and Interrupts

static inline void PORT_GetVersionInfo(PORT_Type *base, *port_version_info_t* *info)
Get PORT version information.

Parameters

- base – PORT peripheral base pointer
- info – PORT version information

static inline void PORT_SecletPortVoltageRange(PORT_Type *base, *port_voltage_range_t* range)

Get PORT version information.

Note: : PORTA_CONFIG[RANGE] controls the voltage ranges of Port A, B, and C. Read or write PORTB_CONFIG[RANGE] and PORTC_CONFIG[RANGE] does not take effect.

Parameters

- base – PORT peripheral base pointer
- range – port voltage range

static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const *port_pin_config_t* *config)

Sets the port PCR register.

This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT PCR register configuration structure.

static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const *port_pin_config_t* *config)

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```
Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

static inline void PORT_SetMultipleInterruptPinsConfig(PORT_Type *base, uint32_t mask, port_interrupt_t config)

Sets the port interrupt configuration in PCR register for multiple pins.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT pin interrupt configuration.
  - #kPORT_InterruptOrDMADisabled: Interrupt/DMA request disabled.
  - #kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - #kPORT_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - #kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - #kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - #kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - #kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - #kPORT_InterruptLogicZero : Interrupt when logic zero.
  - #kPORT_InterruptRisingEdge : Interrupt on rising edge.
  - #kPORT_InterruptFallingEdge: Interrupt on falling edge.
  - #kPORT_InterruptEitherEdge : Interrupt on either edge.
  - #kPORT_InterruptLogicOne : Interrupt when logic one.
  - #kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
  - #kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

static inline void PORT_SetPinMux(PORT_Type *base, uint32_t pin, *port_mux_t* mux)

Configures the pin muxing.

Note: : This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

---

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- mux – pin muxing slot selection.
  - kPORT_PinDisabledOrAnalog: Pin disabled or work in analog function.
  - kPORT_MuxAsGpio : Set as GPIO.
  - kPORT_MuxAlt2 : chip-specific.
  - kPORT_MuxAlt3 : chip-specific.
  - kPORT_MuxAlt4 : chip-specific.

- kPORT_MuxAlt5 : chip-specific.
- kPORT_MuxAlt6 : chip-specific.
- kPORT_MuxAlt7 : chip-specific.

static inline void PORT_EnablePinsDigitalFilter(PORT_Type *base, uint32_t mask, bool enable)

Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

static inline void PORT_SetDigitalFilterConfig(PORT_Type *base, const *port_digital_filter_config_t* *config)

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

static inline void PORT_SetPinDriveStrength(PORT_Type *base, uint32_t pin, uint8_t strength)

Configures the port pin drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- strength – PORT pin drive strength
  - kPORT_LowDriveStrength = 0U - Low-drive strength is configured.
  - kPORT_HighDriveStrength = 1U - High-drive strength is configured.

static inline void PORT_EnablePinDoubleDriveStrength(PORT_Type *base, uint32_t pin, bool enable)

Enables the port pin double drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- enable – PORT pin drive strength configuration.

static inline void PORT_SetPinPullValue(PORT_Type *base, uint32_t pin, uint8_t value)

Configures the port pin pull value.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- value – PORT pin pull value
  - kPORT_LowPullResistor = 0U - Low internal pull resistor value is selected.
  - kPORT_HighPullResistor = 1U - High internal pull resistor value is selected.

static inline uint32_t PORT_GetEFTDetectFlags(PORT_Type *base)

Get EFT detect flags.

Parameters

- base – PORT peripheral base pointer

Returns

EFT detect flags

static inline void PORT_EnableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)

Enable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_DisableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)

Disable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_ClearAllLowEFTDetectors(PORT_Type *base)

Clear all low EFT detector.

Note: : Port B and Port C pins share the same EFT detector clear control from PORTC_EDCR register. Any write to the PORTB_EDCR does not take effect.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_ClearAllHighEFTDetectors(PORT_Type *base)

Clear all high EFT detector.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

FSL_PORT_DRIVER_VERSION

PORT driver version.

enum _port_pull

Internal resistor pull feature selection.

Values:

enumerator kPORT_PullDisable

Internal pull-up/down resistor is disabled.

enumerator kPORT_PullDown

Internal pull-down resistor is enabled.

enumerator kPORT_PullUp

Internal pull-up resistor is enabled.

enum _port_pull_value

Internal resistor pull value selection.

Values:

enumerator kPORT_LowPullResistor

Low internal pull resistor value is selected.

enumerator kPORT_HighPullResistor

High internal pull resistor value is selected.

enum _port_slew_rate

Slew rate selection.

Values:

enumerator kPORT_FastSlewRate

Fast slew rate is configured.

enumerator kPORT_SlowSlewRate

Slow slew rate is configured.

enum _port_open_drain_enable

Open Drain feature enable/disable.

Values:

enumerator kPORT_OpenDrainDisable

Open drain output is disabled.

enumerator kPORT_OpenDrainEnable

Open drain output is enabled.

enum _port_passive_filter_enable

Passive filter feature enable/disable.

Values:

enumerator kPORT_PassiveFilterDisable

Passive input filter is disabled.

enumerator kPORT_PassiveFilterEnable

Passive input filter is enabled.

enum _port_drive_strength

Configures the drive strength.

Values:

enumerator kPORT_LowDriveStrength

Low-drive strength is configured.

enumerator kPORT_HighDriveStrength

High-drive strength is configured.

enum _port_drive_strength1

Configures the drive strength1.

Values:

enumerator kPORT_NormalDriveStrength

Normal drive strength

enumerator kPORT_DoubleDriveStrength

Double drive strength

enum _port_input_buffer

input buffer disable/enable.

Values:

enumerator kPORT_InputBufferDisable

Digital input is disabled

enumerator kPORT_InputBufferEnable

Digital input is enabled

enum _port_invet_input

Digital input is not inverted or it is inverted.

Values:

enumerator kPORT_InputNormal

Digital input is not inverted

enumerator kPORT_InputInvert

Digital input is inverted

enum _port_lock_register

Unlock/lock the pin control register field[15:0].

Values:

enumerator kPORT_UnlockRegister

Pin Control Register fields [15:0] are not locked.

enumerator kPORT_LockRegister

Pin Control Register fields [15:0] are locked.

enum _port_mux

Pin mux selection.

Values:

enumerator kPORT_PinDisabledOrAnalog

Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT_MuxAsGpio

Corresponding pin is configured as GPIO.

enumerator kPORT_MuxAlt0

Chip-specific

enumerator kPORT_MuxAlt1

Chip-specific

enumerator kPORT_MuxAlt2

Chip-specific

enumerator kPORT_MuxAlt3

Chip-specific

enumerator kPORT_MuxAlt4

Chip-specific

enumerator kPORT_MuxAlt5

Chip-specific

enumerator kPORT_MuxAlt6

Chip-specific

enumerator kPORT_MuxAlt7
Chip-specific

enumerator kPORT_MuxAlt8
Chip-specific

enumerator kPORT_MuxAlt9
Chip-specific

enumerator kPORT_MuxAlt10
Chip-specific

enumerator kPORT_MuxAlt11
Chip-specific

enumerator kPORT_MuxAlt12
Chip-specific

enumerator kPORT_MuxAlt13
Chip-specific

enumerator kPORT_MuxAlt14
Chip-specific

enumerator kPORT_MuxAlt15
Chip-specific

enum _port_digital_filter_clock_source
Digital filter clock source selection.

Values:

enumerator kPORT_BusClock
Digital filters are clocked by the bus clock.

enumerator kPORT_LpoClock
Digital filters are clocked by the 1 kHz LPO clock.

enum _port_voltage_range
PORT voltage range.

Values:

enumerator kPORT_VoltageRange1Dot71V_3Dot6V
Port voltage range is 1.71 V - 3.6 V.

enumerator kPORT_VoltageRange2Dot70V_3Dot6V
Port voltage range is 2.70 V - 3.6 V.

typedef enum *_port_mux* port_mux_t
Pin mux selection.

typedef enum *_port_digital_filter_clock_source* port_digital_filter_clock_source_t
Digital filter clock source selection.

typedef struct *_port_digital_filter_config* port_digital_filter_config_t
PORT digital filter feature configuration definition.

typedef struct *_port_pin_config* port_pin_config_t
PORT pin configuration structure.

typedef struct *_port_version_info* port_version_info_t
PORT version information.

typedef enum *_port_voltage_range* port_voltage_range_t
PORT voltage range.

FSL_COMPONENT_ID

struct _port_digital_filter_config
#include <fsl_port.h> PORT digital filter feature configuration definition.

Public Members

uint32_t digitalFilterWidth
Set digital filter width

port_digital_filter_clock_source_t clockSource
Set digital filter clockSource

struct _port_pin_config
#include <fsl_port.h> PORT pin configuration structure.

Public Members

uint16_t pullSelect
No-pull/pull-down/pull-up select

uint16_t pullValueSelect
Pull value select

uint16_t slewRate
Fast/slow slew rate Configure

uint16_t passiveFilterEnable
Passive filter enable/disable

uint16_t openDrainEnable
Open drain enable/disable

uint16_t driveStrength
Fast/slow drive strength configure

uint16_t driveStrength1
Normal/Double drive strength enable/disable

uint16_t inputBuffer
Input Buffer Configure

uint16_t invertInput
Invert Input Configure

uint16_t lockRegister
Lock/unlock the PCR field[15:0]

struct _port_version_info
#include <fsl_port.h> PORT version information.

Public Members

uint16_t feature
Feature Specification Number.

uint8_t minor
Minor Version Number.

uint8_t major
Major Version Number.

2.89 POWERQUAD: PowerQuad hardware accelerator

void PQ_GetDefaultConfig(*pq_config_t* *config)
Get default configuration.

This function initializes the POWERQUAD configuration structure to a default value. FORMAT register field definitions Bits[15:8] scaler (for scaled 'q31' formats) Bits[5:4] external format. 00b=q15, 01b=q31, 10b=float Bits[1:0] internal format. 00b=q15, 01b=q31, 10b=float POWERQUAD->INAFORMAT = (config->inputAPrescale « 8U) | (config->inputAFormat « 4U) | config->machineFormat

For all Powerquad operations internal format must be float (with the only exception being the FFT related functions, ie FFT/IFFT/DCT/IDCT which must be set to q31). The default values are: config->inputAFormat = kPQ_Float; config->inputAPrescale = 0; config->inputBFormat = kPQ_Float; config->inputBPrescale = 0; config->outputFormat = kPQ_Float; config->outputPrescale = 0; config->tmpFormat = kPQ_Float; config->tmpPrescale = 0; config->machineFormat = kPQ_Float; config->tmpBase = 0xE0000000;

Parameters

- config – Pointer to "pq_config_t" structure.

void PQ_SetConfig(POWERQUAD_Type *base, const *pq_config_t* *config)
Set configuration with format/prescale.

Parameters

- base – POWERQUAD peripheral base address
- config – Pointer to "pq_config_t" structure.

static inline void PQ_SetCoprocessorScaler(POWERQUAD_Type *base, const *pq_prescale_t* *prescale)
set coprocessor scaler for coprocessor instructions, this function is used to set output saturation and scaleing for input/output.

Parameters

- base – POWERQUAD peripheral base address
- prescale – Pointer to "pq_prescale_t" structure.

void PQ_Init(POWERQUAD_Type *base)
Initializes the POWERQUAD module.

Parameters

- base – POWERQUAD peripheral base address.

void PQ_Deinit(POWERQUAD_Type *base)
De-initializes the POWERQUAD module.

Parameters

- base – POWERQUAD peripheral base address.

void PQ_SetFormat(POWERQUAD_Type *base, *pq_computationengine_t* engine, *pq_format_t* format)

Set format for non-coprecessor instructions.

Parameters

- base – POWERQUAD peripheral base address
- engine – Computation engine
- format – Data format

static inline void PQ_WaitDone(POWERQUAD_Type *base)

Wait for the completion.

Parameters

- base – POWERQUAD peripheral base address

static inline void PQ_LnF32(float *pSrc, float *pDst)

Processing function for the floating-point natural log.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_InvF32(float *pSrc, float *pDst)

Processing function for the floating-point reciprocal.

Parameters

- *pSrc – points to the block of input data. The range of the input value is non-zero.
- *pDst – points to the block of output data

static inline void PQ_SqrtF32(float *pSrc, float *pDst)

Processing function for the floating-point square-root.

Parameters

- *pSrc – points to the block of input data. The range of the input value is [0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_InvSqrtF32(float *pSrc, float *pDst)

Processing function for the floating-point inverse square-root.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_EtoxF32(float *pSrc, float *pDst)

Processing function for the floating-point natural exponent.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_EtonxF32(float *pSrc, float *pDst)

Processing function for the floating-point natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_SinF32(float *pSrc, float *pDst)

Processing function for the floating-point sine.

Parameters

- *pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_CosF32(float *pSrc, float *pDst)

Processing function for the floating-point cosine.

Parameters

- *pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_BiquadF32(float *pSrc, float *pDst)

Processing function for the floating-point biquad.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data

static inline void PQ_DivF32(float *x1, float *x2, float *pDst)

Processing function for the floating-point division.

Get x1 / x2.

Parameters

- x1 – x1
- x2 – x2
- *pDst – points to the block of output data

static inline void PQ_Biquad1F32(float *pSrc, float *pDst)

Processing function for the floating-point biquad.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data

static inline int32_t PQ_LnFixed(int32_t val)

Processing function for the fixed natural log.

Parameters

- val – value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns ln(val).

static inline int32_t PQ_InvFixed(int32_t val)

Processing function for the fixed reciprocal.

Parameters

- val – value to be calculated. The range of the input value is non-zero.

Returns

returns inv(val).

static inline uint32_t PQ_SqrtFixed(uint32_t val)

Processing function for the fixed square-root.

Parameters

- val – value to be calculated. The range of the input value is [0 +INFINITY).

Returns

returns sqrt(val).

static inline int32_t PQ_InvSqrtFixed(int32_t val)

Processing function for the fixed inverse square-root.

Parameters

- val – value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns 1/sqrt(val).

static inline int32_t PQ_EtoxFixed(int32_t val)

Processing function for the Fixed natural exponent.

Parameters

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etox^(val).

static inline int32_t PQ_EtonxFixed(int32_t val)

Processing function for the fixed natural exponent with negative parameter.

Parameters

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etonx^(val).

static inline int32_t PQ_SinQ31(int32_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns

returns sin(val).

static inline int16_t PQ_SinQ15(int16_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns
returns sin(val).

static inline int32_t PQ_CosQ31(int32_t val)

Processing function for the fixed cosine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns
returns cos(val).

static inline int16_t PQ_CosQ15(int16_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns
returns sin(val).

static inline int32_t PQ_BiquadFixed(int32_t val)

Processing function for the fixed biquad.

Parameters

- val – value to be calculated

Returns
returns biquad(val).

void PQ_VectorLnF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorLnFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinQ15(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the Q15 vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosQ15(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the Q15 vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinQ31(int32_t *pSrc, int32_t *pDst, int32_t length)
Processing function for the Q31 vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosQ31(int32_t *pSrc, int32_t *pDst, int32_t length)
Processing function for the Q31 vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorLnFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)
Processing function for the 16-bit integer vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)
Processing function for the 16-bit integer vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)
Processing function for the 16-bit integer vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)
Processing function for the 16-bit integer vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)
Processing function for the 16-bit integer vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data

- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorBiquadDf2F32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data.

void PQ_VectorBiquadDf2Fixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadDf2Fixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2F32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2Fixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2Fixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

int32_t PQ_ArctanFixed(POWERQUAD_Type *base, int32_t x, int32_t y, *pq_cordic_iter_t* iteration)

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, the behavior is like c function atan.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the arctan(0.5), the result of PQ_ArctanFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_ArctanFixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is in the range of -2^26 to 2^26, which means -pi/2 to pi/2.

int32_t PQ_ArctanhFixed(POWERQUAD_Type *base, int32_t x, int32_t y, *pq_cordic_iter_t* iteration)

Processing function for the fixed inverse trigonometric.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the arctanh(0.5), the result of PQ_ArctanhFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_ArctanhFixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is radians, 2^27 means pi. The range is -1.118 to 1.118 radians.

int32_t PQ_Arctan2Fixed(POWERQUAD_Type *base, int32_t x, int32_t y, *pq_cordic_iter_t* iteration)

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, it calculates the angle in radians for the quadrant. The behavior is like c function atan2.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the arctan(0.5), the result of PQ_Arctan2Fixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_Arctan2Fixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is in the range of -2^27 to 2^27, which means -pi to pi.

static inline int32_t PQ_Biquad1Fixed(int32_t val)

Processing function for the fixed biquad.

Parameters

- val – value to be calculated

Returns

returns biquad(val).

void PQ_TransformCFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the complex FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformRFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the real FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformIFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the inverse complex FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformCDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the complex DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformRDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the real DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformIDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the inverse complex DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_BiquadBackUpInternalState(POWERQUAD_Type *base, int32_t biquad_num, *pq_biquad_state_t* *state)

Processing function for backup biquad context.

Parameters

- base – POWERQUAD peripheral base address
- biquad_num – biquad side
- state – point to states.

void PQ_BiquadRestoreInternalState(POWERQUAD_Type *base, int32_t biquad_num, *pq_biquad_state_t* *state)

Processing function for restore biquad context.

Parameters

- base – POWERQUAD peripheral base address
- biquad_num – biquad side
- state – point to states.

void PQ_BiquadCascadeDf2Init(*pq_biquad_cascade_df2_instance* *S, uint8_t numStages, *pq_biquad_state_t* *pState)

Initialization function for the direct form II Biquad cascade filter.

Parameters

- *S – **[inout]** points to an instance of the filter data structure.
- numStages – **[in]** number of 2nd order stages in the filter.
- *pState – **[in]** points to the state buffer.

void PQ_BiquadCascadeDf2F32(const *pq_biquad_cascade_df2_instance* *S, float *pSrc, float *pDst, uint32_t blockSize)

Processing function for the floating-point direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

void PQ_BiquadCascadeDf2Fixed32(const *pq_biquad_cascade_df2_instance* *S, int32_t *pSrc, int32_t *pDst, uint32_t blockSize)

Processing function for the Q31 direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

void PQ_BiquadCascadeDf2Fixed16(const *pq_biquad_cascade_df2_instance* *S, int16_t *pSrc, int16_t *pDst, uint32_t blockSize)

Processing function for the Q15 direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

void PQ_FIR(POWERQUAD_Type *base, const void *pAData, uint16_t ALength, const void *pBData, uint16_t BLength, void *pResult, uint32_t opType)

Processing function for the FIR.

Parameters

- base – POWERQUAD peripheral base address
- pAData – the first input sequence
- ALength – number of the first input sequence

- pBData – the second input sequence
- BLength – number of the second input sequence
- pResult – array for the output data
- opType – operation type, could be PQ_FIR_FIR, PQ_FIR_CONVOLUTION, PQ_FIR_CORRELATION.

void PQ_FIRIncrement(POWERQUAD_Type *base, uint16_t ALength, uint16_t BLength, int32_t xOffset)

Processing function for the incremental FIR. This function can be used after pq_fir() for incremental FIR operation when new x data are available.

Parameters

- base – POWERQUAD peripheral base address
- ALength – number of input samples
- BLength – number of taps
- xOffset – offset for number of input samples

void PQ_MatrixAddition(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix addition.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_MatrixSubtraction(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix subtraction.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_MatrixMultiplication(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix multiplication.

Parameters

- base – POWERQUAD peripheral base address

- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_MatrixProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix product.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_VectorDotProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the vector dot product.

Parameters

- base – POWERQUAD peripheral base address
- length – length of vector
- pAData – input vector A
- pBData – input vector B
- pResult – array for the output data.

void PQ_MatrixInversion(POWERQUAD_Type *base, uint32_t length, void *pData, void *pTmpData, void *pResult)

Processing function for the matrix inverse.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pData – input matrix
- pTmpData – input temporary matrix, pTmpData length not less than pData lenght and 1024 words is sufficient for the largest supported matrix.
- pResult – array for the output data, round down for fixed point.

void PQ_MatrixTranspose(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the matrix transpose.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pData – input matrix
- pResult – array for the output data.

void PQ_MatrixScale(POWERQUAD_Type *base, uint32_t length, float misc, const void *pData, void *pResult)

Processing function for the matrix scale.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- misc – scaling parameters
- pData – input matrix
- pResult – array for the output data.

FSL_POWERQUAD_DRIVER_VERSION

Version.

enum pq_computationengine_t

powerquad computation engine

Values:

enumerator kPQ_CP_PQ

Math engine.

enumerator kPQ_CP_MTX

Matrix engine.

enumerator kPQ_CP_FFT

FFT engine.

enumerator kPQ_CP_FIR

FIR engine.

enumerator kPQ_CP_CORDIC

CORDIC engine.

enum pq_format_t

powerquad data structure format type

Values:

enumerator kPQ_16Bit

Int16 Fixed point.

enumerator kPQ_32Bit

Int32 Fixed point.

enumerator kPQ_Float
Float point.

enum pq_cordic_iter_t
CORDIC iteration.

Values:

enumerator kPQ_Iteration_8
Iterate 8 times.

enumerator kPQ_Iteration_16
Iterate 16 times.

enumerator kPQ_Iteration_24
Iterate 24 times.

typedef struct *_pq_biquad_param* pq_biquad_param_t
Struct to save biquad parameters.

typedef struct *_pq_biquad_state* pq_biquad_state_t
Struct to save biquad state.

typedef union *_pq_float* pq_float_t
Conversion between integer and float type.

PQ_VectorBiqaudDf2F32

PQ_VectorBiqaudDf2Fixed32

PQ_VectorBiqaudDf2Fixed16

PQ_VectorBiqaudCascadeDf2F32

PQ_VectorBiqaudCascadeDf2Fixed32

PQ_VectorBiqaudCascadeDf2Fixed16

PQ_Vector8BiqaudDf2CascadeF32

PQ_Vector8BiqaudDf2CascadeFixed32

PQ_Vector8BiqaudDf2CascadeFixed16

PQ_FLOAT32

PQ_FIXEDPT

CP_PQ

CP_MTX

CP_FFT

CP_FIR

CP_CORDIC

PQ_TRANS

PQ_TRIG

PQ_BIQUAD

PQ_TRANS_FIXED

PQ_TRIG_FIXED

PQ_BIQUAD_FIXED

PQ_INV

PQ_LN

PQ_SQRT

PQ_INVSQRT

PQ_ETOX

PQ_ETONX

PQ_DIV

PQ_SIN

PQ_COS

PQ_BIQ0_CALC

PQ_BIQ1_CALC

PQ_COMP0_ONLY

PQ_COMP1_ONLY

CORDIC_ITER(**x**)

CORDIC_MIU(**x**)

CORDIC_T(**x**)

CORDIC_ARCTAN

CORDIC_ARCTANH

INST_BUSY

PQ_ERRSTAT_OVERFLOW

PQ_ERRSTAT_NAN

PQ_ERRSTAT_FIXEDOVERFLOW

PQ_ERRSTAT_UNDERFLOW

PQ_TRANS_CFFT

PQ_TRANS_IFFT

PQ_TRANS_CDCT

PQ_TRANS_IDCT

PQ_TRANS_RFFT

PQ_TRANS_RDCT

PQ_MTX_SCALE

PQ_MTX_MULT

PQ_MTX_ADD

PQ_MTX_INV

PQ_MTX_PROD

PQ_MTX_SUB

PQ_VEC_DOTP

PQ_MTX_TRAN

PQ_FIR_FIR

PQ_FIR_CONVOLUTION

PQ_FIR_CORRELATION

PQ_FIR_INCREMENTAL

__pq__ln0(**x**)

__pq__inv0(**x**)

__pq__sqrt0(**x**)

__pq__invsqrt0(**x**)

__pq__etox0(**x**)

__pq__etonx0(**x**)

__pq__sin0(**x**)

__pq__cos0(**x**)

__pq__biquad0(**x**)

__pq__ln__fx0(**x**)

__pq__inv__fx0(**x**)

__pq__sqrt__fx0(**x**)

__pq__invsqrt__fx0(**x**)

__pq__etox__fx0(**x**)

__pq__etonx__fx0(**x**)

__pq__sin__fx0(**x**)

__pq__cos__fx0(**x**)

__pq__biquad0__fx(**x**)

__pq__div0(**x**)

__pq__div1(**x**)

__pq__ln1(**x**)

__pq__inv1(**x**)

__pq__sqrt1(**x**)

__pq__invsqrt1(**x**)

__pq__etox1(**x**)

__pq__etonx1(**x**)

__pq__sin1(**x**)

__pq__cos1(**x**)

__pq__biquad1(**x**)

__pq__ln__fx1(**x**)

__pq__inv__fx1(**x**)

__pq__sqrt__fx1(**x**)

__pq__invsqrt__fx1(**x**)

__pq__etox__fx1(**x**)

__pq__etonx__fx1(**x**)

__pq__sin__fx1(**x**)

__pq__cos__fx1(**x**)

__pq__biquad1__fx(**x**)

__pq__readMult0()

__pq__readAdd0()

__pq__readMult1()

__pq__readAdd1()

__pq__readMult0__fx()

__pq__readAdd0__fx()

__pq__readMult1__fx()

__pq__readAdd1__fx()

PQ__LN__INF
: Parameter used for vector ln(x)

PQ__INV__INF
: Parameter used for vector 1/x

PQ__SQRT__INF
: Parameter used for vector sqrt(x)

PQ__ISQRT__INF
: Parameter used for vector 1/sqrt(x)

PQ__ETOX__INF
: Parameter used for vector e^x

PQ__ETONX__INF
: Parameter used for vector e^(-x)

PQ_SIN_INF
: Parameter used for vector sin(x)

PQ_COS_INF
: Parameter used for vector cos(x)

PQ_RUN_OPCODE_R3_R2(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R5_R4(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R7_R6(BATCH_OPCODE, BATCH_MACHINE)

PQ_Vector8_FP(middle, last, BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

PQ_RUN_OPCODE_R2_R3(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R4_R5(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R6_R7(BATCH_OPCODE, BATCH_MACHINE)

PQ_Vector8_FX(middle, last, BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

PQ_Initiate_Vector_Func(pSrc, pDst)
: Start 32-bit data vector calculation.

  Start the vector calculation, the input data could be float, int32_t or Q31.

  Parameters

  - pSrc – Pointer to the source data.
  - pDst – Pointer to the destination data.

PQ_End_Vector_Func()
: End vector calculation.

  This function should be called after vector calculation.

PQ_StartVector(PSRC, PDST, LENGTH)
: Start 32-bit data vector calculation.

  Start the vector calculation, the input data could be float, int32_t or Q31.

  Parameters

  - PSRC – Pointer to the source data.
  - PDST – Pointer to the destination data.
  - LENGTH – Number of the data, must be multiple of 8.

PQ_StartVectorFixed16(PSRC, PDST, LENGTH)
: Start 16-bit data vector calculation.

  Start the vector calculation, the input data could be int16_t. This function should be use with PQ_Vector8Fixed16.

  Parameters

  - PSRC – Pointer to the source data.
  - PDST – Pointer to the destination data.
  - LENGTH – Number of the data, must be multiple of 8.

PQ_StartVectorQ15(PSRC, PDST, LENGTH)

Start Q15-bit data vector calculation.

Start the vector calculation, the input data could be Q15. This function should be use with PQ_Vector8Q15. This function is dedicate for SinQ15/CosQ15 vector calculation. Because PowerQuad only supports Q31 Sin/Cos fixed function, so the input Q15 data is left shift 16 bits first, after Q31 calculation, the output data is right shift 16 bits.

Parameters

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ_EndVector()

End vector calculation.

This function should be called after vector calculation.

PQ_Vector8F32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Float data vector calculation.

Float data vector calculation, the input data should be float. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Fixed32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 32-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. PQ_SIN_INF, PQ_COS_INF. When this function is used for sin/cos calculation, the input data should be in the format Q1.31. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int32_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Fixed16(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 16-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int16_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
```

(continues on next page)

(continued from previous page)

```
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Q15(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Q15 data vector calculation.

Q15 data vector calculation, this function should only be used for sin/cos Q15 calculation, and the coprocessor output prescaler must be set to 31 before this function. This function loads Q15 data and left shift 16 bits, calculate and right shift 16 bits, then stores to the output array. The input range -1 to 1 means -pi to pi. For example, to calculate sin of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {...}
int16_t output[VECTOR_LEN];
const pq_prescale_t prescale =
{
    .inputPrescale = 0,
    .outputPrescale = 31,
    .outputSaturate = 0
};

PQ_SetCoprocessorScaler(POWERQUAD, const pq_prescale_t *prescale);

PQ_StartVectorQ15(pSrc, pDst, length);
PQ_Vector8Q15(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_DF2_Vector8_FP(middle, last)

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FP(false,false);
PQ_DF2_Vector8_FP(true,true);
PQ_End_Vector_Func();
```

PQ_DF2_Vector8_FX(middle, last)

Fixed data vector biquad direct form II calculation.

Biquad filter, the input and output data are fixed data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FX(false,false);
PQ_DF2_Vector8_FX(true,true);
PQ_End_Vector_Func();
```

PQ_Vector8BiquadDf2F32()

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2F32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2Fixed32()

Fixed 32-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q31 or 32-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
```

(continues on next page)

(continued from previous page)

```
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2Fixed16()

Fixed 16-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q15 or 16-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int16_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed16();
PQ_EndVector();
```

PQ_DF2_Cascade_Vector8_FP(middle, last)

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};
```

(continues on next page)

(continued from previous page)

```
pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FP(false, false);
PQ_DF2_Cascade_Vector8_FP(true, true);
PQ_End_Vector_Func();
```

PQ_DF2_Cascade_Vector8_FX(middle, last)

Fixed data vector direct form II biquad cascade filter.

The input and output data are fixed data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FX(false, false);
PQ_DF2_Cascade_Vector8_FX(true, true);
PQ_End_Vector_Func();
```

PQ_Vector8BiquadDf2CascadeF32()

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeF32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2CascadeFixed32()

Fixed 32-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 32-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
```

(continues on next page)

(continued from previous page)

```
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2CascadeFixed16()

Fixed 16-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 16-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed16();
PQ_EndVector();
```

POWERQUAD_MAKE_MATRIX_LEN(mat1Row, mat1Col, mat2Col)

Make the length used for matrix functions.

PQ_Q31_2_FLOAT(x)

Convert Q31 to float.

PQ__Q15__2__FLOAT(x)

Convert Q15 to float.

struct pq__prescale__t

#include <fsl_powerquad.h> Coprocessor prescale.

Public Members

int8_t inputPrescale

Input prescale.

int8_t outputPrescale

Output prescale.

int8_t outputSaturate

Output saturate at n bits, for example 0x11 is 8 bit space, the value will be truncated at +127 or -128.

struct pq__config__t

#include <fsl_powerquad.h> powerquad data structure format

Public Members

pq_format_t inputAFormat

Input A format.

int8_t inputAPrescale

Input A prescale, for example 1.5 can be 1.5*2^n if you scale by 'shifting' ('scaling' by a factor of n).

pq_format_t inputBFormat

Input B format.

int8_t inputBPrescale

Input B prescale.

pq_format_t outputFormat

Out format.

int8_t outputPrescale

Out prescale.

pq_format_t tmpFormat

Temp format.

int8_t tmpPrescale

Temp prescale.

pq_format_t machineFormat

Machine format.

uint32_t *tmpBase

Tmp base address.

struct __pq__biquad__param

#include <fsl_powerquad.h> Struct to save biquad parameters.

Public Members

float v_n_1
: v[n-1], set to 0 when initialization.

float v_n
: v[n], set to 0 when initialization.

float a_1
: a[1]

float a_2
: a[2]

float b_0
: b[0]

float b_1
: b[1]

float b_2
: b[2]

struct _pq_biquad_state
: *#include <fsl_powerquad.h>* Struct to save biquad state.

Public Members

pq_biquad_param_t param
: Filter parameter.

uint32_t compreg
: Internal register, set to 0 when initialization.

struct pq_biquad_cascade_df2_instance
: *#include <fsl_powerquad.h>* Instance structure for the direct form II Biquad cascade filter.

Public Members

uint8_t numStages
: Number of 2nd order stages in the filter.

pq_biquad_state_t *pState
: Points to the array of state coefficients.

union _pq_float
: *#include <fsl_powerquad.h>* Conversion between integer and float type.

Public Members

float floatX
: Float type.

uint32_t integerX
: Unsigned interger type.

2.90 PRINCE: PRINCE bus crypto engine

FSL_PRINCE_DRIVER_VERSION

PRINCE driver version 2.6.0.

Current version: 2.6.0

Change log:

- Version 2.0.0
  - Initial version.
- Version 2.1.0
  - Update for the A1 rev. of LPC55Sxx serie.
- Version 2.2.0
  - Add runtime checking of the A0 and A1 rev. of LPC55Sxx serie to support both silicone revisions.
- Version 2.3.0
  - Add support for LPC55S1x and LPC55S2x series
- Version 2.3.0
  - Fix MISRA-2012 issues.
- Version 2.3.1
  - Add support for LPC55S0x series
- Version 2.3.2
  - Fix documentation of enumeration. Extend PRINCE example.
- Version 2.4.0
  - Add support for LPC55S3x series
- Version 2.5.0
  - Add PRINCE_Config() and PRINCE_Reconfig() features.
- Version 2.5.1
  - Fix build error due to renamed symbols
- Version 2.6.0
  - Renamed CSS to ELS

enum _skboot_status

Secure status enumeration.

Values:

enumerator kStatus_SKBOOT_Success

PRINCE Success

enumerator kStatus_SKBOOT_Fail

PRINCE Fail

enumerator kStatus_SKBOOT_InvalidArgument

PRINCE Invalid argument

enumerator kStatus_SKBOOT_KeyStoreMarkerInvalid

PRINCE Invalid marker

enum _secure_bool
Secure boolean enumeration.

Values:

enumerator kSECURE_TRUE
PRINCE true

enumerator kSECURE_FALSE
PRINCE false

enum _prince_region
Prince region.

Values:

enumerator kPRINCE_Region0
PRINCE region 0

enumerator kPRINCE_Region1
PRINCE region 1

enumerator kPRINCE_Region2
PRINCE region 2

enum _prince_lock
Prince lock.

Values:

enumerator kPRINCE_Region0Lock
PRINCE region 0 lock

enumerator kPRINCE_Region1Lock
PRINCE region 1 lock

enumerator kPRINCE_Region2Lock
PRINCE region 2 lock

enumerator kPRINCE_MaskLock
PRINCE mask register lock

enum _prince_flags
Prince flag.

Values:

enumerator kPRINCE_Flag_None
PRINCE Flag None

enumerator kPRINCE_Flag_EraseCheck
PRINCE Flag Erase check

enumerator kPRINCE_Flag_WriteCheck
PRINCE Flag Write check

typedef enum *_skboot_status* skboot_status_t
Secure status enumeration.

typedef enum *_secure_bool* secure_bool_t
Secure boolean enumeration.

typedef enum *_prince_region* prince_region_t
Prince region.

typedef enum *_prince_lock* prince_lock_t

Prince lock.

typedef enum *_prince_flags* prince_flags_t

Prince flag.

static inline void PRINCE_EncryptEnable(PRINCE_Type *base)

Enable data encryption.

This function enables PRINCE on-the-fly data encryption.

Parameters

- base – PRINCE peripheral address.

static inline void PRINCE_EncryptDisable(PRINCE_Type *base)

Disable data encryption.

This function disables PRINCE on-the-fly data encryption.

Parameters

- base – PRINCE peripheral address.

static inline bool PRINCE_IsEncryptEnable(PRINCE_Type *base)

Is Enable data encryption.

This function test if PRINCE on-the-fly data encryption is enabled.

Parameters

- base – PRINCE peripheral address.

Returns

true if enabled, false if not

static inline void PRINCE_SetMask(PRINCE_Type *base, uint64_t mask)

Sets PRINCE data mask.

This function sets the PRINCE mask that is used to mask decrypted data.

Parameters

- base – PRINCE peripheral address.
- mask – 64-bit data mask value.

static inline void PRINCE_SetLock(PRINCE_Type *base, uint32_t lock)

Locks access for specified region registers or data mask register.

This function sets lock on specified region registers or mask register.

Parameters

- base – PRINCE peripheral address.
- lock – registers to lock. This is a logical OR of members of the enumeration prince_lock_t

status_t PRINCE_GenNewIV(*prince_region_t* region, uint8_t *iv_code, bool store, *flash_config_t* *flash_context)

Generate new IV code.

This function generates new IV code and stores it into the persistent memory. Ensure about 800 bytes free space on the stack when calling this routine with the store parameter set to true!

Parameters

- region – PRINCE region index.

- iv_code – IV code pointer used for storing the newly generated 52 bytes long IV code.
- store – flag to allow storing the newly generated IV code into the persistent memory (FFR).
- flash_context – pointer to the flash driver context structure.

Returns
kStatus_Success upon success

Returns
kStatus_Fail otherwise, kStatus_Fail is also returned if the key code for the particular PRINCE region is not present in the keystore (though new IV code has been provided)

status_t PRINCE_LoadIV(*prince_region_t* region, uint8_t *iv_code)

Load IV code.

This function enables IV code loading into the PRINCE bus encryption engine.

Parameters

- region – PRINCE region index.
- iv_code – IV code pointer used for passing the IV code.

Returns
kStatus_Success upon success

Returns
kStatus_Fail otherwise

status_t PRINCE_SetEncryptForAddressRange(*prince_region_t* region, uint32_t start_address, uint32_t length, *flash_config_t* *flash_context, bool regenerate_iv)

Allow encryption/decryption for specified address range.

This function sets the encryption/decryption for specified address range. The SR mask value for the selected Prince region is calculated from provided start_address and length parameters. This calculated value is OR'ed with the actual SR mask value and stored into the PRINCE SR_ENABLE register and also into the persistent memory (FFR) to be used after the device reset. It is possible to define several nonadjacent encrypted areas within one Prince region when calling this function repeatedly. If the length parameter is set to 0, the SR mask value is set to 0 and thus the encryption/decryption for the whole selected Prince region is disabled. Ensure about 800 bytes free space on the stack when calling this routine!

Parameters

- region – PRINCE region index.
- start_address – start address of the area to be encrypted/decrypted.
- length – length of the area to be encrypted/decrypted.
- flash_context – pointer to the flash driver context structure.
- regenerate_iv – flag to allow IV code regenerating, storing into the persistent memory (FFR) and loading into the PRINCE engine

Returns
kStatus_Success upon success

Returns
kStatus_Fail otherwise

status_t PRINCE_GetRegionSREnable(PRINCE_Type *base, *prince_region_t* region, uint32_t *sr_enable)

Gets the PRINCE Sub-Region Enable register.

This function gets PRINCE SR_ENABLE register.

Parameters

- base – PRINCE peripheral address.
- region – PRINCE region index.
- sr_enable – Sub-Region Enable register pointer.

Returns
kStatus_Success upon success

Returns
kStatus_InvalidArgument

status_t PRINCE_GetRegionBaseAddress(PRINCE_Type *base, *prince_region_t* region, uint32_t *region_base_addr)

Gets the PRINCE region base address register.

This function gets PRINCE BASE_ADDR register.

Parameters

- base – PRINCE peripheral address.
- region – PRINCE region index.
- region_base_addr – Region base address pointer.

Returns
kStatus_Success upon success

Returns
kStatus_InvalidArgument

status_t PRINCE_SetRegionIV(PRINCE_Type *base, *prince_region_t* region, const uint8_t iv[8])

Sets the PRINCE region IV.

This function sets specified AES IV for the given region.

Parameters

- base – PRINCE peripheral address.
- region – Selection of the PRINCE region to be configured.
- iv – 64-bit AES IV in little-endian byte order.

status_t PRINCE_SetRegionBaseAddress(PRINCE_Type *base, *prince_region_t* region, uint32_t region_base_addr)

Sets the PRINCE region base address.

This function configures PRINCE region base address.

Parameters

- base – PRINCE peripheral address.
- region – Selection of the PRINCE region to be configured.
- region_base_addr – Base Address for region.

status_t PRINCE_SetRegionSREnable(PRINCE_Type *base, *prince_region_t* region, uint32_t sr_enable)

Sets the PRINCE Sub-Region Enable register.

This function configures PRINCE SR_ENABLE register.

Parameters

- base – PRINCE peripheral address.
- region – Selection of the PRINCE region to be configured.
- sr_enable – Sub-Region Enable register value.

status_t PRINCE_FlashEraseWithChecker(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length. It deals with the flash erase function complenentary to the standard erase API of the IAP1 driver. This implementation additionally checks if the whole encrypted PRINCE subregions are erased at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

Parameters

- config – The pointer to the flash driver context structure.
- start – The start address of the desired flash memory to be erased. The start address needs to be prince-sburegion-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be prince-sburegion-size-aligned.
- key – The value used to validate all flash erase APIs.

Returns
kStatus_FLASH_Success API was executed successfully.

Returns
kStatus_FLASH_InvalidArgument An invalid argument is provided.

Returns
kStatus_FLASH_AlignmentError The parameter is not aligned with the specified baseline.

Returns
kStatus_FLASH_AddressError The address is out of range.

Returns
kStatus_FLASH_EraseKeyError The API erase key is invalid.

Returns
kStatus_FLASH_CommandFailure Run-time error during the command execution.

Returns
kStatus_FLASH_CommandNotSupported Flash API is not supported.

Returns
kStatus_FLASH_EccError A correctable or uncorrectable error during command execution.

Returns
kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce Encrypted flash subregions are not erased at once.

status_t PRINCE_FlashProgramWithChecker(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length. It deals with the flash program function complenentary to the standard program API of the IAP1 driver. This implementation additionally checks if the whole PRINCE subregions are programmed at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

Parameters

- config – The pointer to the flash driver context structure.
- start – The start address of the desired flash memory to be programmed. Must be prince-sburegion-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be prince-sburegion-size-aligned.

Returns
kStatus_FLASH_Success API was executed successfully.

Returns
kStatus_FLASH_InvalidArgument An invalid argument is provided.

Returns
kStatus_FLASH_AlignmentError Parameter is not aligned with the specified baseline.

Returns
kStatus_FLASH_AddressError Address is out of range.

Returns
kStatus_FLASH_AccessError Invalid instruction codes and out-of bounds addresses.

Returns
kStatus_FLASH_CommandFailure Run-time error during the command execution.

Returns
kStatus_FLASH_CommandFailure Run-time error during the command execution.

Returns
kStatus_FLASH_CommandNotSupported Flash API is not supported.

Returns
kStatus_FLASH_EccError A correctable or uncorrectable error during command execution.

Returns
kStatus_FLASH_SizeError Encrypted flash subregions are not programmed at once.

FSL_PRINCE_DRIVER_SUBREGION_SIZE_IN_KB

FSL_PRINCE_DRIVER_MAX_FLASH_ADDR

ALIGN_DOWN(x, a)

2.91 PUF: Physical Unclonable Function

FSL_PUF_DRIVER_VERSION

PUF driver version. Version 2.2.0.

Current version: 2.2.0

Change log:

- 2.0.0
  - Initial version.
- 2.0.1
  - Fixed puf_wait_usec function optimization issue.
- 2.0.2
  - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
  - Fix MISRA C-2012 issue.
- 2.1.0
  - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
  - Update initizalition logic .
- 2.1.1
  - Fix ARMGCC build warning .
- 2.1.2
  - Update: Add automatic big to little endian swap for user (pre-shared) keys destinated to secret hardware bus (PUF key index 0).
- 2.1.3
  - Fix MISRA C-2012 issue.
- 2.1.4
  - Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.
- 2.1.5
  - Use common SDK delay in puf_wait_usec()
- 2.1.6
  - Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.
- 2.2.0
- Add support for kPUF_KeySlot4.
- Add new PUF_ClearKey() function, that clears a desired PUF internal HW key register.

enum _puf_key_index_register

Values:

enumerator kPUF_KeyIndex_00

enumerator kPUF_KeyIndex_01

enumerator kPUF_KeyIndex_02

enumerator kPUF_KeyIndex_03

enumerator kPUF_KeyIndex_04

enumerator kPUF_KeyIndex_05

enumerator kPUF_KeyIndex_06

enumerator kPUF_KeyIndex_07

enumerator kPUF_KeyIndex_08

enumerator kPUF_KeyIndex_09

enumerator kPUF_KeyIndex_10

enumerator kPUF_KeyIndex_11

enumerator kPUF_KeyIndex_12

enumerator kPUF_KeyIndex_13

enumerator kPUF_KeyIndex_14

enumerator kPUF_KeyIndex_15

enum __puf_min_max

Values:

enumerator kPUF_KeySizeMin

enumerator kPUF_KeySizeMax

enumerator kPUF_KeyIndexMax

enum __puf_key_slot

PUF key slot.

Values:

enumerator kPUF_KeySlot0

PUF key slot 0

enumerator kPUF_KeySlot1

PUF key slot 1

PUF status return codes.

Values:

enumerator kStatus_EnrollNotAllowed

enumerator kStatus_StartNotAllowed

typedef enum *_puf_key_index_register* puf_key_index_register_t

typedef enum *_puf_min_max* puf_min_max_t

typedef enum *_puf_key_slot* puf_key_slot_t

PUF key slot.

PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x)
Get Key Code size in bytes from key size in bytes at compile time.

PUF_MIN_KEY_CODE_SIZE

PUF_ACTIVATION_CODE_SIZE

KEYSTORE_PUF_DISCHARGE_TIME_FIRST_TRY_MS

KEYSTORE_PUF_DISCHARGE_TIME_MAX_MS

struct puf_config_t
#include <fsl_puf.h>

2.92 Puf_v3_driver

FSL_PUF_V3_DRIVER_VERSION
PUFv3 driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- 2.0.4
  - Fix warning
- 2.0.3
  - Update for various PUF CTRL wrapper
- 2.0.2
  - Fix MISRA issue in driver.
- 2.0.1
  - Fix PUF initialization issue and update driver to reflect SoC header changes.
- 2.0.0
  - Initial version.

Values:

enumerator kStatus_PUF_OperationNotAllowed

enumerator kStatus_PUF_AcNotForThisProductPhase1

enumerator kStatus_PUF_AcNotForThisProductPhase2

enumerator kStatus_PUF_AcCorruptedPhase1

enumerator kStatus_PUF_AcCorruptedPhase2

enumerator kStatus_PUF_AcAuthFailedPhase1

enumerator kStatus_PUF_NBOOT_AcAuthFailedPhase2

enumerator kStatus_PUF_QualityVerificationFail

enumerator kStatus_PUF_ContextIncorrect

enumerator kStatus_PUF_DestinationNotAllowed

enumerator kStatus_PUF_Failure

typedef uint32_t puf_endianness_t

typedef uint32_t puf_key_dest_t

typedef uint32_t puf_key_scope_t

typedef uint32_t puf_result_code_t

typedef uint32_t puf_sec_level_t

puf_endianness_t dataEndianness

uint8_t CKGATING

puf_key_scope_t keyScopeStarted

puf_key_scope_t keyScopeEnrolled

uint32_t userCtx0

uint32_t userCtx1

void PUF_GetDefaultConfig(*puf_config_t* *conf)

brief Sets the default configuration of PUF

This function initialize PUF config structure to default values.

Parameters

- conf – PUF configuration structure

status_t PUF_Init(PUF_Type *base, *puf_config_t* *conf)

brief Initialize PUF

This function enables power to PUF block and waits until the block initializes.

Parameters

- conf – PUF configuration structure

Returns

Status of the init operation

void PUF_Deinit(PUF_Type *base, *puf_config_t* *conf)

brief Denitialize PUF

This function disables power to PUF SRAM and peripheral clock.

Parameters

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

status_t PUF_Enroll(PUF_Type *base, uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)

brief Enroll PUF

This function derives a digital fingerprint, generates the corresponding Activation Code (AC) and returns it to be stored in an NVM or a file. This step needs to be performed only once for each device. This function may be permanently disallowed by a fuse.

Parameters

- base – PUF peripheral base address

- activationCode – **[out]** Word aligned address of the resulting activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns
Status of enroll operation.

status_t PUF_Start(PUF_Type *base, const uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)

brief Start PUF

The Activation Code generated during the Enroll operation is used to reconstruct the digital fingerprint. This needs to be done after every power-up and reset.

Parameters

- base – PUF peripheral base address
- activationCode – **[in]** Word aligned address of the input activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the start operation. return Status of start operation.

status_t PUF_Stop(PUF_Type *base)

brief Stop PUF

The Stop operation removes all key material from PUF flipflops and PUF SRAM, and sets PUF to the Stopped state.

Parameters

- base – PUF peripheral base address

Returns
Status of stop operation.

status_t PUF_GetKey(PUF_Type *base, *puf_key_ctx_t* *keyCtx, *puf_key_dest_t* keyDest, uint8_t *key, size_t keySize)

brief PUF Get Key

The Get Key operation derives a key from the intrinsic PUF key and externally provided context.

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keyDest – output destination of the derived PUF key
- key – **[out]** Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the derived key in bytes.

Returns
Status of get key operation.

status_t PUF_WrapGeneratedRandom(PUF_Type *base, *puf_key_ctx_t* *keyCtx, size_t keySize, uint8_t *keyCode, size_t keyCodeSize)

brief PUF Wrap generated random

The Wrap Generated Random operation wraps a random key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keySize – Size of the key to be generated in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns

Status of wrap generated random operation.

status_t PUF_Wrap(PUF_Type *base, *puf_key_ctx_t* *keyCtx, uint8_t *userKey, size_t userKeySize, uint8_t *keyCode, size_t keyCodeSize)

brief PUF Wrap user key

The Wrap operation wraps a user defined key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct.
- userKey – Word aligned address of input user key.
- userKeySize – Size of the key to be wrapped in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns

Status of wrap operation.

status_t PUF_Unwrap(PUF_Type *base, *puf_key_dest_t* keyDest, uint8_t *keyCode, size_t keyCodeSize, uint8_t *key, size_t keySize)

brief PUF Unwrap user key

The unwrap operation unwraps the key from a previously created Key Code (KC)

Parameters

- base – PUF peripheral base address
- keyDest – output destination of the unwraped PUF key
- keyCode – **[in]** Word aligned address of the input key code.
- keyCodeSize – Size of the input keycode in bytes.
- key – Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the key to be generated in bytes.

Returns

Status of unwrap operation.

status_t PUF_GenerateRandom(PUF_Type *base, uint8_t *data, size_t size)

brief Generate Random

The Generate Random operation outputs the requested amount of random data as specified in a provided context.

Parameters

- base – PUF peripheral base address
- size – Size of random data to be genarated in bytes.

Returns

Status of generate random operation.

status_t PUF_Zeroize(PUF_Type *base)

brief Zeroize PUF

This function clears all PUF internal logic and puts the PUF to zeroized state.

Parameters

- base – PUF peripheral base address

Returns

Status of the zeroize operation.

status_t PUF_Test(PUF_Type *base, uint8_t *score)

brief Test PUF

With the Test PUF operation, diagnostics about the PUF quality is collected and presented in a PUF score.

Parameters

- base – PUF peripheral base address
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns

Status of the test operation.

static inline void PUF_BlockCommand(PUF_Type *base, uint32_t mask)

Blocks specified PUF commands.

This function blocks PUF commands specified by mask parameter.

Parameters

- base – PUF peripheral base address
- mask – Mask of parameters which should be blocked until power-cycle.

Returns

Status of the test operation.

status_t PUF_SetLock(PUF_Type *base, *puf_sec_level_t* securityLevel)

brief Set lock of PUF operation

Lock the security level of PUF block until key generate, wrap or unwrap operation is completed. Note: Only security level defined in SEC_LOCK register can use PUFv3 or change its security level. Default setting after leaving ROM is Secure-Privilege

Parameters

- base – PUF peripheral base address
- securityLevel – Security level of PUF block.

Returns
Status of the test operation.

status_t PUF_SetCtxMask(PUF_Type *base, uint32_t appCtxMask)

brief Set App Context mask

This function sets Application defined context mask used in conjunction with key user context 2. Whenever bit in this register is 1, corresponding bit in user context 2 provided during key code creation should be zero only.

This register is only modifiable by task running at secure-privilege level.

Parameters

- base – PUF peripheral base address
- appCtxMask – Value of the Application defined context mask.

Returns
Status of the test operation.

kPUF_EndianLittle

kPUF_EndianBig

kPUF_KeyDestRegister

kPUF_KeyDestKeyBus

kPUF_KeyDestInvalid

kPUF_KeyAllowRegister

kPUF_KeyAllowKeyBus

kPUF_KeyAllowAll

kPUF_ResultOK

kPUF_AcNotForThisProductPhase1

kPUF_AcNotForThisProductPhase2

kPUF_AcCorruptedPhase1

kPUF_AcCorruptedPhase2

kPUF_AcAuthFailedPhase1

kPUF_AcAuthFailedPhase2

kPUF_QualityVerificationFail

kPUF_ContextIncorrect

kPUF_DestinationNotAllowed

kPUF_Failure

kPUF_NonsecureUser

kPUF_NonsecurePrivilege

kPUF_SecureUser

kPUF_SecurePrivilege

PUF_ACTIVATION_CODE_SIZE

PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x)

SEC_LOCK_PATTERN

struct puf_config_t

#include <fsl_puf.h>

struct puf_key_ctx_t

#include <fsl_puf_v3.h>

2.93 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

2.94 QDC: Quadrature Encoder/Decoder

void QDC_Init(QDC_Type *base, const *qdc_config_t* *config)

Initialization for the QDC module.

This function is to make the initialization for the QDC module. It should be called firstly before any operation to the QDC with the operations like:

- Enable the clock for QDC module.
- Configure the QDC's working attributes.

Parameters

- base – QDC peripheral base address.
- config – Pointer to configuration structure. See to "qdc_config_t".

void QDC_Deinit(QDC_Type *base)

De-initialization for the QDC module.

This function is to make the de-initialization for the QDC module. It could be called when QDC is no longer used with the operations like:

- Disable the clock for QDC module.

Parameters

- base – QDC peripheral base address.

void QDC_GetDefaultConfig(*qdc_config_t* *config)

Get an available pre-defined settings for QDC's configuration.

This function initializes the QDC configuration structure with an available settings, the default value are:

```
config->enableReverseDirection                = false;
config->decoderWorkMode                       = kQDC_DecoderWorkAsNormalMode;
config->HOMETriggerMode                       = kQDC_HOMETriggerDisabled;
config->INDEXTriggerMode                      = kQDC_INDEXTriggerDisabled;
config->enableTRIGGERClearPositionCounter     = false;
config->enableTRIGGERClearHoldPositionCounter = false;
config->enableWatchdog                        = false;
config->watchdogTimeoutValue                  = 0U;
config->filterCount                           = 0U;
config->filterSamplePeriod                    = 0U;
config->positionMatchMode                     = kQDC_
↪POSMATCHOnPositionCounterEqualToComapreValue;
config->positionCompareValue                  = 0xFFFFFFFFU;
config->revolutionCountCondition              = kQDC_RevolutionCountOnINDEXPulse;
config->enableModuloCountMode                 = false;
config->positionModulusValue                  = 0U;
config->positionInitialValue                  = 0U;
config->prescalerValue                        = kQDC_ClockDiv1;
config->enablePeriodMeasurementFunction       = true;
```

Parameters

- config – Pointer to a variable of configuration structure. See to "qdc_config_t".

void QDC_DoSoftwareLoadInitialPositionValue(QDC_Type *base)

Load the initial position value to position counter.

This function is to transfer 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 – QDC peripheral base address.

void QDC_SetSelfTestConfig(QDC_Type *base, const *qdc_self_test_config_t* *config)

Enable and configure the self test function.

This function is to enable and configuration the self test function. It controls and sets the frequency of a quadrature signal generator. It provides a quadrature test signal to the inputs of the quadrature decoder module. It is a factory test feature; however, it may be useful to customers' software development and testing.

Parameters

- base – QDC peripheral base address.
- config – Pointer to configuration structure. See to "qdc_self_test_config_t". Pass "NULL" to disable.

void QDC_EnableWatchdog(QDC_Type *base, bool enable)

Enable watchdog for QDC module.

Parameters

- base – QDC peripheral base address
- enable – Enables or disables the watchdog

void QDC_SetInitialPositionValue(QDC_Type *base, uint32_t value)

Set initial position value for QDC module.

Parameters

- base – QDC peripheral base address
- value – Positive initial value

uint32_t QDC_GetStatusFlags(QDC_Type *base)

Get the status flags.

Parameters

- base – QDC peripheral base address.

Returns

Mask value of status flags. For available mask, see to "_qdc_status_flags".

void QDC_ClearStatusFlags(QDC_Type *base, uint32_t mask)

Clear the status flags.

Parameters

- base – QDC peripheral base address.
- mask – Mask value of status flags to be cleared. For available mask, see to "_qdc_status_flags".

static inline uint16_t QDC_GetSignalStatusFlags(QDC_Type *base)

Get the signals' real-time status.

Parameters

- base – QDC peripheral base address.

Returns

Mask value of signals' real-time status. For available mask, see to "_qdc_signal_status_flags"

void QDC_EnableInterrupts(QDC_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – QDC peripheral base address.
- mask – Mask value of interrupts to be enabled. For available mask, see to "_qdc_interrupt_enable".

void QDC_DisableInterrupts(QDC_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – QDC peripheral base address.
- mask – Mask value of interrupts to be disabled. For available mask, see to "_qdc_interrupt_enable".

uint32_t QDC_GetEnabledInterrupts(QDC_Type *base)

Get the enabled interrupts' flags.

Parameters

- base – QDC peripheral base address.

Returns

Mask value of enabled interrupts.

uint32_t QDC_GetPositionValue(QDC_Type *base)

Get the current position counter's value.

Parameters

- base – QDC peripheral base address.

Returns

Current position counter's value.

uint32_t QDC_GetHoldPositionValue(QDC_Type *base)

Get the hold position counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- base – QDC peripheral base address.

Returns

Hold position counter's value.

static inline uint16_t QDC_GetPositionDifferenceValue(QDC_Type *base)

Get the position difference counter's value.

Parameters

- base – QDC peripheral base address.

Returns
The position difference counter's value.

static inline uint16_t QDC_GetHoldPositionDifferenceValue(QDC_Type *base)

Get the hold position difference counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- base – QDC peripheral base address.

Returns
Hold position difference counter's value.

static inline uint16_t QDC_GetRevolutionValue(QDC_Type *base)

Get the position revolution counter's value.

Parameters

- base – QDC peripheral base address.

Returns
The position revolution counter's value.

static inline uint16_t QDC_GetHoldRevolutionValue(QDC_Type *base)

Get the hold position revolution counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- base – QDC peripheral base address.

Returns
Hold position revolution counter's value.

static inline uint16_t QDC_GetLastEdgeTimeValue(QDC_Type *base)

Get the last edge time value.

Parameters

- base – QDC peripheral base address.

Returns
The last edge time hold value.

static inline uint16_t QDC_GetHoldLastEdgeTimeValue(QDC_Type *base)

Get the last edge time hold value.

Parameters

- base – QDC peripheral base address.

Returns
The last edge time hold value.

static inline uint16_t QDC_GetPositionDifferencePeriodValue(QDC_Type *base)

Get the position difference period value.

Parameters

- base – QDC peripheral base address.

Returns
The position difference period hold value.

static inline uint16_t QDC_GetPositionDifferencePeriodBufferValue(QDC_Type *base)

Get the position difference period buffer value.

Parameters

- base – QDC peripheral base address.

Returns
The position difference period hold value.

static inline uint16_t QDC_GetHoldPositionDifferencePeriodValue(QDC_Type *base)

Get the position difference period hold value.

Parameters

- base – QDC peripheral base address.

Returns
The position difference period hold value.

enum _qdc_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kQDC_HOMETransitionInterruptEnable
HOME interrupt enable.

enumerator kQDC_INDEXPulseInterruptEnable
INDEX pulse interrupt enable.

enumerator kQDC_WatchdogTimeoutInterruptEnable
Watchdog timeout interrupt enable.

enumerator kQDC_PositionCompareInerruptEnable
Position compare interrupt enable.

enumerator kQDC_PositionRollOverInterruptEnable
Roll-over interrupt enable.

enumerator kQDC_PositionRollUnderInterruptEnable
Roll-under interrupt enable.

enum _qdc_status_flags

Status flag mask.

These flags indicate the counter's events.

Values:

enumerator kQDC_HOMETransitionFlag
HOME signal transition interrupt request.

enumerator kQDC_INDEXPulseFlag
INDEX Pulse Interrupt Request.

enumerator kQDC_WatchdogTimeoutFlag
Watchdog timeout interrupt request.

enumerator kQDC_PositionCompareFlag
Position compare interrupt request.

enumerator kQDC_PositionRollOverFlag
Roll-over interrupt request.

enumerator kQDC_PositionRollUnderFlag
Roll-under interrupt request.

enumerator kQDC_LastCountDirectionFlag
Last count was in the up direction, or the down direction.

enum _qdc_signal_status_flags
Signal status flag mask.

These flags indicate the counter's signal.

Values:

enumerator kQDC_RawHOMEStatusFlag
Raw HOME input.

enumerator kQDC_RawINDEXStatusFlag
Raw INDEX input.

enumerator kQDC_RawPHBStatusFlag
Raw PHASEB input.

enumerator kQDC_RawPHAEXStatusFlag
Raw PHASEA input.

enumerator kQDC_FilteredHOMEStatusFlag
The filtered version of HOME input.

enumerator kQDC_FilteredINDEXStatusFlag
The filtered version of INDEX input.

enumerator kQDC_FilteredPHBStatusFlag
The filtered version of PHASEB input.

enumerator kQDC_FilteredPHAStatusFlag
The filtered version of PHASEA input.

enum _qdc_home_trigger_mode
Define HOME signal's trigger mode.

The QDC would count the trigger from HOME signal line.

Values:

enumerator kQDC_HOMETriggerDisabled
HOME signal's trigger is disabled.

enumerator kQDC_HOMETriggerOnRisingEdge
Use positive going edge-to-trigger initialization of position counters.

enumerator kQDC_HOMETriggerOnFallingEdge
Use negative going edge-to-trigger initialization of position counters.

enum _qdc_index_trigger_mode
Define INDEX signal's trigger mode.

The QDC would count the trigger from INDEX signal line.

Values:

enumerator kQDC_INDEXTriggerDisabled
INDEX signal's trigger is disabled.

enumerator kQDC_INDEXTriggerOnRisingEdge
Use positive going edge-to-trigger initialization of position counters.

enumerator kQDC_INDEXTriggerOnFallingEdge
Use negative going edge-to-trigger initialization of position counters.

enum _qdc_decoder_work_mode
Define type for decoder work mode.

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

Values:

enumerator kQDC_DecoderWorkAsNormalMode
Use standard quadrature decoder with PHASEA and PHASEB.

enumerator kQDC_DecoderWorkAsSignalPhaseCountMode
PHASEA input generates a count signal while PHASEB input control the direction.

enum _qdc_position_match_mode
Define type for the condition of POSMATCH pulses.

Values:

enumerator kQDC_POSMATCHOnPositionCounterEqualToComapreValue
POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

enumerator kQDC_POSMATCHOnReadingAnyPositionCounter
POSMATCH pulses when any position counter register is read.

enum _qdc_revolution_count_condition
Define type for determining how the revolution counter (REV) is incremented/decremented.

Values:

enumerator kQDC_RevolutionCountOnINDEXPulse
Use INDEX pulse to increment/decrement revolution counter.

enumerator kQDC_RevolutionCountOnRollOverModulus
Use modulus counting roll-over/under to increment/decrement revolution counter.

enum _qdc_self_test_direction
Define type for direction of self test generated signal.

Values:

enumerator kQDC_SelfTestDirectionPositive
Self test generates the signal in positive direction.

enumerator kQDC_SelfTestDirectionNegative
Self test generates the signal in negative direction.

enum _qdc_prescaler
Define prescaler value for clock in CTRL3.

The clock is prescaled by a value of 2^PRSC which means that the prescaler logic can divide the clock by a minimum of 1 and a maximum of 32,768.

Values:

enumerator kQDC_ClockDiv1

enumerator kQDC_ClockDiv2

enumerator kQDC_ClockDiv4

enumerator kQDC_ClockDiv8

enumerator kQDC_ClockDiv16

enumerator kQDC_ClockDiv32

enumerator kQDC_ClockDiv64

enumerator kQDC_ClockDiv128

enumerator kQDC_ClockDiv256

enumerator kQDC_ClockDiv512

enumerator kQDC_ClockDiv1024

enumerator kQDC_ClockDiv2048

enumerator kQDC_ClockDiv4096

enumerator kQDC_ClockDiv8192

enumerator kQDC_ClockDiv16384

enumerator kQDC_ClockDiv32768

enum __qdc_filter_prescaler

Define input filter prescaler value.

The input filter prescaler value is to prescale the IPBus clock. (Frequency of FILT clock) = (Frequency of IPBus clock) / 2^FILT_PRSC.

Values:

enumerator kQDC_FilterPrescalerDiv1

Input filter prescaler is 1.

enumerator kQDC_FilterPrescalerDiv2

Input filter prescaler is 2.

enumerator kQDC_FilterPrescalerDiv4

Input filter prescaler is 4.

enumerator kQDC_FilterPrescalerDiv8

Input filter prescaler is 8.

enumerator kQDC_FilterPrescalerDiv16

Input filter prescaler is 16.

enumerator kQDC_FilterPrescalerDiv32

Input filter prescaler is 32.

enumerator kQDC_FilterPrescalerDiv64

Input filter prescaler is 64.

enumerator kQDC_FilterPrescalerDiv128

Input filter prescaler is 128.

typedef enum *_qdc_home_trigger_mode* qdc_home_trigger_mode_t

Define HOME signal's trigger mode.

The QDC would count the trigger from HOME signal line.

typedef enum *_qdc_index_trigger_mode* qdc_index_trigger_mode_t

Define INDEX signal's trigger mode.

The QDC would count the trigger from INDEX signal line.

typedef enum *_qdc_decoder_work_mode* qdc_decoder_work_mode_t

Define type for decoder work mode.

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

typedef enum *_qdc_position_match_mode* qdc_position_match_mode_t

Define type for the condition of POSMATCH pulses.

typedef enum *_qdc_revolution_count_condition* qdc_revolution_count_condition_t

Define type for determining how the revolution counter (REV) is incremented/decremented.

typedef enum *_qdc_self_test_direction* qdc_self_test_direction_t

Define type for direction of self test generated signal.

typedef enum *_qdc_prescaler* qdc_prescaler_t

Define prescaler value for clock in CTRL3.

The clock is prescaled by a value of 2^PRSC which means that the prescaler logic can divide the clock by a minimum of 1 and a maximum of 32,768.

typedef enum *_qdc_filter_prescaler* qdc_filter_prescaler_t

Define input filter prescaler value.

The input filter prescaler value is to prescale the IPBus clock. (Frequency of FILT clock) = (Frequency of IPBus clock) / 2^FILT_PRSC.

typedef struct *_qdc_config* qdc_config_t

Define user configuration structure for QDC module.

typedef struct *_qdc_self_test_config* qdc_self_test_config_t

Define configuration structure for self test module.

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

FSL_QDC_DRIVER_VERSION

struct _qdc_config

#include <fsl_qdc.h> Define user configuration structure for QDC module.

Public Members

bool enableReverseDirection

Enable reverse direction counting.

qdc_decoder_work_mode_t decoderWorkMode

Enable signal phase count mode.

qdc_home_trigger_mode_t HOMETriggerMode

Enable HOME to initialize position counters.

qdc_index_trigger_mode_t INDEXTriggerMode

Enable INDEX to initialize position counters.

bool enableTRIGGERClearPositionCounter

Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.

bool enableTRIGGERClearHoldPositionCounter

Enable update of hold registers on rising edge of TRIGGER, or not.

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. This field is only available when "enableWatchdog" = true. The available value is a 16-bit unsigned number.

qdc_filter_prescaler_t filterPrescaler

Input filter prescaler.

uint16_t filterCount

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. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. The Available range is 0 - 7.

uint16_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.

qdc_position_match_mode_t positionMatchMode

The condition of POSMATCH pulses.

uint32_t positionCompareValue

Position compare value. The available value is a 32-bit number.

qdc_revolution_count_condition_t revolutionCountCondition

Revolution Counter Modulus Enable.

bool enableModuloCountMode

Enable Modulo Counting.

uint32_t positionModulusValue

Position modulus value. This value would be available only when "enableModulo-CountMode" = true. The available value is a 32-bit number.

uint32_t positionInitialValue

Position initial value. The available value is a 32-bit number.

bool enablePeriodMeasurementFunction

Enable period measurement function.

qdc_prescaler_t prescalerValue

The value of prescaler.

struct _qdc_self_test_config

#include <fsl_qdc.h> Define configuration structure for self test module.

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

Public Members

qdc_self_test_direction_t signalDirection

Direction of self test generated signal.

uint16_t signalCount

Hold the number of quadrature advances to generate. The available range is 0 - 255.

uint16_t signalPeriod

Hold the period of quadrature phase in IPBus clock cycles. The available range is 0 - 31.

2.95 Reset Driver

enum _SYSCON_RSTn

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

Values:

enumerator kRST_None

None RESET gate.

enumerator kFMU_RST_SHIFT_RSTn

Flash management unit reset control

enumerator kFLEXSPI_RST_SHIFT_RSTn

FLEXSPI reset control

enumerator kMUX_RST_SHIFT_RSTn

Input mux reset control

enumerator kPORT0_RST_SHIFT_RSTn

PORT0 reset control

enumerator kPORT1_RST_SHIFT_RSTn

PORT1 reset control

enumerator kPORT2_RST_SHIFT_RSTn

PORT2 reset control

enumerator kPORT3_RST_SHIFT_RSTn

PORT3 reset control

enumerator kPORT4_RST_SHIFT_RSTn

PORT4 reset control

enumerator kGPIO0_RST_SHIFT_RSTn

GPIO0 reset control

enumerator kGPIO1_RST_SHIFT_RSTn
GPIO1 reset control

enumerator kGPIO2_RST_SHIFT_RSTn
GPIO2 reset control

enumerator kGPIO3_RST_SHIFT_RSTn
GPIO3 reset control

enumerator kGPIO4_RST_SHIFT_RSTn
GPIO4 reset control

enumerator kPINT_RST_SHIFT_RSTn
Pin interrupt (PINT) reset control

enumerator kDMA0_RST_SHIFT_RSTn
DMA0 reset control

enumerator kCRC_RST_SHIFT_RSTn
CRC reset control

enumerator kMAILBOX_RST_SHIFT_RSTn
Mailbox reset control

enumerator kMRT_RST_SHIFT_RSTn
Multi-rate timer (MRT) reset control

enumerator kOSTIMER_RST_SHIFT_RSTn
OSTimer reset control

enumerator kSCT_RST_SHIFT_RSTn
SCTimer/PWM(SCT) reset control

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control

enumerator kADC1_RST_SHIFT_RSTn
ADC1 reset control

enumerator kDAC0_RST_SHIFT_RSTn
DAC0 reset control

enumerator kEVSIM0_RST_SHIFT_RSTn
EVSIM0 reset control

enumerator kEVSIM1_RST_SHIFT_RSTn
EVSIM1 reset control

enumerator kUTICK_RST_SHIFT_RSTn
Micro-tick timer reset control

enumerator kFC0_RST_SHIFT_RSTn
Flexcomm Interface 0 reset control

enumerator kFC1_RST_SHIFT_RSTn
Flexcomm Interface 1 reset control

enumerator kFC2_RST_SHIFT_RSTn
Flexcomm Interface 2 reset control

enumerator kFC3_RST_SHIFT_RSTn
Flexcomm Interface 3 reset control

enumerator kFC4_RST_SHIFT_RSTn
Flexcomm Interface 4 reset control

enumerator kFC5_RST_SHIFT_RSTn
Flexcomm Interface 5 reset control

enumerator kFC6_RST_SHIFT_RSTn
Flexcomm Interface 6 reset control

enumerator kFC7_RST_SHIFT_RSTn
Flexcomm Interface 7 reset control

enumerator kFC8_RST_SHIFT_RSTn
Flexcomm Interface 8 reset control

enumerator kFC9_RST_SHIFT_RSTn
MICFIL reset control

enumerator kMICFIL_RST_SHIFT_RSTn
Flexcomm Interface 7 reset control

enumerator kCTIMER2_RST_SHIFT_RSTn
CTimer 2 reset control

enumerator kUSB0_RAM_RST_SHIFT_RSTn
USB0 RAM reset control

enumerator kUSB0_FS_DCD_RST_SHIFT_RSTn
USB0-FS DCD reset control

enumerator kUSB0_FS_RST_SHIFT_RSTn
USB0-FS reset control

enumerator kCTIMER0_RST_SHIFT_RSTn
CTimer 0 reset control

enumerator kCTIMER1_RST_SHIFT_RSTn
CTimer 1 reset control

enumerator kSMART_DMA_RST_SHIFT_RSTn
SmartDMA reset control

enumerator kDMA1_RST_SHIFT_RSTn
DMA1 reset control

enumerator kENET_RST_SHIFT_RSTn
Ethernet reset control

enumerator kUSDHC_RST_SHIFT_RSTn
uSDHC reset control

enumerator kFLEXIO_RST_SHIFT_RSTn
FLEXIO reset control

enumerator kSAI0_RST_SHIFT_RSTn
SAI0 reset control

enumerator kSAI1_RST_SHIFT_RSTn
SAI1 reset control

enumerator kTRO_RST_SHIFT_RSTn
TRO reset control

enumerator kFREQME_RST_SHIFT_RSTn
FREQME reset control

enumerator kTRNG_RST_SHIFT_RSTn
TRNG reset control

enumerator kFLEXCAN0_RST_SHIFT_RSTn
Flexcan0 reset control

enumerator kFLEXCAN1_RST_SHIFT_RSTn
Flexcan1 reset control

enumerator kUSB_HS_RST_SHIFT_RSTn
USB HS reset control

enumerator kUSB_HS_PHY_RST_SHIFT_RSTn
USB HS PHY reset control

enumerator kPOWERQUAD_RST_SHIFT_RSTn
PowerQuad reset control

enumerator kPLU_RST_SHIFT_RSTn
PLU reset control

enumerator kCTIMER3_RST_SHIFT_RSTn
CTimer 3 reset control

enumerator kCTIMER4_RST_SHIFT_RSTn
CTimer 4 reset control

enumerator kPUF_RST_SHIFT_RSTn
PUF reset control

enumerator kPKC_RST_SHIFT_RSTn
PKC reset control

enumerator kSM3_RST_SHIFT_RSTn
SM3 reset control

enumerator kI3C0_RST_SHIFT_RSTn
I3C0 reset control

enumerator kI3C1_RST_SHIFT_RSTn
I3C1 reset control

enumerator kSINC_RST_SHIFT_RSTn
SINC reset control

enumerator kCOOLFLUX_RST_SHIFT_RSTn
CoolFlux reset control

enumerator kQDC0_RST_SHIFT_RSTn
QDC0 reset control

enumerator kQDC1_RST_SHIFT_RSTn
QDC1 reset control

enumerator kPWM0_RST_SHIFT_RSTn
PWM0 reset control

enumerator kPWM1_RST_SHIFT_RSTn
PWM1 reset control

enumerator kAOI0_RST_SHIFT_RSTn
AOI0 reset control

enumerator kDAC1_RST_SHIFT_RSTn
DAC1 reset control

enumerator kDAC2_RST_SHIFT_RSTn
DAC2 reset control

enumerator kOPAMP0_RST_SHIFT_RSTn
OPAMP0 reset control

enumerator kOPAMP1_RST_SHIFT_RSTn
OPAMP1 reset control

enumerator kOPAMP2_RST_SHIFT_RSTn
OPAMP2 reset control

enumerator kCMP2_RST_SHIFT_RSTn
CMP2 reset control

enumerator kVREF_RST_SHIFT_RSTn
VREF reset control

enumerator kCOOLFLUX_APB_RST_SHIFT_RSTn
CoolFlux APB reset control

enumerator kNEUTRON_RST_SHIFT_RSTn
Neutron mini reset control

enumerator kTSI_RST_SHIFT_RSTn
TSI reset control

enumerator kEWM_RST_SHIFT_RSTn
EWM reset control

enumerator kEIM_RST_SHIFT_RSTn
EIM reset control

enumerator kSEMA42_RST_SHIFT_RSTn
Semaphore reset control

typedef enum *_SYSCON_RSTn* SYSCON_RSTn_t
Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

typedef *SYSCON_RSTn_t* reset_ip_name_t

void RESET_SetPeripheralReset(*reset_ip_name_t* peripheral)
Assert reset to peripheral.

Asserts reset signal to specified peripheral module.

Parameters

- peripheral – Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_ClearPeripheralReset(*reset_ip_name_t* peripheral)
Clear reset to peripheral.

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

- peripheral – Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_PeripheralReset(*reset_ip_name_t* peripheral)

Reset peripheral module.

Reset peripheral module.

Parameters

- peripheral – Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void RESET_ReleasePeripheralReset(*reset_ip_name_t* peripheral)

Release peripheral module.

Release peripheral module.

Parameters

- peripheral – Peripheral to release. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void CLOCK_SetCpu1Reset(SYSCON_Type *base, bool enable)

Set/clear the CPU1 reset signal.

Parameters

- enable – True to set the reset signal, false to clear the reset signal.

FSL_RESET_DRIVER_VERSION

reset driver version 2.4.1

ADC_RSTS

Array initializers with peripheral reset bits

CRC_RSTS

CTIMER_RSTS

DMA_RSTS_N

LP_FLEXCOMM_RSTS

GPIO_RSTS_N

INPUTMUX_RSTS

FLASH_RSTS

MRT_RSTS

PINT_RSTS

TRNG_RSTS

SCT_RSTS

UTICK_RSTS

PLU_RSTS_N

OSTIMER_RSTS

POWERQUAD_RSTS

I3C_RSTS

2.96 RNG: Random Number Generator

FSL_RNG_DRIVER_VERSION

RNG driver version. Version 2.0.3.

Current version: 2.0.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Fix MISRA C-2012 issue.
- Version 2.0.2
  - Add RESET_PeripheralReset function inside RNG_Init and RNG_Deinit functions.
- Version 2.0.3
  - Modified RNG_Init and RNG_GetRandomData functions, added rng_accumulateEntropy and rng_readEntropy functions.
  - These changes are reflecting recommended usage of RNG according to device UM.

void RNG_Init(RNG_Type *base)

Initializes the RNG.

This function initializes the RNG. When called, the RNG module and ring oscillator is enabled.

Parameters

- base – RNG base address

Returns

If successful, returns the kStatus_RNG_Success. Otherwise, it returns an error.

void RNG_Deinit(RNG_Type *base)

Shuts down the RNG.

This function shuts down the RNG.

Parameters

- base – RNG base address.

status_t RNG_GetRandomData(RNG_Type *base, void *data, size_t dataSize)

Gets random data.

This function gets random data from the RNG.

Parameters

- base – RNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

Returns

random data

static inline uint32_t RNG_GetRandomWord(RNG_Type *base)

Returns random 32-bit number.

This function gets random number from the RNG.

Parameters

- base – RNG base address.

Returns
random number

2.97 RTC: Real Time Clock

void RTC_Init(RTC_Type *base, const *rtc_config_t* *config)

Ungates the RTC clock and configures the peripheral for basic operation.

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

Parameters

- base – RTC peripheral base address
- config – Pointer to the user's RTC configuration structure.

void RTC_Deinit(RTC_Type *base)

Stops the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

void RTC_GetDefaultConfig(*rtc_config_t* *config)

Fills in the RTC config struct with the default settings.

The default values are as follows.

```
config->clockSource = kRTC_BusClock;
config->prescaler = kRTC_ClockDivide_16_2048;
config->time_us = 1000000U;
```

Parameters

- config – Pointer to the user's RTC configuration structure.

status_t RTC_SetDatetime(*rtc_datetime_t* *datetime)

Sets the RTC date and time according to the given time structure.

Parameters

- datetime – Pointer to the structure where the date and time details are stored.

Returns
kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void RTC_GetDatetime(*rtc_datetime_t* *datetime)

Gets the RTC time and stores it in the given time structure.

Parameters

- datetime – Pointer to the structure where the date and time details are stored.

void RTC_SetAlarm(uint32_t second)

Sets the RTC alarm time.

Parameters

- second – Second value. User input the number of second. After seconds user input, alarm occurs.

void RTC_GetAlarm(*rtc_datetime_t* *datetime)

Returns the RTC alarm time.

Parameters

- datetime – Pointer to the structure where the alarm date and time details are stored.

void RTC_SetAlarmCallback(*rtc_alarm_callback_t* callback)

Set the RTC alarm callback.

Parameters

- callback – The callback function.

static inline void RTC_SelectSourceClock(RTC_Type *base, *rtc_clock_source_t* clock, *rtc_clock_prescaler_t* divide)

Select Real-Time Clock Source and Clock Prescaler.

Parameters

- base – RTC peripheral base address
- clock – Select RTC clock source
- divide – Select RTC clock prescaler value

uint32_t RTC_GetDivideValue(RTC_Type *base)

Get the RTC Divide value.

Note: This API should be called after selecting clock source and clock prescaler.

Parameters

- base – RTC peripheral base address

Returns

The Divider value. The Divider value depends on clock source and clock prescaler

static inline void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected RTC interrupts.

Parameters

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

static inline void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.

Parameters

- base – PIT peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration rtc_interrupt_enable_t

static inline uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled RTC interrupts.

Parameters

- base – RTC peripheral base address

Returns

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

static inline uint32_t RTC_GetInterruptFlags(RTC_Type *base)

Gets the RTC interrupt flags.

Parameters

- base – RTC peripheral base address

Returns

The interrupt flags. This is the logical OR of members of the enumeration rtc_interrupt_flags_t

static inline void RTC_ClearInterruptFlags(RTC_Type *base, uint32_t mask)

Clears the RTC interrupt flags.

Parameters

- base – RTC peripheral base address
- mask – The interrupt flags to clear. This is a logical OR of members of the enumeration rtc_interrupt_flags_t

static inline void RTC_EnableOutput(RTC_Type *base, uint32_t mask)

Enable the RTC output. If RTC output is enabled, the RTCO pinout will be toggled when RTC counter overflows.

Parameters

- base – RTC peripheral base address
- mask – The Output to enable. This is a logical OR of members of the enumeration rtc_output_enable_t

static inline void RTC_DisableOutput(RTC_Type *base, uint32_t mask)

Disable the RTC output.

Parameters

- base – RTC peripheral base address
- mask – The Output to disable. This is a logical OR of members of the enumeration rtc_output_enable_t

static inline void RTC_SetModuloValue(RTC_Type *base, uint32_t value)

Set the RTC module value.

Parameters

- base – RTC peripheral base address
- value – The Module Value. The RTC Modulo register allows the compare value to be set to any value from 0x0000 to 0xFFFF

static inline uint16_t RTC_GetCountValue(RTC_Type *base)

Get the RTC Count value.

Parameters

- base – RTC peripheral base address

Returns
The Count Value. The Count Value is allowed from 0x0000 to 0xFFFF

FSL_RTC_DRIVER_VERSION
Version 2.0.6

enum _rtc_clock_source
List of RTC clock source.

Values:

enumerator kRTC_ExternalClock
External clock source

enumerator kRTC_LPOCLK
Real-time clock source is 1 kHz (LPOCLK)

enumerator kRTC_ICSIRCLK
Internal reference clock (ICSIRCLK)

enumerator kRTC_BusClock
Bus clock

enum _rtc_clock_prescaler
List of RTC clock prescaler.

Values:

enumerator kRTC_ClockDivide_off
Off

enumerator kRTC_ClockDivide_1_128
If RTCLKS = x0, it is 1; if RTCLKS = x1, it is 128

enumerator kRTC_ClockDivide_2_256
If RTCLKS = x0, it is 2; if RTCLKS = x1, it is 256

enumerator kRTC_ClockDivide_4_512
If RTCLKS = x0, it is 4; if RTCLKS = x1, it is 512

enumerator kRTC_ClockDivide_8_1024
If RTCLKS = x0, it is 8; if RTCLKS = x1, it is 1024

enumerator kRTC_ClockDivide_16_2048
If RTCLKS = x0, it is 16; if RTCLKS = x1, it is 2048

enumerator kRTC_ClockDivide_32_100
If RTCLKS = x0, it is 32; if RTCLKS = x1, it is 100

enumerator kRTC_ClockDivide_64_1000
If RTCLKS = x0, it is 64; if RTCLKS = x1, it is 1000

enum _rtc_interrupt_enable
List of RTC interrupts.

Values:

enumerator kRTC_InterruptEnable
Interrupt enable

enum _RTC_interrupt_flags
List of RTC Interrupt flags.

Values:

enumerator kRTC_InterruptFlag
Interrupt flag

enum _RTC_output_enable
List of RTC Output.

Values:

enumerator kRTC_OutputEnable
Output enable

typedef struct *_rtc_datetime* rtc_datetime_t
Structure is used to hold the date and time.

typedef enum *_rtc_clock_source* rtc_clock_source_t
List of RTC clock source.

typedef enum *_rtc_clock_prescaler* rtc_clock_prescaler_t
List of RTC clock prescaler.

typedef enum *_rtc_interrupt_enable* rtc_interrupt_enable_t
List of RTC interrupts.

typedef enum *_RTC_interrupt_flags* rtc_interrupt_flags_t
List of RTC Interrupt flags.

typedef enum *_RTC_output_enable* rtc_output_enable_t
List of RTC Output.

typedef struct *_rtc_config* rtc_config_t
RTC config structure.

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

typedef void (*rtc_alarm_callback_t)(void)
RTC alarm callback function.

struct _rtc_datetime
#include <fsl_rtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year
Range from 1970 to 2099.

uint8_t month
Range from 1 to 12.

uint8_t day
Range from 1 to 31 (depending on month).

uint8_t hour
Range from 0 to 23.

uint8_t minute
Range from 0 to 59.

uint8_t second
Range from 0 to 59.

struct _rtc_config

#include <fsl_rtc.h> RTC config structure.

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

2.98 Runbootloader

void bootloader_user_entry(void *arg)

Run the Bootloader API to force into the ISP mode base on the user arg.

Parameters

- arg – Indicates API prototype fields definition. Refer to the above user_app_boot_invoke_option_t structure

FSL_ROMAPI_RUNBOOTLOADER_DRIVER_VERSION

ROMAPI_RUNBOOTLOADER driver version 2.0.0.

uint32_t reserved

uint32_t boot_image_index

uint32_t instance

uint32_t boot_interface

uint32_t mode

uint32_t tag

struct *user_app_boot_invoke_option_t* B

uint32_t U

union *user_app_boot_invoke_option_t* option

struct user_app_boot_invoke_option_t

#include <fsl_runbootloader.h>

union option

Public Members

struct *user_app_boot_invoke_option_t* B

uint32_t U

struct B

2.99 SAI: Serial Audio Interface

2.100 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

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

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_SetMasterClockConfig(I2S_Type *base, *sai_master_clock_t* *config)

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

void SAI_TxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.

- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly    = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

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

static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

void SAI_TxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer

- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer

- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

void SAI_TransferTxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Tx handle.

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

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SAI_TransferRxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Rx handle.

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

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

void SAI_TransferTxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – tranmitter configurations.

void SAI_TransferRxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.

- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

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

status_t SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

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

void SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

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

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

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

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.10

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy

SAI Tx is busy.

enumerator kStatus_SAI_RxBusy

SAI Rx is busy.

enumerator kStatus_SAI_TxError

SAI Tx FIFO error.

enumerator kStatus_SAI_RxError

SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull

SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle

SAI Tx is idle

enumerator kStatus_SAI_RxIdle

SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.

Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.

Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.

Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx
Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx
Synchronous with another SAI receiver

enum _sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1

Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable

FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable

FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable

FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable

FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable

FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing
The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate
Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz
Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz
Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz
Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz
Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz
Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz
Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz
Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz
Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz
Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz
Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width
Audio word width.

Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_data_pin_state
sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine
sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled
sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead
sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite
sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite
sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead
sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite
sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type
sai transceiver type

Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
: sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
: sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
: sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
: sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
: sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
: master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
: sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
: sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
: sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
: sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
: sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
: SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)
: SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK
: Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

  If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG
: Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

  Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE
: SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE
: sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol
: Audio bus protocol in SAI

sai_sync_mode_t syncMode
: SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable
: Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource
: Bit Clock source

sai_master_slave_t masterSlave
: Master or slave

struct _sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz
: Sample rate of audio data

uint32_t bitWidth
: Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo
: Mono or stereo

uint32_t masterClockHz
: Master clock frequency in Hz

uint8_t watermark
: Watermark value

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct _sai_master_clock

#include <fsl_sai.h> master clock configurations

Public Members

bool mclkOutputEnable
: master clock output enable

uint32_t mclkHz
: target mclk frequency

uint32_t mclkSourceClkHz
: mclk source frequency

struct _sai_fifo
: *#include <fsl_sai.h>* sai fifo configurations

Public Members

bool fifoContinueOneError
: fifo continues when error occur

sai_fifo_combine_t fifoCombine
: fifo combine mode

sai_fifo_packing_t fifoPacking
: fifo packing mode

uint8_t fifoWatermark
: fifo watermark

struct _sai_bit_clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkInputDelay
: bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
: bit clock polarity

sai_bclk_source_t bclkSource
: bit Clock source

struct _sai_frame_sync
: *#include <fsl_sai.h>* sai frame sync configurations

Public Members

uint8_t frameSyncWidth
: frame sync width in number of bit clocks

bool frameSyncEarly
: TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
: internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity
frame sync polarity

struct __sai__serial__data
#include <fsl_sai.h> sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
configure the number of bits in the first word in each frame

uint8_t dataWordNLength
configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
used to record the data length for dma transfer

uint8_t dataFirstBitShifted
Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
configure the number of words in each frame

uint32_t dataMaskedWord
configure whether the transmit word is masked

struct __sai__transceiver
#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

struct __sai__transfer
: *#include <fsl_sai.h>* SAI transfer structure.

Public Members

uint8_t *data
: Data start address to transfer.

size_t dataSize
: Transfer size.

struct __sai__handle
: *#include <fsl_sai.h>* SAI handle structure.

Public Members

I2S_Type *base
: base address

uint32_t state
: Transfer status

sai_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

uint8_t bitWidth
: Bit width for transfer, 8/16/24/32 bits

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

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

uint8_t watermark

Watermark value

2.101 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

2.102 Sbloader

SB loader status codes.

Values:

enumerator kStatusRomLdrSectionOverrun

enumerator kStatusRomLdrSignature

enumerator kStatusRomLdrSectionLength

enumerator kStatusRomLdrUnencryptedOnly

enumerator kStatusRomLdrEOFReached

enumerator kStatusRomLdrChecksum

enumerator kStatusRomLdrCrc32Error

enumerator kStatusRomLdrUnknownCommand

enumerator kStatusRomLdrIdNotFound

enumerator kStatusRomLdrDataUnderrun

enumerator kStatusRomLdrJumpReturned

enumerator kStatusRomLdrCallFailed

enumerator kStatusRomLdrKeyNotFound

enumerator kStatusRomLdrSecureOnly

enumerator kStatusRomLdrResetReturned

enumerator kStatusRomLdrRollbackBlocked

enumerator kStatusRomLdrInvalidSectionMacCount

enumerator kStatusRomLdrUnexpectedCommand

enumerator kStatusRomLdrBadSBKEK

enumerator kStatusRomLdrPendingJumpCommand

enum _fw_version_check_option

Values:

enumerator kRomLdr_FwCheckOption_SecureVersion

enumerator kRomLdr_FwCheckOption_NonSecureVersion

enum _sectionType

sb3 section definitions

section type

Values:

enumerator kSectionNone

end or invalid

enumerator kSectionDataRange

enumerator kSectionDiffUpdate

enumerator kSectionDDRConfig

enumerator kSectionRegister

Values:

enumerator kFwVerChk_Id_none

enumerator kFwVerChk_Id_nonsecure

enumerator kFwVerChk_Id_secure

enum _loader_command_sb3

loader command enum

Values:

enumerator kSB3_CmdInvalid

enumerator kSB3_CmdErase

enumerator kSB3_CmdLoad

enumerator kSB3_CmdExecute

enumerator kSB3_CmdCall

enumerator kSB3_CmdProgramFuse

enumerator kSB3_CmdProgramIFR

enumerator kSB3_CmdLoadCmac

enumerator kSB3_CmdCopy

enumerator kSB3_CmdLoadHashLocking

enumerator kSB3_CmdLoadKeyBlob

enumerator kSB3_CmdConfigMem

enumerator kSB3_CmdFillMem

enumerator kSB3_CmdFwVerCheck

enum kb_operation_t

Details of the operation to be performed by the ROM.

The kRomAuthenticateImage operation requires the entire signed image to be available to the application.

Values:

enumerator kRomAuthenticateImage

Authenticate a signed image.

enumerator kRomLoadImage

Load SB file.

enumerator kRomOperationCount

typedef uint8_t chunk_t[16]

typedef struct *_boot_cmd* boot_cmd_t

Boot command definition.

typedef struct *_boot_hdr1* boot_hdr1_t

Definition for boot image file header chunk 1.

typedef struct *_boot_hdr2* boot_hdr2_t

Definition for boot image file header chunk 2.

typedef struct *_ldr_Context* ldr_Context_t

Provides forward reference to the loader context definition.

typedef *status_t* (*pLdrFnc_t)(*ldr_Context_t* *context)

Function pointer definition for all loader action functions.

typedef *status_t* (*pJumpFnc_t)(uint32_t parameter)

Jump command function pointer definition.

typedef *status_t* (*pCallFnc_t)(uint32_t parameter, uint32_t *func)
Call command function pointer definition.

typedef struct *Crc32Data* crc32_data_t
State information for the CRC32 algorithm.

typedef struct *soc_memory_map_struct* soc_mem_regions_t

typedef struct *arena_context* arena_context_t

typedef struct *mem_region* mem_region_t
Memory region information table.

typedef struct *memory_attribute_struct* mem_attribute_t
Memory Attribute Structure.

typedef struct *memory_context_struct* mem_context_t
Memory context structure.

typedef struct *api_memory_region_interface* api_memory_region_interface_t
Memory region interface structure.

typedef struct *memory_map_entry* api_memory_map_entry_t
Memory entry data structure.

typedef struct *api_core_context* api_core_context_t
The API context structure.

typedef uint8_t chunk_v3_t[16]

typedef struct *_ldr_buf* ldr_buf_t

typedef struct *_ldr_Context_v3* ldr_Context_v3_t
Provides forward reference to the loader context definition.

typedef *status_t* (*pLdrFnc_v3_t)(*ldr_Context_v3_t* *content)
Function pointer definition for all loader action functions.

typedef enum *_sectionType* section_type_t
sb3 section definitions

section type

typedef struct *range_header* sb3_data_range_header_t
section data range structure

typedef struct *range_header_expansion* sb3_data_range_expansion_t

typedef struct *copy_memory_expansion* sb3_copy_memory_expansion_t

typedef struct *copy* sb3_copy_memory_t

typedef struct *load_keyblob* sb3_load_keyblob_t

typedef struct *fill_memory_expansion* sb3_fill_memory_expansion_t

typedef struct *fill_memory* sb3_fill_memory_t

typedef struct *config_memory* sb3_config_memory_t

typedef struct *fw_ver_check* sb3_fw_ver_check_t

typedef struct *section_header* sb3_section_header_t
sb3 DATA section header format

typedef enum *_loader_command_sb3* sb3_cmd_t

loader command enum

status_t Sbloader_Init(*api_core_context_t* *ctx)

Perform the Sbloader runtime environment initialization This API is used for initializing the sbloader state machine before calling the api_sbloader_pump. This API should be called after the iap_api_init API.

Parameters

- ctx – Pointer to IAP API core context structure.

Return values

kStatus_Success – Api was executed succesfuly.

status_t Sbloader_Pump(*api_core_context_t* *ctx, uint8_t *data, uint32_t length)

Handle the SB data stream This API is used for handling the secure binary(SB3.1 format) data stream, which is used for image update, lifecycle advancing, etc. This API should be called after the iap_api_init and api_sbloader_init APIs.

Parameters

- ctx – Pointer to IAP API core context structure.
- data – Pointer to source data that is the sb file buffer data.
- length – The size of the process buffer data.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – An invalid argument is provided.
- kStatus_Fail – API execution failed.

status_t Sbloader_Finalize(*api_core_context_t* *ctx)

Finish the sbloader handling The API is used for finalizing the sbloader operations.

Parameters

- ctx – Pointer to IAP API core context structure.

Return values

kStatus_Success – Api was executed succesfuly.

SB_FILE_MAJOR_VERSION

Determines the version of SB loader implementation (1: sb1.0; 2: sb2.0; 3.1: sb3.1)

SB_FILE_MINOR_VERSION

kStatusGroup_SBLoader

Bootloader status group numbers.

RAM_REGION_COUNT

Contiguous RAM region count.

FLASH_REGION_COUNT

Contiguous FLASH region count.

FFR_REGION_COUNT

Contiguous FFR region count.

MEM_INTERFACE_COUNT

Memory Interface count.

FLEXSPINOR_REGION_COUNT
: Contiguous FLEXSPINOR meomry count.

BYTES_PER_CHUNK
: Defines the number of bytes in a cipher block (chunk). This is dictated by the encryption algorithm.

SB_SECTION_COUNT_MAX

BOOT_SIGNATURE
: Boot image signature in 32-bit little-endian format “PMTS”.

BOOT_SIGNATURE2
: Boot image signature in 32-bit little-endian format “ltgs”.

FFLG_DISPLAY_PROGRESS
: These define file header flags.

SFLG_SECTION_BOOTABLE
: These define section header flags.

CFLG_LAST_TAG
: These define boot command flags.

ROM_ERASE_ALL_MASK
: ROM_ERASE_CMD flags.

ROM_ERASE_ALL_UNSECURE_MASK

ROM_JUMP_SP_MASK
: ROM_JUMP_CMD flags.

ROM_MEM_DEVICE_ID_SHIFT
: Memory device id shift at sb command flags.

ROM_MEM_DEVICE_ID_MASK
: Memory device id mask.

ROM_MEM_GROUP_ID_SHIFT
: Memory group id shift at sb command flags.

ROM_MEM_GROUP_ID_MASK
: Memory group id flags mask.

ROM_PROG_8BYTE_MASK
: ROM_PROG_CMD flags.

ROM_NOP_CMD
: These define the boot command tags.

ROM_TAG_CMD

ROM_LOAD_CMD

ROM_FILL_CMD

ROM_JUMP_CMD

ROM_CALL_CMD

ROM_MODE_CMD

ROM_ERASE_CMD

ROM_RESET_CMD

ROM_MEM_ENABLE_CMD

ROM_PROG_CMD

ROM_FW_VER_CHK

SBLOADER_CMD_SET_IN_ISP_MODE

SBLOADER_CMD_SET_IN_REC_MODE

ROM_BOOT_SECTION_ID
: Plugin return codes.

ROM_BOOT_IMAGE_ID

SB3_BYTES_PER_CHUNK
: Defines the number of bytes in a cipher block (chunk). This is dictated by the encryption algorithm.

SB3_DATA_RANGE_HEADER_FLAGS_ERASE_MASK
: bit 0

SB3_DATA_RANGE_HEADER_FLAGS_LOAD_MASK
: bit 1

SBLOADER_V3_CMD_SET_ALL
: The all of the allowed command.

SBLOADER_V3_CMD_SET_IN_ISP_MODE
: The allowed command set in ISP mode.

SBLOADER_V3_CMD_SET_IN_REC_MODE
: The allowed command set in recovery mode.

SB3_DATA_BUFFER_SIZE_IN_BYTE

struct _boot_cmd
: *#include <fsl_sbloader.h>* Boot command definition.

Public Members

uint8_t checksum
: 8-bit checksum over command chunk

uint8_t tag
: command tag (identifier)

uint16_t flags
: command flags (modifier)

uint32_t address
: address argument

uint32_t count
: count argument

uint32_t data
: data argument

struct _boot_hdr1
: *#include <fsl_sbloader.h>* Definition for boot image file header chunk 1.

Public Members

uint32_t hash
: last 32-bits of SHA-1 hash

uint32_t signature
: must equal “STMP”

uint8_t major
: major file format version

uint8_t minor
: minor file format version

uint16_t fileFlags
: global file flags

uint32_t fileChunks
: total chunks in the file

struct __boot__hdr2
: *#include <fsl_sbloader.h>* Definition for boot image file header chunk 2.

Public Members

uint32_t bootOffset
: chunk offset to the first boot section

uint32_t bootSectID
: section ID of the first boot section

uint16_t keyCount
: number of keys in the key dictionary

uint16_t keyOffset
: chunk offset to the key dictionary

uint16_t hdrChunks
: number of chunks in the header

uint16_t sectCount
: number of sections in the image

struct Crc32Data
: *#include <fsl_sbloader.h>* State information for the CRC32 algorithm.

Public Members

uint32_t currentCrc
: Current CRC value.

uint32_t byteCountCrc
: Number of bytes processed.

struct __ldr__Context
: *#include <fsl_sbloader.h>* Loader context definition.

Public Members

pLdrFnc_t Action
: pointer to loader action function

uint32_t fileChunks
: chunks remaining in file

uint32_t sectChunks
: chunks remaining in section

uint32_t bootSectChunks
: number of chunks we need to complete the boot section

uint32_t receivedChunks
: number of chunks we need to complete the boot section

uint16_t fileFlags
: file header flags

uint16_t keyCount
: number of keys in the key dictionary

uint32_t objectID
: ID of the current boot section or image

crc32_data_t crc32
: crc calculated over load command payload

uint8_t *src
: source buffer address

chunk_t initVector
: decryption initialization vector

chunk_t dek
: chunk size DEK if the image is encrypted

chunk_t scratchPad
: chunk size scratch pad area

boot_cmd_t bootCmd
: current boot command

uint32_t skipCount
: Number of chunks to skip

bool skipToEnd
: true if skipping to end of file

uint32_t offsetSignatureBytes
: offset to signagure block header in bytesn

struct soc_memory_map_struct
: *#include <fsl_sbloader.h>*

struct arena_context
: *#include <fsl_sbloader.h>*

struct mem_region
: *#include <fsl_sbloader.h>* Memory region information table.

struct memory_attribute_struct
#include <fsl_sbloader.h> Memory Attribute Structure.

struct memory_context_struct
#include <fsl_sbloader.h> Memory context structure.

struct api_memory_region_interface
#include <fsl_sbloader.h> Memory region interface structure.

struct memory_map_entry
#include <fsl_sbloader.h> Memory entry data structure.

struct api_core_context
#include <fsl_sbloader.h> The API context structure.

struct _ldr_buf
#include <fsl_sbloader_v3.h>

struct range_header
#include <fsl_sbloader_v3.h> section data range structure

struct range_header_expansion
#include <fsl_sbloader_v3.h>

struct copy_memory_expansion
#include <fsl_sbloader_v3.h>

struct copy
#include <fsl_sbloader_v3.h>

struct load_keyblob
#include <fsl_sbloader_v3.h>

struct fill_memory_expansion
#include <fsl_sbloader_v3.h>

Public Members

uint32_t pattern
word to be used as pattern

struct fill_memory
#include <fsl_sbloader_v3.h>

struct config_memory
#include <fsl_sbloader_v3.h>

Public Members

uint32_t address
address of config blob

struct fw_ver_check
#include <fsl_sbloader_v3.h>

struct section_header
#include <fsl_sbloader_v3.h> sb3 DATA section header format

struct kb_region_t
#include <fsl_sbloader_v3.h> Memory region definition.

struct kb_load_sb_t

#include <fsl_sbloader_v3.h>

struct kb_authenticate_t

#include <fsl_sbloader_v3.h>

struct kb_options_t

#include <fsl_sbloader_v3.h>

Public Members

uint32_t version

Should be set to #kKbootApiVersion.

uint8_t *buffer

Caller-provided buffer used by Kboot.

struct _ldr_Context_v3

#include <fsl_sbloader_v3.h> Loader context definition.

Public Members

pLdrFnc_v3_t Action

pointer to loader action function

uint32_t block_size

size of each block in bytes

uint32_t block_data_size

data size in bytes (NBOOT_SB3_CHUNK_SIZE_IN_BYTES)

uint32_t block_data_total

data max size in bytes (block_size * data_size

uint32_t block_buffer_size

block0 and block size

uint32_t processedBlocks

will be used for both block0 and blockx

bool in_data_block

data block offset in a block. in progress of handling a data block within a block

bool in_data_section

in progress of handling a data section within a data block

bool in_data_range

in progress of handling a data range within a data section

uint32_t commandSet

support command set during sb file handling

uint8_t data_buffer[(*MAX*(128, *NBOOT_KEY_BLOB_SIZE_IN_BYTE_MAX*))]

temporary data buffer

kb_options_t fromAPI

options from ROM API

struct ramRegions

struct flashRegions

struct ffrRegions

struct flexspiNorRegions

union ___unnamed49___

Public Members

kb_authenticate_t authenticate
Settings for #kKbootAuthenticate operation.

kb_load_sb_t loadSB
Settings for #kKbootLoadSB operation.

2.103 SCTimer: SCTimer/PWM (SCT)

status_t SCTIMER_Init(SCT_Type *base, const *sctimer_config_t* *config)
Ungates the SCTimer clock and configures the peripheral for basic operation.

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

Parameters

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

Returns
kStatus_Success indicates success; Else indicates failure.

void SCTIMER_Deinit(SCT_Type *base)
Gates the SCTimer clock.

Parameters

- base – SCTimer peripheral base address

void SCTIMER_GetDefaultConfig(*sctimer_config_t* *config)
Fills in the SCTimer configuration structure with the default settings.

The default values are:

```
config->enableCounterUnify = true;
config->clockMode = kSCTIMER_System_ClockMode;
config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
config->enableBidirection_l = false;
config->enableBidirection_h = false;
config->prescale_l = 0U;
config->prescale_h = 0U;
config->outInitState = 0U;
config->inputsync  = 0xFU;
```

Parameters

- config – Pointer to the user configuration structure.

status_t SCTIMER_SetupPwm(SCT_Type *base, const *sctimer_pwm_signal_param_t* *pwmParams, *sctimer_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event)

Configures the PWM signal parameters.

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events are enabled for the same state. The state number can be retrieved by calling the function SCTIMER_GetCurrentStateNumber(). The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

Note: When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's pwmFreq_Hz.

Parameters

- base – SCTimer peripheral base address
- pwmParams – PWM parameters to configure the output
- mode – PWM operation mode, options available in enumeration sctimer_pwm_mode_t
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – SCTimer counter clock in Hz
- event – Pointer to a variable where the PWM period event number is stored

Returns

kStatus_Success on success kStatus_Fail If we have hit the limit in terms of number of events created or if an incorrect PWM dutycylce is passed in.

void SCTIMER_UpdatePwmDutycycle(SCT_Type *base, *sctimer_out_t* output, uint8_t dutyCyclePercent, uint32_t event)

Updates the duty cycle of an active PWM signal.

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

Parameters

- base – SCTimer peripheral base address
- output – The output to configure
- dutyCyclePercent – New PWM pulse width; the value should be between 1 to 100
- event – Event number associated with this PWM signal. This was returned to the user by the function SCTIMER_SetupPwm().

static inline void SCTIMER_EnableInterrupts(SCT_Type *base, uint32_t mask)

Enables the selected SCTimer interrupts.

Parameters

- base – SCTimer peripheral base address

- mask – The interrupts to enable. This is a logical OR of members of the enumeration sctimer_interrupt_enable_t

static inline void SCTIMER_DisableInterrupts(SCT_Type *base, uint32_t mask)

Disables the selected SCTimer interrupts.

Parameters

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

static inline uint32_t SCTIMER_GetEnabledInterrupts(SCT_Type *base)

Gets the enabled SCTimer interrupts.

Parameters

- base – SCTimer peripheral base address

Returns

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

static inline uint32_t SCTIMER_GetStatusFlags(SCT_Type *base)

Gets the SCTimer status flags.

Parameters

- base – SCTimer peripheral base address

Returns

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

static inline void SCTIMER_ClearStatusFlags(SCT_Type *base, uint32_t mask)

Clears the SCTimer status flags.

Parameters

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

static inline void SCTIMER_StartTimer(SCT_Type *base, uint32_t countertoStart)

Starts the SCTimer counter.

Note: In 16-bit mode, we can enable both Counter_L and Counter_H, In 32-bit mode, we only can select Counter_U.

Parameters

- base – SCTimer peripheral base address
- countertoStart – The SCTimer counters to enable. This is a logical OR of members of the enumeration sctimer_counter_t.

static inline void SCTIMER_StopTimer(SCT_Type *base, uint32_t countertoStop)

Halts the SCTimer counter.

Parameters

- base – SCTimer peripheral base address
- countertoStop – The SCTimer counters to stop. This is a logical OR of members of the enumeration sctimer_counter_t.

status_t SCTIMER_CreateAndScheduleEvent(SCT_Type *base, *sctimer_event_t* howToMonitor, uint32_t matchValue, uint32_t whichIO, *sctimer_counter_t* whichCounter, uint32_t *event)

Create an event that is triggered on a match or IO and schedule in current state.

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

- base – SCTimer peripheral base address
- howToMonitor – Event type; options are available in the enumeration sctimer_interrupt_enable_t
- matchValue – The match value that will be programmed to a match register
- whichIO – The input or output that will be involved in event triggering. This field is ignored if the event type is "match only"
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Pointer to a variable where the new event number is stored

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

void SCTIMER_ScheduleEvent(SCT_Type *base, uint32_t event)

Enable an event in the current state.

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function SCTIMER_SetupPwm() or function SCTIMER_CreateAndScheduleEvent() .

Parameters

- base – SCTimer peripheral base address
- event – Event number to enable in the current state

status_t SCTIMER_IncreaseState(SCT_Type *base)

Increase the state by 1.

All future events created by calling the function SCTIMER_ScheduleEvent() will be enabled in this new state.

Parameters

- base – SCTimer peripheral base address

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of states used

uint32_t SCTIMER_GetCurrentState(SCT_Type *base)

Provides the current state.

User can use this to set the next state by calling the function SCTIMER_SetupNextStateAction().

Parameters

- base – SCTimer peripheral base address

Returns
The current state

static inline void SCTIMER_SetCounterState(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t state)

Set the counter current state.

The function is to set the state variable bit field of STATE register. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- state – The counter current state number (only support range from 0~31).

static inline uint16_t SCTIMER_GetCounterState(SCT_Type *base, *sctimer_counter_t* whichCounter)

Get the counter current state value.

The function is to get the state variable bit field of STATE register.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns
The the counter current state value.

status_t SCTIMER_SetupCaptureAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t *captureRegister, uint32_t event)

Setup capture of the counter value on trigger of a selected event.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- captureRegister – Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
- event – Event number that will trigger the capture

Returns
kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of match/capture registers available

void SCTIMER_SetCallback(SCT_Type *base, *sctimer_event_callback_t* callback, uint32_t event)

Receive noticification when the event trigger an interrupt.

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

- base – SCTimer peripheral base address

- event – Event number that will trigger the interrupt
- callback – Function to invoke when the event is triggered

static inline void SCTIMER_SetupStateLdMethodAction(SCT_Type *base, uint32_t event, bool fgLoad)

Change the load method of transition to the specified state.

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- event – Event number that will change the method to trigger the state transition
- fgLoad – The method to load highest-numbered event occurring for that state to the STATE register.
  - true: Load the STATEV value to STATE when the event occurs to be the next state.
  - false: Add the STATEV value to STATE when the event occurs to be the next state.

static inline void SCTIMER_SetupNextStateActionwithLdMethod(SCT_Type *base, uint32_t nextState, uint32_t event, bool fgLoad)

Transition to the specified state with Load method.

This transition will be triggered by the event number that is passed in by the user, the method decide how to load the highest-numbered event occurring for that state to the STATE register.

Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition
- fgLoad – The method to load the highest-numbered event occurring for that state to the STATE register.
  - true: Load the STATEV value to STATE when the event occurs to be the next state.
  - false: Add the STATEV value to STATE when the event occurs to be the next state.

static inline void SCTIMER_SetupNextStateAction(SCT_Type *base, uint32_t nextState, uint32_t event)

Transition to the specified state.

Deprecated:

Do not use this function. It has been superceded by SCTIMER_SetupNextStateActionwithLdMethod

This transition will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address

- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition

static inline void SCTIMER_SetupEventActiveDirection(SCT_Type *base, *sctimer_event_active_direction_t* activeDirection, uint32_t event)

Setup event active direction when the counters are operating in BIDIR mode.

Parameters

- base – SCTimer peripheral base address
- activeDirection – Event generation active direction, see sctimer_event_active_direction_t.
- event – Event number that need setup the active direction.

static inline void SCTIMER_SetupOutputSetAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Set the Output.

This output will be set when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to set
- event – Event number that will trigger the output change

static inline void SCTIMER_SetupOutputClearAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Clear the Output.

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to clear
- event – Event number that will trigger the output change

void SCTIMER_SetupOutputToggleAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Toggle the output level.

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to toggle
- event – Event number that will trigger the output change

static inline void SCTIMER_SetupCounterLimitAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Limit the running counter.

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address

- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be limited

static inline void SCTIMER_SetupCounterStopAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Stop the running counter.

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be stopped

static inline void SCTIMER_SetupCounterStartAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Re-start the stopped counter.

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to re-start

static inline void SCTIMER_SetupCounterHaltAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Halt the running counter.

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the SCTIMER_StartTimer() function.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be halted

static inline void SCTIMER_SetupDmaTriggerAction(SCT_Type *base, uint32_t dmaNumber, uint32_t event)

Generate a DMA request.

DMA request will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- dmaNumber – The DMA request to generate
- event – Event number that will trigger the DMA request

static inline void SCTIMER_SetCOUNTValue(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t value)

Set the value of counter.

The function is to set the value of Count register, Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- value – the counter value update to the COUNT register.

static inline uint32_t SCTIMER_GetCOUNTValue(SCT_Type *base, *sctimer_counter_t* whichCounter)

Get the value of counter.

The function is to read the value of Count register, software can read the counter registers at any time..

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The value of counter selected.

static inline void SCTIMER_SetEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Set the state mask bit field of EV_STATE register.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be set.
- state – The state value in which the event is enabled to occur.

static inline void SCTIMER_ClearEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Clear the state mask bit field of EV_STATE register.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be clear.
- state – The state value in which the event is disabled to occur.

static inline bool SCTIMER_GetEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Get the state mask bit field of EV_STATE register.

Note: This function is to check whether the event is enabled in a specific state.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be read.
- state – The state value.

Returns
The the state mask bit field of EV_STATE register.

- true: The event is enable in state.
- false: The event is disable in state.

static inline uint32_t SCTIMER_GetCaptureValue(SCT_Type *base, *sctimer_counter_t* whichCounter, uint8_t capChannel)

Get the value of capture register.

This function returns the captured value upon occurrence of the events selected by the corresponding Capture Control registers occurred.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- capChannel – SCTimer capture register of capture channel.

Returns
The SCTimer counter value at which this register was last captured.

void SCTIMER_EventHandleIRQ(SCT_Type *base)

SCTimer interrupt handler.

Parameters

- base – SCTimer peripheral base address.

FSL_SCTIMER_DRIVER_VERSION

Version

enum _sctimer_pwm_mode

SCTimer PWM operation modes.

Values:

enumerator kSCTIMER_EdgeAlignedPwm

Edge-aligned PWM

enumerator kSCTIMER_CenterAlignedPwm

Center-aligned PWM

enum _sctimer_counter

SCTimer counters type.

Values:

enumerator kSCTIMER_Counter_L

16-bit Low counter.

enumerator kSCTIMER_Counter_H

16-bit High counter.

enumerator kSCTIMER_Counter_U

32-bit Unified counter.

enum _sctimer_input

List of SCTimer input pins.

Values:

enumerator kSCTIMER_Input_0
SCTIMER input 0

enumerator kSCTIMER_Input_1
SCTIMER input 1

enumerator kSCTIMER_Input_2
SCTIMER input 2

enumerator kSCTIMER_Input_3
SCTIMER input 3

enumerator kSCTIMER_Input_4
SCTIMER input 4

enumerator kSCTIMER_Input_5
SCTIMER input 5

enumerator kSCTIMER_Input_6
SCTIMER input 6

enumerator kSCTIMER_Input_7
SCTIMER input 7

enum _sctimer_out
List of SCTimer output pins.

Values:

enumerator kSCTIMER_Out_0
SCTIMER output 0

enumerator kSCTIMER_Out_1
SCTIMER output 1

enumerator kSCTIMER_Out_2
SCTIMER output 2

enumerator kSCTIMER_Out_3
SCTIMER output 3

enumerator kSCTIMER_Out_4
SCTIMER output 4

enumerator kSCTIMER_Out_5
SCTIMER output 5

enumerator kSCTIMER_Out_6
SCTIMER output 6

enumerator kSCTIMER_Out_7
SCTIMER output 7

enumerator kSCTIMER_Out_8
SCTIMER output 8

enumerator kSCTIMER_Out_9
SCTIMER output 9

enum _sctimer_pwm_level_select
SCTimer PWM output pulse mode: high-true, low-true or no output.

Values:

enumerator kSCTIMER_LowTrue
Low true pulses

enumerator kSCTIMER_HighTrue
High true pulses

enum _sctimer_clock_mode
SCTimer clock mode options.

Values:

enumerator kSCTIMER_System_ClockMode
System Clock Mode

enumerator kSCTIMER_Sampled_ClockMode
Sampled System Clock Mode

enumerator kSCTIMER_Input_ClockMode
SCT Input Clock Mode

enumerator kSCTIMER_Asynchronous_ClockMode
Asynchronous Mode

enum _sctimer_clock_select
SCTimer clock select options.

Values:

enumerator kSCTIMER_Clock_On_Rise_Input_0
Rising edges on input 0

enumerator kSCTIMER_Clock_On_Fall_Input_0
Falling edges on input 0

enumerator kSCTIMER_Clock_On_Rise_Input_1
Rising edges on input 1

enumerator kSCTIMER_Clock_On_Fall_Input_1
Falling edges on input 1

enumerator kSCTIMER_Clock_On_Rise_Input_2
Rising edges on input 2

enumerator kSCTIMER_Clock_On_Fall_Input_2
Falling edges on input 2

enumerator kSCTIMER_Clock_On_Rise_Input_3
Rising edges on input 3

enumerator kSCTIMER_Clock_On_Fall_Input_3
Falling edges on input 3

enumerator kSCTIMER_Clock_On_Rise_Input_4
Rising edges on input 4

enumerator kSCTIMER_Clock_On_Fall_Input_4
Falling edges on input 4

enumerator kSCTIMER_Clock_On_Rise_Input_5
Rising edges on input 5

enumerator kSCTIMER_Clock_On_Fall_Input_5
Falling edges on input 5

enumerator kSCTIMER_Clock_On_Rise_Input_6
Rising edges on input 6

enumerator kSCTIMER_Clock_On_Fall_Input_6
Falling edges on input 6

enumerator kSCTIMER_Clock_On_Rise_Input_7
Rising edges on input 7

enumerator kSCTIMER_Clock_On_Fall_Input_7
Falling edges on input 7

enum _sctimer_conflict_resolution
SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Values:

enumerator kSCTIMER_ResolveNone
No change

enumerator kSCTIMER_ResolveSet
Set output

enumerator kSCTIMER_ResolveClear
Clear output

enumerator kSCTIMER_ResolveToggle
Toggle output

enum _sctimer_event_active_direction
List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

Values:

enumerator kSCTIMER_ActiveIndependent
This event is triggered regardless of the count direction.

enumerator kSCTIMER_ActiveInCountUp
This event is triggered only during up-counting when BIDIR = 1.

enumerator kSCTIMER_ActiveInCountDown
This event is triggered only during down-counting when BIDIR = 1.

enum _sctimer_event
List of SCTimer event types.

Values:

enumerator kSCTIMER_InputLowOrMatchEvent

enumerator kSCTIMER_InputRiseOrMatchEvent

enumerator kSCTIMER_InputFallOrMatchEvent

enumerator kSCTIMER_InputHighOrMatchEvent

enumerator kSCTIMER_MatchEventOnly

enumerator kSCTIMER_InputLowEvent

enumerator kSCTIMER_InputRiseEvent

enumerator kSCTIMER_InputFallEvent

enumerator kSCTIMER_InputHighEvent

enumerator kSCTIMER_InputLowAndMatchEvent

enumerator kSCTIMER_InputRiseAndMatchEvent

enumerator kSCTIMER_InputFallAndMatchEvent

enumerator kSCTIMER_InputHighAndMatchEvent

enumerator kSCTIMER_OutputLowOrMatchEvent

enumerator kSCTIMER_OutputRiseOrMatchEvent

enumerator kSCTIMER_OutputFallOrMatchEvent

enumerator kSCTIMER_OutputHighOrMatchEvent

enumerator kSCTIMER_OutputLowEvent

enumerator kSCTIMER_OutputRiseEvent

enumerator kSCTIMER_OutputFallEvent

enumerator kSCTIMER_OutputHighEvent

enumerator kSCTIMER_OutputLowAndMatchEvent

enumerator kSCTIMER_OutputRiseAndMatchEvent

enumerator kSCTIMER_OutputFallAndMatchEvent

enumerator kSCTIMER_OutputHighAndMatchEvent

enum _sctimer_interrupt_enable

List of SCTimer interrupts.

Values:

enumerator kSCTIMER_Event0InterruptEnable

Event 0 interrupt

enumerator kSCTIMER_Event1InterruptEnable

Event 1 interrupt

enumerator kSCTIMER_Event2InterruptEnable

Event 2 interrupt

enumerator kSCTIMER_Event3InterruptEnable

Event 3 interrupt

enumerator kSCTIMER_Event4InterruptEnable

Event 4 interrupt

enumerator kSCTIMER_Event5InterruptEnable

Event 5 interrupt

enumerator kSCTIMER_Event6InterruptEnable

Event 6 interrupt

enumerator kSCTIMER_Event7InterruptEnable
Event 7 interrupt

enumerator kSCTIMER_Event8InterruptEnable
Event 8 interrupt

enumerator kSCTIMER_Event9InterruptEnable
Event 9 interrupt

enumerator kSCTIMER_Event10InterruptEnable
Event 10 interrupt

enumerator kSCTIMER_Event11InterruptEnable
Event 11 interrupt

enumerator kSCTIMER_Event12InterruptEnable
Event 12 interrupt

enum _sctimer_status_flags
List of SCTimer flags.

Values:

enumerator kSCTIMER_Event0Flag
Event 0 Flag

enumerator kSCTIMER_Event1Flag
Event 1 Flag

enumerator kSCTIMER_Event2Flag
Event 2 Flag

enumerator kSCTIMER_Event3Flag
Event 3 Flag

enumerator kSCTIMER_Event4Flag
Event 4 Flag

enumerator kSCTIMER_Event5Flag
Event 5 Flag

enumerator kSCTIMER_Event6Flag
Event 6 Flag

enumerator kSCTIMER_Event7Flag
Event 7 Flag

enumerator kSCTIMER_Event8Flag
Event 8 Flag

enumerator kSCTIMER_Event9Flag
Event 9 Flag

enumerator kSCTIMER_Event10Flag
Event 10 Flag

enumerator kSCTIMER_Event11Flag
Event 11 Flag

enumerator kSCTIMER_Event12Flag
Event 12 Flag

enumerator kSCTIMER_BusErrorLFlag
Bus error due to write when L counter was not halted

enumerator kSCTIMER_BusErrorHFlag
Bus error due to write when H counter was not halted

typedef enum *_sctimer_pwm_mode* sctimer_pwm_mode_t
SCTimer PWM operation modes.

typedef enum *_sctimer_counter* sctimer_counter_t
SCTimer counters type.

typedef enum *_sctimer_input* sctimer_input_t
List of SCTimer input pins.

typedef enum *_sctimer_out* sctimer_out_t
List of SCTimer output pins.

typedef enum *_sctimer_pwm_level_select* sctimer_pwm_level_select_t
SCTimer PWM output pulse mode: high-true, low-true or no output.

typedef struct *_sctimer_pwm_signal_param* sctimer_pwm_signal_param_t
Options to configure a SCTimer PWM signal.

typedef enum *_sctimer_clock_mode* sctimer_clock_mode_t
SCTimer clock mode options.

typedef enum *_sctimer_clock_select* sctimer_clock_select_t
SCTimer clock select options.

typedef enum *_sctimer_conflict_resolution* sctimer_conflict_resolution_t
SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

typedef enum *_sctimer_event_active_direction* sctimer_event_active_direction_t
List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

typedef enum *_sctimer_event* sctimer_event_t
List of SCTimer event types.

typedef void (*sctimer_event_callback_t)(void)
SCTimer callback typedef.

typedef enum *_sctimer_interrupt_enable* sctimer_interrupt_enable_t
List of SCTimer interrupts.

typedef enum *_sctimer_status_flags* sctimer_status_flags_t
List of SCTimer flags.

typedef struct *_sctimer_config* sctimer_config_t
SCTimer configuration structure.

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

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

SCT_EV_STATE_STATEMSKn(x)

struct sctimer_pwm_signal_param
#include <fsl_sctimer.h> Options to configure a SCTimer PWM signal.

Public Members

sctimer_out_t output

The output pin to use to generate the PWM signal

sctimer_pwm_level_select_t level

PWM output active level select.

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).

struct _sctimer_config

#include <fsl_sctimer.h> SCTimer configuration structure.

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

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

Public Members

bool enableCounterUnify

true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters. User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

sctimer_clock_mode_t clockMode

SCT clock mode value

sctimer_clock_select_t clockSelect

SCT clock select value

bool enableBidirection_l

true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter

bool enableBidirection_h

true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter. This field is used only if the enableCounterUnify is set to false

uint8_t prescale_l

Prescale value to produce the L or unified counter clock

uint8_t prescale_h

Prescale value to produce the H counter clock. This field is used only if the enableCounterUnify is set to false

uint8_t outInitState

Defines the initial output value

uint8_t inputsync

SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16. it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ~bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNC0 (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 « INPUTSYNC2) | (1 « INPUTSYNC3);

2.104 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.

2.105 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;
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_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.

2.106 Smart Card

FSL_SMARTCARD_DRIVER_VERSION
: Smart card driver version 2.3.0.

Smart card Error codes.

Values:

enumerator kStatus_SMARTCARD_Success
: Transfer ends successfully

enumerator kStatus_SMARTCARD_TxBusy
: Transmit in progress

enumerator kStatus_SMARTCARD_RxBusy
: Receiving in progress

enumerator kStatus_SMARTCARD_NoTransferInProgress
: No transfer in progress

enumerator kStatus_SMARTCARD_Timeout
: Transfer ends with time-out

enumerator kStatus_SMARTCARD_Initialized
: Smart card driver is already initialized

enumerator kStatus_SMARTCARD_PhyInitialized
: Smart card PHY drive is already initialized

enumerator kStatus_SMARTCARD_CardNotActivated
: Smart card is not activated

enumerator kStatus_SMARTCARD_InvalidInput
Function called with invalid input arguments

enumerator kStatus_SMARTCARD_OtherError
Some other error occur

enum _smartcard_control
Control codes for the Smart card protocol timers and misc.

Values:

enumerator kSMARTCARD_EnableADT

enumerator kSMARTCARD_DisableADT

enumerator kSMARTCARD_EnableGTV

enumerator kSMARTCARD_DisableGTV

enumerator kSMARTCARD_ResetWWT

enumerator kSMARTCARD_EnableWWT

enumerator kSMARTCARD_DisableWWT

enumerator kSMARTCARD_ResetCWT

enumerator kSMARTCARD_EnableCWT

enumerator kSMARTCARD_DisableCWT

enumerator kSMARTCARD_ResetBWT

enumerator kSMARTCARD_EnableBWT

enumerator kSMARTCARD_DisableBWT

enumerator kSMARTCARD_EnableInitDetect

enumerator kSMARTCARD_EnableAnack

enumerator kSMARTCARD_DisableAnack

enumerator kSMARTCARD_ConfigureBaudrate

enumerator kSMARTCARD_SetupATRMode

enumerator kSMARTCARD_SetupT0Mode

enumerator kSMARTCARD_SetupT1Mode

enumerator kSMARTCARD_EnableReceiverMode

enumerator kSMARTCARD_DisableReceiverMode

enumerator kSMARTCARD_EnableTransmitterMode

enumerator kSMARTCARD_DisableTransmitterMode

enumerator kSMARTCARD_ResetWaitTimeMultiplier

enum _smartcard_card_voltage_class
Defines Smart card interface voltage class values.

Values:

enumerator kSMARTCARD_VoltageClassUnknown

enumerator kSMARTCARD_VoltageClassA5_0V

enumerator kSMARTCARD_VoltageClassB3_3V

enumerator kSMARTCARD_VoltageClassC1_8V

enum _smartcard_transfer_state

Defines Smart card I/O transfer states.

Values:

enumerator kSMARTCARD_IdleState

enumerator kSMARTCARD_WaitingForTSState

enumerator kSMARTCARD_InvalidTSDetecetedState

enumerator kSMARTCARD_ReceivingState

enumerator kSMARTCARD_TransmittingState

enum _smartcard_reset_type

Defines Smart card reset types.

Values:

enumerator kSMARTCARD_ColdReset

enumerator kSMARTCARD_WarmReset

enumerator kSMARTCARD_NoColdReset

enumerator kSMARTCARD_NoWarmReset

enum _smartcard_transport_type

Defines Smart card transport protocol types.

Values:

enumerator kSMARTCARD_T0Transport

enumerator kSMARTCARD_T1Transport

enum _smartcard_parity_type

Defines Smart card data parity types.

Values:

enumerator kSMARTCARD_EvenParity

enumerator kSMARTCARD_OddParity

enum _smartcard_card_convention

Defines data Convention format.

Values:

enumerator kSMARTCARD_DirectConvention

enumerator kSMARTCARD_InverseConvention

enum _smartcard_interface_control

Defines Smart card interface IC control types.

Values:

enumerator kSMARTCARD_InterfaceSetVcc

enumerator kSMARTCARD_InterfaceSetClockToResetDelay

enumerator kSMARTCARD_InterfaceReadStatus

enum _smartcard_direction

Defines transfer direction.

Values:

enumerator kSMARTCARD_Receive

enumerator kSMARTCARD_Transmit

typedef enum *_smartcard_control* smartcard_control_t

Control codes for the Smart card protocol timers and misc.

typedef enum *_smartcard_card_voltage_class* smartcard_card_voltage_class_t

Defines Smart card interface voltage class values.

typedef enum *_smartcard_transfer_state* smartcard_transfer_state_t

Defines Smart card I/O transfer states.

typedef enum *_smartcard_reset_type* smartcard_reset_type_t

Defines Smart card reset types.

typedef enum *_smartcard_transport_type* smartcard_transport_type_t

Defines Smart card transport protocol types.

typedef enum *_smartcard_parity_type* smartcard_parity_type_t

Defines Smart card data parity types.

typedef enum *_smartcard_card_convention* smartcard_card_convention_t

Defines data Convention format.

typedef enum *_smartcard_interface_control* smartcard_interface_control_t

Defines Smart card interface IC control types.

typedef enum *_smartcard_direction* smartcard_direction_t

Defines transfer direction.

typedef void (*smartcard_interface_callback_t)(void *smartcardContext, void *param)

Smart card interface interrupt callback function type.

typedef void (*smartcard_transfer_callback_t)(void *smartcardContext, void *param)

Smart card transfer interrupt callback function type.

typedef void (*smartcard_time_delay_t)(uint32_t us)

Time Delay function used to passive waiting using RTOS [us].

typedef struct *_smartcard_card_params* smartcard_card_params_t

Defines card-specific parameters for Smart card driver.

typedef struct *_smartcard_timers_state* smartcard_timers_state_t

Smart card defines the state of the EMV timers in the Smart card driver.

typedef struct *_smartcard_interface_config* smartcard_interface_config_t

Defines user specified configuration of Smart card interface.

typedef struct *_smartcard_xfer* smartcard_xfer_t

Defines user transfer structure used to initialize transfer.

typedef struct *_smartcard_context* smartcard_context_t

Runtime state of the Smart card driver.

SMARTCARD_INIT_DELAY_CLOCK_CYCLES

Smart card global define which specify number of clock cycles until initial 'TS' character has to be received.

SMARTCARD_EMV_ATR_DURATION_ETU

Smart card global define which specify number of clock cycles during which ATR string has to be received.

SMARTCARD_TS_DIRECT_CONVENTION

Smart card specification initial TS character definition of direct convention.

SMARTCARD_TS_INVERSE_CONVENTION

Smart card specification initial TS character definition of inverse convention.

struct _smartcard_card_params

#include <fsl_smartcard.h> Defines card-specific parameters for Smart card driver.

Public Members

uint16_t Fi

4 bits Fi - clock rate conversion integer

uint8_t fMax

Maximum Smart card frequency in MHz

uint8_t WI

8 bits WI - work wait time integer

uint8_t Di

4 bits DI - baud rate divisor

uint8_t BWI

4 bits BWI - block wait time integer

uint8_t CWI

4 bits CWI - character wait time integer

uint8_t BGI

4 bits BGI - block guard time integer

uint8_t GTN

8 bits GTN - extended guard time integer

uint8_t IFSC

Indicates IFSC value of the card

uint8_t modeNegotiable

Indicates if the card acts in negotiable or a specific mode.

uint8_t currentD

4 bits DI - current baud rate divisor

uint8_t status

Indicates smart card status

bool t0Indicated

Indicates ff T=0 indicated in TD1 byte

bool t1Indicated
: Indicates if T=1 indicated in TD2 byte

bool atrComplete
: Indicates whether the ATR received from the card was complete or not

bool atrValid
: Indicates whether the ATR received from the card was valid or not

bool present
: Indicates if a smart card is present

bool active
: Indicates if the smart card is activated

bool faulty
: Indicates whether smart card/interface is faulty

smartcard_card_convention_t convention
: Card convention, kSMARTCARD_DirectConvention for direct convention, kSMARTCARD_InverseConvention for inverse convention

struct _smartcard_timers_state
: *#include <fsl_smartcard.h>* Smart card defines the state of the EMV timers in the Smart card driver.

Public Members

volatile bool adtExpired
: Indicates whether ADT timer expired

volatile bool wwtExpired
: Indicates whether WWT timer expired

volatile bool cwtExpired
: Indicates whether CWT timer expired

volatile bool bwtExpired
: Indicates whether BWT timer expired

volatile bool initCharTimerExpired
: Indicates whether reception timer for initialization character (TS) after the RST has expired

struct _smartcard_interface_config
: *#include <fsl_smartcard.h>* Defines user specified configuration of Smart card interface.

Public Members

uint32_t smartCardClock
: Smart card interface clock [Hz]

uint32_t clockToResetDelay
: Indicates clock to RST apply delay [smart card clock cycles]

uint8_t clockModule
: Smart card clock module number

uint8_t clockModuleChannel
: Smart card clock module channel number

uint8_t clockModuleSourceClock
: Smart card clock module source clock [e.g., BusClk]

smartcard_card_voltage_class_t vcc
: Smart card voltage class

uint8_t controlPort
: Smart card PHY control port instance

uint8_t controlPin
: Smart card PHY control pin instance

uint8_t irqPort
: Smart card PHY Interrupt port instance

uint8_t irqPin
: Smart card PHY Interrupt pin instance

uint8_t resetPort
: Smart card reset port instance

uint8_t resetPin
: Smart card reset pin instance

uint8_t vsel0Port
: Smart card PHY Vsel0 control port instance

uint8_t vsel0Pin
: Smart card PHY Vsel0 control pin instance

uint8_t vsel1Port
: Smart card PHY Vsel1 control port instance

uint8_t vsel1Pin
: Smart card PHY Vsel1 control pin instance

uint8_t dataPort
: Smart card PHY data port instance

uint8_t dataPin
: Smart card PHY data pin instance

uint8_t dataPinMux
: Smart card PHY data pin mux option

uint8_t tsTimerId
: Numerical identifier of the External HW timer for Initial character detection

struct __smartcard_xfer
: *#include <fsl_smartcard.h>* Defines user transfer structure used to initialize transfer.

Public Members

smartcard_direction_t direction
: Direction of communication. (RX/TX)

uint8_t *buff
: The buffer of data.

size_t size
: The number of transferred units.

struct __smartcard_context
: *#include <fsl_smartcard.h>* Runtime state of the Smart card driver.

Public Members

void *base
Smart card module base address

smartcard_direction_t direction
Direction of communication. (RX/TX)

uint8_t *xBuff
The buffer of data being transferred.

volatile size_t xSize
The number of bytes to be transferred.

volatile bool xIsBusy
True if there is an active transfer.

uint8_t txFifoEntryCount
Number of data word entries in transmit FIFO.

uint8_t rxFifoThreshold
The max value of the receiver FIFO threshold.

smartcard_interface_callback_t interfaceCallback
Callback to invoke after interface IC raised interrupt.

smartcard_transfer_callback_t transferCallback
Callback to invoke after transfer event occur.

void *interfaceCallbackParam
Interface callback parameter pointer.

void *transferCallbackParam
Transfer callback parameter pointer.

smartcard_time_delay_t timeDelay
Function which handles time delay defined by user or RTOS.

smartcard_reset_type_t resetType
Indicates whether a Cold reset or Warm reset was requested.

smartcard_transport_type_t tType
Indicates current transfer protocol (T0 or T1)

volatile *smartcard_transfer_state_t* transferState
Indicates the current transfer state

smartcard_timers_state_t timersState
Indicates the state of different protocol timers used in driver

smartcard_card_params_t cardParams
Smart card parameters(ATR and current) and interface slots states(ATR and current)

uint8_t IFSD
Indicates the terminal IFSD

smartcard_parity_type_t parity
Indicates current parity even/odd

volatile bool rxtCrossed
Indicates whether RXT thresholds has been crossed

volatile bool txtCrossed
Indicates whether TXT thresholds has been crossed

volatile bool wtxRequested
Indicates whether WTX has been requested or not

volatile bool parityError
Indicates whether a parity error has been detected

uint8_t statusBytes[2]
Used to store Status bytes SW1, SW2 of the last executed card command response

smartcard_interface_config_t interfaceConfig
Smart card interface configuration structure

bool abortTransfer
Used to abort transfer.

2.107 Smart Card EMVSIM Driver

void SMARTCARD_EMVSIM_GetDefaultConfig(*smartcard_card_params_t* *cardParams)
Fills in the smartcard_card_params structure with default values according to the EMV 4.3 specification.

Parameters

- cardParams – The configuration structure of type smartcard_interface_config_t. Function fill in members: Fi = 372; Di = 1; currentD = 1; WI = 0x0A; GTN = 0x00; with default values.

status_t SMARTCARD_EMVSIM_Init(EMVSIM_Type *base, *smartcard_context_t* *context, uint32_t srcClock_Hz)
Initializes an EMVSIM peripheral for the Smart card/ISO-7816 operation.

This function un-gates the EMVSIM clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core and, initializes the driver context.

Parameters

- base – The EMVSIM peripheral base address.
- context – A pointer to the smart card driver context structure.
- srcClock_Hz – Smart card clock generation module source clock.

Returns
An error code or kStatus_SMARTCARD_Success.

void SMARTCARD_EMVSIM_Deinit(EMVSIM_Type *base)
This function disables the EMVSIM interrupts, disables the transmitter and receiver, flushes the FIFOs, and gates EMVSIM clock in SIM.

Parameters

- base – The EMVSIM module base address.

int32_t SMARTCARD_EMVSIM_GetTransferRemainingBytes(EMVSIM_Type *base, *smartcard_context_t* *context)
Returns whether the previous EMVSIM transfer has finished.

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred.

Parameters

- base – The EMVSIM module base address.
- context – A pointer to a smart card driver context structure.

Returns
The number of bytes not transferred.

status_t SMARTCARD_EMVSIM_AbortTransfer(EMVSIM_Type *base, *smartcard_context_t* *context)

Terminates an asynchronous EMVSIM transfer early.

During an async EMVSIM transfer, the user can terminate the transfer early if the transfer is still in progress.

Parameters

- base – The EMVSIM peripheral address.
- context – A pointer to a smart card driver context structure.

Return values

- kStatus_SMARTCARD_Success – The transmit abort was successful.
- kStatus_SMARTCARD_NoTransmitInProgress – No transmission is currently in progress.

status_t SMARTCARD_EMVSIM_TransferNonBlocking(EMVSIM_Type *base, *smartcard_context_t* *context, *smartcard_xfer_t* *xfer)

Transfer data using interrupts.

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling the non-blocking (asynchronous) transfer function, the application must get the transfer status to check if the transmit is completed or not.

Parameters

- base – The EMVSIM peripheral base address.
- context – A pointer to a smart card driver context structure.
- xfer – A pointer to the smart card transfer structure where the linked buffers and sizes are stored.

Returns
An error code or kStatus_SMARTCARD_Success.

status_t SMARTCARD_EMVSIM_Control(EMVSIM_Type *base, *smartcard_context_t* *context, *smartcard_control_t* control, uint32_t param)

Controls the EMVSIM module per different user request.

return kStatus_SMARTCARD_Success in success. return kStatus_SMARTCARD_OtherError in case of error.

Parameters

- base – The EMVSIM peripheral base address.
- context – A pointer to a smart card driver context structure.
- control – Control type.
- param – Integer value of specific to control command.

void SMARTCARD_EMVSIM_IRQHandler(EMVSIM_Type *base, *smartcard_context_t* *context)

Handles EMVSIM module interrupts.

Parameters

- base – The EMVSIM peripheral base address.
- context – A pointer to a smart card driver context structure.

enum _emvsim_gpc_clock_select

General Purpose Counter clock selections.

Values:

enumerator kEMVSIM_GPCClockDisable

Disabled

enumerator kEMVSIM_GPCCardClock

Card clock

enumerator kEMVSIM_GPCRxClock

Receive clock

enumerator kEMVSIM_GPCTxClock

Transmit ETU clock

enum _presence_detect_edge

EMVSIM card presence detection edge control.

Values:

enumerator kEMVSIM_DetectOnFallingEdge

Presence detected on the falling edge

enumerator kEMVSIM_DetectOnRisingEdge

Presence detected on the rising edge

enum _presence_detect_status

EMVSIM card presence detection status.

Values:

enumerator kEMVSIM_DetectPinIsLow

Presence detected pin is logic low

enumerator kEMVSIM_DetectPinIsHigh

Presence detected pin is logic high

typedef enum *_emvsim_gpc_clock_select* emvsim_gpc_clock_select_t

General Purpose Counter clock selections.

typedef enum *_presence_detect_edge* emvsim_presence_detect_edge_t

EMVSIM card presence detection edge control.

typedef enum *_presence_detect_status* emvsim_presence_detect_status_t

EMVSIM card presence detection status.

SMARTCARD_EMV_RX_NACK_THRESHOLD

EMV RX NACK interrupt generation threshold.

SMARTCARD_EMV_TX_NACK_THRESHOLD

EMV TX NACK interrupt generation threshold.

SMARTCARD_WWT_ADJUSTMENT

Smart card Word Wait Timer adjustment value.

SMARTCARD_CWT_ADJUSTMENT
Smart card Character Wait Timer adjustment value.

2.108 Smart Card PHY Driver

void SMARTCARD_PHY_GetDefaultConfig(*smartcard_interface_config_t* *config)
Fills in the configuration structure with default values.

Parameters

- config – The Smart card user configuration structure which contains configuration structure of type smartcard_interface_config_t. Function fill in members: clockToResetDelay = 42000, vcc = kSmartcardVoltageClassB3_3V, with default values.

status_t SMARTCARD_PHY_Init(void *base, *smartcard_interface_config_t* const *config, uint32_t srcClock_Hz)
Initializes a Smart card interface instance.

Parameters

- base – The Smart card peripheral base address.
- config – The user configuration structure of type smartcard_interface_config_t. Call the function SMARTCARD_PHY_GetDefaultConfig() to fill the configuration structure.
- srcClock_Hz – Smart card clock generation module source clock.

Return values
kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

void SMARTCARD_PHY_Deinit(void *base, *smartcard_interface_config_t* const *config)
De-initializes a Smart card interface, stops the Smart card clock, and disables the VCC.

Parameters

- base – The Smart card peripheral module base address.
- config – The user configuration structure of type smartcard_interface_config_t.

status_t SMARTCARD_PHY_Activate(void *base, *smartcard_context_t* *context, *smartcard_reset_type_t* resetType)
Activates the Smart card IC.

Parameters

- base – The Smart card peripheral module base address.
- context – A pointer to a Smart card driver context structure.
- resetType – type of reset to be performed, possible values = kSmartcardColdReset, kSmartcardWarmReset

Return values
kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

status_t SMARTCARD_PHY_Deactivate(void *base, *smartcard_context_t* *context)
De-activates the Smart card IC.

Parameters

- base – The Smart card peripheral module base address.
- context – A pointer to a Smart card driver context structure.

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

status_t SMARTCARD_PHY_Control(void *base, *smartcard_context_t* *context, *smartcard_interface_control_t* control, uint32_t param)

Controls the Smart card interface IC.

Parameters

- base – The Smart card peripheral module base address.
- context – A pointer to a Smart card driver context structure.
- control – A interface command type.
- param – Integer value specific to control type

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

SMARTCARD_ATR_DURATION_ADJUSTMENT

Smart card definition which specifies the adjustment number of clock cycles during which an ATR string has to be received.

SMARTCARD_INIT_DELAY_CLOCK_CYCLES_ADJUSTMENT

Smart card definition which specifies the adjustment number of clock cycles until an initial 'TS' character has to be received.

2.109 Smart Card PHY EMVSIM Driver

2.110 Smart Card PHY TDA8035 Driver

2.111 Smart Card PHY USIM W

2.112 Smart Card USIM Driver

void SMARTCARD_USIM_GetDefaultConfig(*smartcard_card_params_t* *cardParams)

Fills in the smartcard_card_params structure with default values according to the EMV 4.3 specification.

Parameters

- cardParams – The configuration structure of type smartcard_interface_config_t. Function fill in members: Fi = 372; Di = 1; currentD = 1; WI = 0x0A; GTN = 0x00; with default values.

status_t SMARTCARD_USIM_Init(USIM_Type *base, *smartcard_context_t* *context, uint32_t srcClock_Hz)

Initializes an USIM peripheral for the Smart card/ISO-7816 operation.

This function un-gates the USIM clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core and, initializes the driver context.

Parameters

- base – The USIM peripheral base address.
- context – A pointer to the smart card driver context structure.
- srcClock_Hz – Smart card clock generation module source clock.

Returns
An error code or kStatus_SMARTCARD_Success.

void SMARTCARD_USIM_Deinit(USIM_Type *base)

This function disables the USIM interrupts, disables the transmitter and receiver, flushes the FIFOs, and gates USIM clock in SIM.

Parameters

- base – The USIM module base address.

int32_t SMARTCARD_USIM_GetTransferRemainingBytes(USIM_Type *base, *smartcard_context_t* *context)

Returns whether the previous USIM transfer has finished.

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred.

Parameters

- base – The USIM module base address.
- context – A pointer to a smart card driver context structure.

Returns
The number of bytes not transferred.

status_t SMARTCARD_USIM_AbortTransfer(USIM_Type *base, *smartcard_context_t* *context)

Terminates an asynchronous USIM transfer early.

During an async USIM transfer, the user can terminate the transfer early if the transfer is still in progress.

Parameters

- base – The USIM peripheral address.
- context – A pointer to a smart card driver context structure.

Returns
kStatus_SMARTCARD_Success The transmit abort was successful.

Returns
kStatus_SMARTCARD_NoTransmitInProgress No transmission is currently in progress.

status_t SMARTCARD_USIM_TransferNonBlocking(USIM_Type *base, *smartcard_context_t* *context, *smartcard_xfer_t* *xfer)

Transfer data using interrupts.

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling the non-blocking (asynchronous) transfer function, the application must get the transfer status to check if the transmit is completed or not.

Parameters

- base – The USIM peripheral base address.

- context – A pointer to a smart card driver context structure.
- xfer – A pointer to the smart card transfer structure where the linked buffers and sizes are stored.

Returns
An error code or kStatus_SMARTCARD_Success.

status_t SMARTCARD_USIM_Control(USIM_Type *base, *smartcard_context_t* *context, *smartcard_control_t* control, uint32_t param)

Controls the USIM module per different user request.

Parameters

- base – The USIM peripheral base address.
- context – A pointer to a smart card driver context structure.
- control – Control type.
- param – Integer value of specific to control command.

Returns
kStatus_SMARTCARD_Success in success.

Returns
kStatus_SMARTCARD_OtherError in case of error.

void SMARTCARD_USIM_IRQHandler(USIM_Type *base, *smartcard_context_t* *context)

Handles USIM module interrupts.

Parameters

- base – The USIM peripheral base address.
- context – A pointer to a smart card driver context structure.

void SMARTCARD_USIM_TSExpiryCallback(USIM_Type *base, *smartcard_context_t* *context)

Handles initial TS character timer time-out event.

Parameters

- base – The USIM peripheral base address.
- context – A pointer to a Smart card driver context structure.

void SMARTCARD_USIM_TimerStart(uint32_t time)

Initializes timer with input period, enable interrupt and start counter.

Parameters

- time – The time period.

enum _usim_rx_fifo_trigger_level

USIM Rx FIFO receiver trigger level enumeration.

Values:

enumerator kUSIM_1ByteOrMore

1 byte or more in the RX-FIFO can trigger receiver data ready interrupt.

enumerator kUSIM_4ByteOrMore

4 byte or more in the RX-FIFO can trigger receiver data ready interrupt.

enumerator kUSIM_8ByteOrMore

8 byte or more in the RX-FIFO can trigger receiver data ready interrupt.

enumerator kUSIM_12ByteOrMore

12 byte or more in the RX-FIFO can trigger receiver data ready interrupt.

typedef enum *_usim_rx_fifo_trigger_level* usim_rx_fifo_trigger_level_t
USIM Rx FIFO receiver trigger level enumeration.

SMARTCARD_Control(base, handle, control, param)

SMARTCARD_TransferNonBlocking(base, handle, xfer)

SMARTCARD_Init(base, handle, sourceClockHz)

SMARTCARD_Deinit(base)

SMARTCARD_GetTransferRemainingBytes(base, handle)

SMARTCARD_AbortTransfer(base, handle)

SMARTCARD_EMV_RX_NACK_THRESHOLD
EMV RX NACK interrupt generation threshold.

SMARTCARD_EMV_TX_NACK_THRESHOLD
EMV TX NACK interrupt generation threshold.

SMARTCARD_T0_CWT_ADJUSTMENT
Smart card T0 Character Wait Timer adjustment value.

SMARTCARD_T1_CWT_ADJUSTMENT
Smart card T1 Character Wait Timer adjustment value.

SMARTCARD_T0_BWT_ADJUSTMENT
Smart card T0 Block Wait Timer adjustment value.

SMARTCARD_T1_BWT_ADJUSTMENT
Smart card T1 Block Wait Timer adjustment value.

SMARTCARD_MAX_RX_TRIGGER_LEVEL
Rx FIFO max receive trigger level.

USIM_FIND_RX_FIFO_TRIGGER_LEVEL(x)
USIM Find max Rx FIFO receiver trigger level according to bytes numbers.

2.113 SMARTDMA: SMART DMA Driver

typedef void (*smartdma_callback_t)(void *param)
Callback function prototype for the smartdma driver.

void SMARTDMA_Init(uint32_t apiMemAddr, const void *firmware, uint32_t firmwareSizeByte)
Initialize the SMARTDMA.

Deprecated:
Do not use this function. It has been superceded by SMARTDMA_InitWithoutFirmware and SMARTDMA_InstallFirmware.

Parameters

- apiMemAddr – The address firmware will be copied to.
- firmware – The firmware to use.
- firmwareSizeByte – Size of firmware.

void SMARTDMA_InitWithoutFirmware(void)

Initialize the SMARTDMA.

This function is similar with SMARTDMA_Init, the difference is this function does not install the firmware, the firmware could be installed using SMARTDMA_InstallFirmware.

void SMARTDMA_InstallFirmware(uint32_t apiMemAddr, const void *firmware, uint32_t firmwareSizeByte)

Install the firmware.

Note: Only call this function when SMARTDMA is not busy.

Parameters

- apiMemAddr – The address firmware will be copied to.
- firmware – The firmware to use.
- firmwareSizeByte – Size of firmware.

void SMARTDMA_InstallCallback(*smartdma_callback_t* callback, void *param)

Install the complete callback function.

Note: Only call this function when SMARTDMA is not busy.

Parameters

- callback – The callback called when smartdma program finished.
- param – Parameter for the callback.

void SMARTDMA_Boot(uint32_t apiIndex, void *pParam, uint8_t mask)

Boot the SMARTDMA to run program.

Note: Only call this function when SMARTDMA is not busy.

Note: The memory *pParam shall not be freed before the SMARTDMA function finished.

Parameters

- apiIndex – Index of the API to call.
- pParam – Pointer to the parameter allocated by caller.
- mask – Value set to register SMARTDMA->ARM2EZH[0:1].

void SMARTDMA_Boot1(uint32_t apiIndex, const smartdma_param_t *pParam, uint8_t mask)

Copy SMARTDMA params and Boot to run program.

This function is similar with SMARTDMA_Boot, the only difference is, this function copies the *pParam to a local variable, upper layer can free the pParam's memory before the SMARTDMA execution finished, for example, upper layer can define the param as a local variable.

Note: Only call this function when SMARTDMA is not busy.

Parameters

- apiIndex – Index of the API to call.
- pParam – Pointer to the parameter.
- mask – Value set to SMARTDMA_ARM2SMARTDMA[0:1].

void SMARTDMA_Deinit(void)
Deinitialize the SMARTDMA.

void SMARTDMA_Reset(void)
Reset the SMARTDMA.

void SMARTDMA_HandleIRQ(void)
SMARTDMA IRQ.

void SMARTDMA_SetExternalFlag(uint8_t flag)
SMARTDMA set EX flag.

void SMARTDMA_AccessShareRAM(uint8_t flag)
SMARTDMA access RAM.

FSL_SMARTDMA_DRIVER_VERSION
SMARTDMA driver version.

2.114 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.

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

2.115 Digital Tamper

2.116 TDET

status_t TDET_Init(DIGTMP_Type *base)

Initialize TDET.

This function initializes TDET.

Parameters

- base – TDET peripheral base address

Returns

Status of the init operation

void TDET_Deinit(DIGTMP_Type *base)

Deinitialize TDET.

This function disables glitch filters and active tampers This function disables the TDET clock and prescaler in TDET Control Register.

Parameters

- base – TDET peripheral base address

void TDET_GetDefaultConfig(DIGTMP_Type *base, *tdet_config_t* *defaultConfig)

Gets default values for the TDET Control Register.

This function fills the given structure with default values for the TDET Control Register. The default values are:

```
defaultConfig->innerClockAndPrescalerEnable = true
defaultConfig->tamperForceSystemResetEnable = false
defaultConfig->updateMode = kTDET_StatusLockWithTamper
defaultConfig->clockSourceActiveTamper0 = kTDET_ClockType1Hz
defaultConfig->clockSourceActiveTamper1 = kTDET_ClockType1Hz
defaultConfig->prescaler = 0
```

Parameters

- base – TDET peripheral base address

- defaultConfig – **[out]** Pointer to structure to be filled with default parameters

status_t TDET_SetConfig(DIGTMP_Type *base, const *tdet_config_t* *config)

Writes to the TDET Control Register.

This function writes the given structure to the TDET Control Register.

Parameters

- base – TDET peripheral base address
- config – Pointer to structure with TDET peripheral configuration parameters

Returns

kStatus_Fail when writing to TDET Control Register is not allowed

Returns

kStatus_Success when operation completes successfully

status_t TDET_SoftwareReset(DIGTMP_Type *base)

Software reset.

This function resets all TDET registers. The CR[SWR] itself is not affected; it is reset by VBAT POR only.

Parameters

- base – TDET peripheral base address

Returns

kStatus_Fail when writing to TDET Control Register is not allowed

Returns

kStatus_Success when operation completes successfully

status_t TDET_ActiveTamperSetConfig(DIGTMP_Type *base, const *tdet_active_tamper_config_t* *activeTamperConfig, uint32_t activeTamperRegisterSelect)

Writes to the active tamper register(s).

This function writes per active tamper register parameters to active tamper register(s).

Parameters

- base – TDET peripheral base address
- activeTamperConfig – Pointer to structure with active tamper register parameters
- activeTamperRegisterSelect – Bit mask for active tamper registers to be configured. The passed value is combination of tdet_active_tamper_register_t values (OR'ed).

Returns

kStatus_Fail when writing to TDET Active Tamper Register(s) is not allowed

Returns

kStatus_Success when operation completes successfully

void TDET_PinGetDefaultConfig(DIGTMP_Type *base, *tdet_pin_config_t* *pinConfig)

Gets default values for tamper pin configuration.

This function fills the give structure with default values for the tamper pin and glitch filter configuration. The default values are: code pinConfig->pinPolarity = kTDET_TamperPinPolarityExpectNormal; pinConfig->pinDirection = kTDET_TamperPinDirectionIn; pinConfig->tamperPullEnable = false; pinConfig->tamperPinSampleFrequency = kTDET_GlitchFilterSamplingEveryCycle8;

pinConfig->tamperPinSampleWidth = kTDET_GlitchFilterSampleDisable; pinConfig->glitchFilterEnable = false; pinConfig->glitchFilterPrescaler = kTDET_GlitchFilterClock512Hz; pinConfig->glitchFilterWidth = 0; pinConfig->tamperPinExpected = kTDET_GlitchFilterExpectedLogicZero; pinConfig->tamperPullSelect = kTDET_GlitchFilterPullTypeAssert; endcode

Parameters

- base – TDET peripheral base address
- pinConfig – **[out]** Pointer to structure to be filled with tamper pins default parameters

status_t TDET_PinSetConfig(DIGTMP_Type *base, const *tdet_pin_config_t* *pinConfig, uint32_t pinSelect)

Writes the tamper pin configuration.

This function writes per pin parameters to tamper pin and glitch filter configuration registers.

Parameters

- base – TDET peripheral base address
- pinConfig – Pointer to structure with tamper pin and glitch filter configuration parameters
- pinSelect – Bit mask for tamper pins to be configured. The passed value is combination of enum _tdet_tamper_pin (tdet_tamper_pin_t) values (OR'ed).

Returns

kStatus_Fail when writing to TDET Pin Direction, Pin Polarity or Glitch Filter Register(s) is not allowed

Returns

kStatus_Success when operation completes successfully

status_t TDET_GetStatusFlags(DIGTMP_Type *base, uint32_t *result)

Reads the Status Register.

This function reads flag bits from TDET Status Register.

Parameters

- base – TDET peripheral base address
- result – **[out]** Pointer to uint32_t where to write Status Register read value. Use tdet_status_flag_t to decode individual flags.

Returns

kStatus_Fail when Status Register reading is not allowed

Returns

kStatus_Success when result is written with the Status Register read value

status_t TDET_ClearStatusFlags(DIGTMP_Type *base, uint32_t mask)

Writes to the Status Register.

This function clears specified flag bits in TDET Status Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the flag bits to be cleared. Use tdet_status_flag_t to encode flags.

Returns
kStatus_Fail when Status Register writing is not allowed

Returns
kStatus_Success when mask is written to the Status Register

status_t TDET_EnableInterrupts(DIGTMP_Type *base, uint32_t mask)

Writes to the Interrupt Enable Register.

This function sets specified interrupt enable bits in TDET Interrupt Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the interrupt enable bits to be set.

Returns
kStatus_Fail when Interrupt Enable Register writing is not allowed

Returns
kStatus_Success when mask is written to the Interrupt Enable Register

status_t TDET_DisableInterrupts(DIGTMP_Type *base, uint32_t mask)

Writes to the Interrupt Enable Register.

This function clears specified interrupt enable bits in TDET Interrupt Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the interrupt enable bits to be cleared.

Returns
kStatus_Fail when Interrupt Enable Register writing is not allowed

Returns
kStatus_Success when specified bits are cleared in the Interrupt Enable Register

status_t TDET_EnableTampers(DIGTMP_Type *base, uint32_t mask)

Writes to the Tamper Enable Register.

This function sets specified tamper enable bits in TDET Tamper Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the tamper enable bits to be set.

Returns
kStatus_Fail when Tamper Enable Register writing is not allowed

Returns
kStatus_Success when mask is written to the Tamper Enable Register

status_t TDET_DisableTampers(DIGTMP_Type *base, uint32_t mask)

Writes to the Tamper Enable Register.

This function clears specified tamper enable bits in TDET Tamper Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the tamper enable bits to be cleared.

Returns
kStatus_Fail when Tamper Enable Register writing is not allowed

Returns
kStatus_Success when specified bits are cleared in the Tamper Enable Register

status_t TDET_ForceTamper(DIGTMP_Type *base)

Writes to the Tamper Seconds Register.

This function writes to TDET Tamper Seconds Register. This causes Status Register DTF flag to be set (TDET tampering detected).

Parameters

- base – TDET peripheral base address

Returns
kStatus_Fail when Tamper Seconds Register writing is not allowed

Returns
kStatus_Success when Tamper Seconds Register is written

status_t TDET_GetTamperTimeSeconds(DIGTMP_Type *base, uint32_t *tamperTimeSeconds)

Reads the Tamper Seconds Register.

This function reads TDET Tamper Seconds Register. The read value returns the time in seconds at which the Status Register DTF flag was set.

Parameters

- base – TDET peripheral base address
- tamperTimeSeconds – Time in seconds at which the tamper detection SR[DTF] flag was set.

Returns
kStatus_Fail when Tamper Seconds Register reading is not allowed

Returns
kStatus_Success when Tamper Seconds Register is read

void TDET_LockRegisters(DIGTMP_Type *base, uint32_t mask)

Writes to the TDET Lock Register.

This function clears specified lock bits in the TDET Lock Register. When a lock bit is clear, a write to corresponding TDET Register is ignored. Once cleared, these bits can only be set by VBAT POR or software reset.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the lock bits to be cleared. Use tdet_register_t values to encode (OR'ed) which TDET Registers shall be locked.

FSL_TDET_DRIVER_VERSION

Defines TDET driver version 2.3.0.

Change log:

- Version 2.3.0
  - Added enum for TIF10.
- Version 2.2.0
  - Added support for chips without active tamper pins.
- Version 2.1.1
  - Added clearing SR_TAF and SR_DTF into TDET_Init().
  - Fix typo in kTDET_ClockType64Hz comment

- Version 2.1.0
  - Added setting of disabling prescaler on tamper event into TDET_SetConfig() and TDET_GetDefaultConfig functions.
- Version 2.0.0
  - Initial version

enum _tdet_update_mode

TDET Update Mode.

These constants allow TDET interrupts to be cleared if no tampering has been detected, while still preventing the TDET Tamper Flag (SR[DTF]) from being cleared once it is set.

Values:

enumerator kTDET_StatusLockNormal

TDET Status Register cannot be written when the Status Register Lock bit within the Lock Register (LR[SRL]) is clear

enumerator kTDET_StatusLockWithTamper

TDET Status Register cannot be written when the Status Register Lock bit within the Lock Register (LR[SRL]) is clear and TDET Tamper Flag (SR[DTF]) is set

enum _tdet_active_tamper_clock

TDET Active Tamper Clock Source.

These constants define the clock source for Active Tamper Shift Register to configure in a TDET base.

Values:

enumerator kTDET_ClockType1Hz

clocked by 1 Hz prescaler clock

enumerator kTDET_ClockType64Hz

clocked by 64 Hz prescaler clock

enum _tdet_pin_polarity

TDET Tamper Pin Polarity.

These constants define tamper pin polarity to configure in a TDET base.

Values:

enumerator kTDET_TamperPinPolarityExpectNormal

Tamper pin expected value is not inverted

enumerator kTDET_TamperPinPolarityExpectInverted

Tamper pin expected value is inverted

enum _tdet_pin_direction

TDET Tamper Pin Direction.

These constants define tamper pin direction to configure in a TDET base.

Values:

enumerator kTDET_TamperPinDirectionIn

Tamper pins configured as input

enumerator kTDET_TamperPinDirectionOut

Tamper pins configured as output, drives inverse of expected value

enum _tdet_glitch_filter_sample_freq

TDET Glitch Filter Tamper Pin Sample Frequency.

These constants define tamper pin glitch filter sample frequency to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterSamplingEveryCycle8

Sample once every 8 cycles

enumerator kTDET_GlitchFilterSamplingEveryCycle32

Sample once every 32 cycles

enumerator kTDET_GlitchFilterSamplingEveryCycle128

Sample once every 128 cycles

enumerator kTDET_GlitchFilterSamplingEveryCycle512

Sample once every 512 cycles

enum _tdet_glitch_filter_sample_width

TDET Glitch Filter Tamper Pin Sample Width.

These constants define tamper pin glitch filter sample width to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterSampleDisable

Sampling disabled

enumerator kTDET_GlitchFilterSampleCycle2

Sample width pull enable/input buffer enable=2 cycles/1 cycle

enumerator kTDET_GlitchFilterSampleCycle4

Sample width pull enable/input buffer enable=4 cycles/2 cycles

enumerator kTDET_GlitchFilterSampleCycle8

Sample width pull enable/input buffer enable=8 cycles/4 cycles

enum _tdet_glitch_filter_prescaler

TDET Glitch Filter Tamper Pin Clock Source.

These constants define tamper pin glitch filter clock source to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterClock512Hz

Glitch Filter on tamper pin is clocked by the 512 Hz prescaler clock

enumerator kTDET_GlitchFilterClock32768Hz

Glitch Filter on tamper pin is clocked by the 32768 Hz prescaler clock

enum _tdet_glitch_filter_expected

TDET Glitch Filter Tamper Pin Expected Value.

These constants define tamper pin glitch filter expected value to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterExpectedLogicZero

Expected value is logic zero

enumerator kTDET_GlitchFilterExpectedActTamperOut0

Expected value is active tamper 0 output

enumerator kTDET_GlitchFilterExpectedActTamperOut1
Expected value is active tamper 1 output

enumerator kTDET_GlitchFilterExpectedActTamperOutXOR
Expected value is active tamper 0 output XORed with active tamper 1 output

enum _tdet_glitch_filter_pull
TDET Glitch Filter Tamper Pull Select.

These constants define tamper pin glitch filter pull direction to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterPullTypeAssert
Tamper pin pull direction always asserts the tamper pin.

enumerator kTDET_GlitchFilterPullTypeNegate
Tamper pin pull direction always negates the tamper pin.

enum _tdet_external_tamper_pin
List of TDET external tampers.

Values:

enumerator kTDET_ExternalTamper0

enumerator kTDET_ExternalTamper1

enumerator kTDET_ExternalTamper2

enumerator kTDET_ExternalTamper3

enumerator kTDET_ExternalTamper4

enumerator kTDET_ExternalTamper5

enumerator kTDET_ExternalTamper6

enumerator kTDET_ExternalTamper7

enum _tdet_active_tamper_register
TDET Active Tamper Register Select.

These constants are used to define activeTamperRegisterSelect argument to be used with TDET_ActiveTamperConfigure().

Values:

enumerator kTDET_ActiveTamperRegister0

enumerator kTDET_ActiveTamperRegister1

enum _tdet_status_flag
TDET Status Register flags.

This provides constants for the TDET Status Register.

Values:

enumerator kTDET_StatusTamperFlag
TDET Digital Tamper Flag

enumerator kTDET_StatusTamperAcknowledgeFlag
TDET Tamper Acknowledge Flag

enumerator kTDET_TIF0
TDET Tamper input 1

enumerator kTDET_TIF1
TDET Tamper input 1

enumerator kTDET_TIF2
TDET Tamper input 2

enumerator kTDET_TIF3
TDET Tamper input 3

enumerator kTDET_TIF4
TDET Tamper input 4

enumerator kTDET_TIF5
TDET Tamper input 5

enumerator kTDET_TIF6
TDET Tamper input 6

enumerator kTDET_TIF7
TDET Tamper input 7

enumerator kTDET_TIF8
TDET Tamper input 8

enumerator kTDET_TIF9
TDET Tamper input 9

enumerator kTDET_StatusTamperPinTamper0
TDET Tamper Pin 0 Tamper detected

enumerator kTDET_StatusTamperPinTamper1
TDET Tamper Pin 1 Tamper detected

enumerator kTDET_StatusTamperPinTamper2
TDET Tamper Pin 2 Tamper detected

enumerator kTDET_StatusTamperPinTamper3
TDET Tamper Pin 3 Tamper detected

enumerator kTDET_StatusTamperPinTamper4
TDET Tamper Pin 4 Tamper detected

enumerator kTDET_StatusTamperPinTamper5
TDET Tamper Pin 5 Tamper detected

enumerator kTDET_StatusAll
Mask for all of the TDET Status Register bits

enum _tdet_interrupt
TDET Interrupt Enable Register.

This provides constants for the TDET Interrupt Enable Register.

Values:

enumerator kTDET_InterruptTamper
TDET Digital Tamper Interrupt

enumerator kTDET_InterruptTIF0
TDET TIF0 Interrupt

enumerator kTDET_InterruptTIF1
TDET TIF1 Interrupt

enumerator kTDET_InterruptTIF2
TDET TIF2 Interrupt

enumerator kTDET_InterruptTIF3
TDET TIF3 Interrupt

enumerator kTDET_InterruptTIF4
TDET TIF4 Interrupt

enumerator kTDET_InterruptTIF5
TDET TIF5 Interrupt

enumerator kTDET_InterruptTIF6
TDET TIF6 Interrupt

enumerator kTDET_InterruptTIF7
TDET TIF7 Interrupt

enumerator kTDET_InterruptTIF8
TDET TIF8 Interrupt

enumerator kTDET_InterruptTIF9
TDET TIF9 Interrupt

enumerator kTDET_InterruptTamperPinTamper0
TDET Tamper Pin Tamper 0 Interrupt

enumerator kTDET_InterruptTamperPinTamper1
TDET Tamper Pin Tamper 1 Interrupt

enumerator kTDET_InterruptTamperPinTamper2
TDET Tamper Pin Tamper 2 Interrupt

enumerator kTDET_InterruptTamperPinTamper3
TDET Tamper Pin Tamper 3 Interrupt

enumerator kTDET_InterruptTamperPinTamper4
TDET Tamper Pin Tamper 4 Interrupt

enumerator kTDET_InterruptTamperPinTamper5
TDET Tamper Pin Tamper 5 Interrupt

enumerator kTDET_InterruptTamperPinTamper_All
TDET All Tamper Pins Interrupt

enumerator kTDET_InterruptAll
Mask to select all TDET Interrupt Enable Register bits

enum _tdet_tamper
TDET Tamper Enable Register.

This provides constants for the TDET Tamper Enable Register.

Values:

enumerator kTDET_TamperTIF0
TIF0 Tamper Enable

enumerator kTDET_TamperTIF1
TIF1 Tamper Enable

enumerator kTDET_TamperTIF2
TIF2 Tamper Enable

enumerator kTDET_TamperTIF3
TIF3 Tamper Enable

enumerator kTDET_TamperTIF4
TIF4 Tamper Enable

enumerator kTDET_TamperTIF5
TIF5 Tamper Enable

enumerator kTDET_TamperTIF6
TIF6 Tamper Enable

enumerator kTDET_TamperTIF7
TIF7 Tamper Enable

enumerator kTDET_TamperTIF8
TIF8 Tamper Enable

enumerator kTDET_TamperTIF9
TIF9 Tamper Enable

enumerator kTDET_TamperTamperPin0
Tamper Pin 0 Tamper Enable

enumerator kTDET_TamperTamperPin1
Tamper Pin 1 Tamper Enable

enumerator kTDET_TamperTamperPin2
Tamper Pin 2 Tamper Enable

enumerator kTDET_TamperTamperPin3
Tamper Pin 3 Tamper Enable

enumerator kTDET_TamperTamperPin4
Tamper Pin 4 Tamper Enable

enumerator kTDET_TamperTamperPin5
Tamper Pin 5 Tamper Enable

enumerator kTDET_TamperTamperPinAll
All Tamper Pin Tamper Enable

enumerator kTDET_TamperAll
Mask to select all Tamper Enable Register bits

enum _tdet_register
TDET Registers.

This provides constants to encode a mask for the TDET Registers.

Values:

enumerator kTDET_NoRegister
No Register

enumerator kTDET_Control
Control Register

enumerator kTDET_Status
Status Register

enumerator kTDET_Lock
Lock Register

enumerator kTDET_InterruptEnable
Interrupt Enable Register

enumerator kTDET_TamperSeconds
Tamper Seconds Register

enumerator kTDET_TamperEnable
Tamper Enable Register

enumerator kTDET_PinPolarity
Pin Polarity Register

enumerator kTDET_GlitchFilter0
Glitch Filter Register 0

enumerator kTDET_GlitchFilter1
Glitch Filter Register 1

enumerator kTDET_GlitchFilter2
Glitch Filter Register 2

enumerator kTDET_GlitchFilter3
Glitch Filter Register 3

enumerator kTDET_GlitchFilter4
Glitch Filter Register 4

enumerator kTDET_GlitchFilter5
Glitch Filter Register 5

enumerator kTDET_PinConfigurationRegisters
Mask to select all TDET Pin Configuration Registers

enumerator kTDET_AllRegisters
Mask to select all TDET Registers

typedef enum *_tdet_update_mode* tdet_update_mode_t
TDET Update Mode.

These constants allow TDET interrupts to be cleared if no tampering has been detected, while still preventing the TDET Tamper Flag (SR[DTF]) from being cleared once it is set.

typedef enum *_tdet_active_tamper_clock* tdet_active_tamper_clock_t
TDET Active Tamper Clock Source.

These constants define the clock source for Active Tamper Shift Register to configure in a TDET base.

typedef struct *_tdet_config* tdet_config_t
TDET Control Register.

This structure defines values for TDET Control Register.

typedef enum *_tdet_pin_polarity* tdet_pin_polarity_t
TDET Tamper Pin Polarity.

These constants define tamper pin polarity to configure in a TDET base.

typedef enum *_tdet_pin_direction* tdet_pin_direction_t
TDET Tamper Pin Direction.

These constants define tamper pin direction to configure in a TDET base.

typedef enum *_tdet_glitch_filter_sample_freq* tdet_glitch_filter_sample_freq_t

TDET Glitch Filter Tamper Pin Sample Frequency.

These constants define tamper pin glitch filter sample frequency to configure in a TDET base.

typedef enum *_tdet_glitch_filter_sample_width* tdet_glitch_filter_sample_width_t

TDET Glitch Filter Tamper Pin Sample Width.

These constants define tamper pin glitch filter sample width to configure in a TDET base.

typedef enum *_tdet_glitch_filter_prescaler* tdet_glitch_filter_prescaler_t

TDET Glitch Filter Tamper Pin Clock Source.

These constants define tamper pin glitch filter clock source to configure in a TDET base.

typedef enum *_tdet_glitch_filter_expected* tdet_glitch_filter_expected_t

TDET Glitch Filter Tamper Pin Expected Value.

These constants define tamper pin glitch filter expected value to configure in a TDET base.

typedef enum *_tdet_glitch_filter_pull* tdet_glitch_filter_pull_t

TDET Glitch Filter Tamper Pull Select.

These constants define tamper pin glitch filter pull direction to configure in a TDET base.

typedef struct *_tdet_pin_config* tdet_pin_config_t

TDET Tamper Pin configuration registers.

This structure defines values for TDET Pin Direction, Pin Polarity, and Glitch Filter registers.

typedef enum *_tdet_external_tamper_pin* tdet_external_tamper_pin_t

List of TDET external tampers.

typedef enum *_tdet_active_tamper_register* tdet_active_tamper_register_t

TDET Active Tamper Register Select.

These constants are used to define activeTamperRegisterSelect argument to be used with TDET_ActiveTamperConfigure().

typedef struct *_tdet_active_tamper_config* tdet_active_tamper_config_t

TDET Active Tamper registers.

This structure defines values for TDET Active Tamper Registers.

typedef enum *_tdet_status_flag* tdet_status_flag_t

TDET Status Register flags.

This provides constants for the TDET Status Register.

typedef enum *_tdet_interrupt* tdet_interrupt_t

TDET Interrupt Enable Register.

This provides constants for the TDET Interrupt Enable Register.

typedef enum *_tdet_tamper* tdet_tamper_t

TDET Tamper Enable Register.

This provides constants for the TDET Tamper Enable Register.

typedef enum *_tdet_register* tdet_register_t

TDET Registers.

This provides constants to encode a mask for the TDET Registers.

void VBAT0_DriverIRQHandler(void)

struct __tdet__config

#include <fsl_tdet.h> TDET Control Register.

This structure defines values for TDET Control Register.

Public Members

bool innerClockAndPrescalerEnable

Enable/disable 32768 Hz clock within TDET and the TDET prescaler that generates 512 Hz, 64Hz and 1 Hz prescaler clocks

bool tamperForceSystemResetEnable

Enable/disable assertion of chip reset when tampering is detected

enum *_tdet_update_mode* updateMode

Selects update mode for TDET Status Register

enum *_tdet_active_tamper_clock* clockSourceActiveTamper0

Selects clock source for Active Tamper Shift Register 0

enum *_tdet_active_tamper_clock* clockSourceActiveTamper1

Selects clock source for Active Tamper Shift Register 1

bool disablePrescalerAfterTamper

Allows the 32-KHz clock and prescaler to be automatically disabled after tamper detection and until the system acknowledges the tamper. Disabling the prescaler after detecting a tamper event conserves power and freezes the state of the active tamper outputs and glitch filters. To ensure a clean transition, the prescaler is disabled at the end of a 1 Hz period.

uint32_t prescaler

Initial value for the TDET prescaler 15-bit value.

struct __tdet__pin__config

#include <fsl_tdet.h> TDET Tamper Pin configuration registers.

This structure defines values for TDET Pin Direction, Pin Polarity, and Glitch Filter registers.

Public Members

enum *_tdet_pin_polarity* pinPolarity

Selects tamper pin expected value

enum *_tdet_pin_direction* pinDirection

Selects tamper pin direction

bool tamperPullEnable

Enable/disable pull resistor on the tamper pin

enum *_tdet_glitch_filter_sample_freq* tamperPinSampleFrequency

Selects tamper pin sample frequency

enum *_tdet_glitch_filter_sample_width* tamperPinSampleWidth

Selects tamper pin sample width

bool glitchFilterEnable

Enable/disable glitch filter on the tamper pin

enum *_tdet_glitch_filter_prescaler* glitchFilterPrescaler

Selects the prescaler for the glitch filter on tamper pin

uint8_t glitchFilterWidth
: 6-bit value to configure number of clock edges the input must remain stable for to be passed through the glitch filter for the tamper pin

enum *_tdet_glitch_filter_expected* tamperPinExpected
: Selects tamper pin expected value

enum *_tdet_glitch_filter_pull* tamperPullSelect
: Selects the direction of the tamper pin pull resistor

struct __tdet_active_tamper_config
: *#include <fsl_tdet.h>* TDET Active Tamper registers.

This structure defines values for TDET Active Tamper Registers.

Public Members

uint32_t activeTamperShift
: Active tamper shift register. initialize to non-zero value.

uint32_t activeTamperPolynomial
: Polynomial of the active tamper shift register.

2.117 TRDC: Trusted Resource Domain Controller

void TRDC_Init(TRDC_Type *base)
: Initializes the TRDC module.

This function enables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

void TRDC_Deinit(TRDC_Type *base)
: De-initializes the TRDC module.

This function disables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

static inline uint8_t TRDC_GetCurrentMasterDomainId(TRDC_Type *base)
: Gets the domain ID of the current bus master.

Parameters

- base – TRDC peripheral base address.

Returns
Domain ID of current bus master.

void TRDC_GetHardwareConfig(TRDC_Type *base, *trdc_hardware_config_t* *config)
: Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

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

static inline void TRDC_SetDacGlobalValid(TRDC_Type *base)

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline void TRDC_LockMasterDomainAssignment(TRDC_Type *base, uint8_t master, uint8_t regNum)

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to trdcx_master_t in processor header file, x is trdc instance.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- assignIndex – Which assignment register to lock.

static inline void TRDC_SetMasterDomainAssignmentValid(TRDC_Type *base, uint8_t master, uint8_t regNum, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- assignIndex – Index for the domain assignment register.
- valid – True to set valid, false to set invalid.

void TRDC_GetDefaultProcessorDomainAssignment(*trdc_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId         = 0U;
assignment->domainIdSelect   = kTRDC_DidMda;
assignment->lock             = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void TRDC_GetDefaultNonProcessorDomainAssignment(*trdc_non_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId             = 0U;
assignment->privilegeAttr        = kTRDC_ForceUser;
assignment->secureAttr       = kTRDC_ForceSecure;
assignment->bypassDomainId       = 0U;
assignment->lock                 = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, uint8_t master, uint8_t regNum, const *trdc_processor_domain_assignment_t* *domainAssignment)

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;

TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to trdc_master_t in processor header file.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- domainAssignment – Pointer to the assignment structure.

void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const *trdc_non_processor_domain_assignment_t* *domainAssignment)

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to trdc_master_t in processor header file.
- domainAssignment – Pointer to the assignment structure.

static inline uint64_t TRDC_GetActiveMasterPidMap(TRDC_Type *base)

Gets the bit map of the bus master(s) that is(are) sourcing a PID register.

This function sets the non-processor master domain assignment as valid.

Parameters

- base – TRDC peripheral base address.

Returns

the bit map of the master(s). Bit 1 sets indicates bus master 1.

void TRDC_SetPid(TRDC_Type *base, uint8_t master, const *trdc_pid_config_t* *pidConfig)

Sets the current Process identifier(PID) for processor core.

Each processor has a corresponding process identifier (PID) which can be used to group tasks into different domains. Secure privileged software saves and restores the PID as part of any context switch. This data structure defines an array of 32-bit values, one per MDA module, that define the PID. Since this register resource is only applicable to processor cores, the data structure is typically sparsely populated. The HWCFG[2-3] registers provide a bitmap of the implemented PIDn registers. This data structure is indexed using the corresponding MDA instance number. Depending on the operating clock domain of each DAC instance, there may be optional information stored in the corresponding PIDm register to properly implement the LK2 = 2 functionality.

Parameters

- base – TRDC peripheral base address.
- master – Which processor master to configure, refer to trdc_master_t in processor header file.
- pidConfig – Pointer to the configuration structure.

void TRDC_GetDefaultIDAUConfig(*trdc_idau_config_t* *idauConfiguration)

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```
config->lockSecureVTOR    = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU     = false;
config->lockNonsecureMPU  = false;
config->lockSAU           = false;
```

Parameters

- domainAssignment – Pointer to the configuration structure.

void TRDC_SetIDAU(TRDC_Type *base, const *trdc_idau_config_t* *idauConfiguration)

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
```

(continues on next page)

(continued from previous page)

```
idauConfiguration.lockNonsecureMPU      = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the configuration structure.

static inline void TRDC_EnableFlashLogicalWindow(TRDC_Type *base, bool enable)

Enables/disables the FLW(flash logical window) function.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

static inline void TRDC_LockFlashLogicalWindow(TRDC_Type *base)

Locks FLW registers. Once locked the registers can 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;
```

(continues on next page)

(continued from previous page)

```
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.

Returns

the region number of the given MRC instance

void TRDC_MrcSetMemoryAccessConfig(TRDC_Type *base, const *trdc_memory_access_control_config_t* *config, uint8_t mrcIdx, uint8_t regIdx)

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mrcIdx – MRC index.
- regIdx – Register number.

void TRDC_MrcEnableDomainNseUpdate(TRDC_Type *base, uint8_t mrcIdx, uint16_t domianMask, bool enable)

Enables the update of the selected domians.

After the domians' update are enabled, their regions' NSE bits can be set or clear.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domianMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

void TRDC_MrcRegionNseSet(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

void TRDC_MrcRegionNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to clear.

void TRDC_MrcDomainNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domians whose NSE bits to clear.

void TRDC_MrcSetRegionDescriptorConfig(TRDC_Type *base, const *trdc_mrc_region_descriptor_config_t* *config)

Sets the configuration for one of the region descriptor per domain per MRC instnce.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to region descriptor configuration structure.

static inline void TRDC_SetMbcGlobalValid(TRDC_Type *base)

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

void TRDC_GetMbcHardwareConfig(TRDC_Type *base, *trdc_slave_memory_hardware_config_t* *config, uint8_t mbcIdx, uint8_t slvIdx)

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

void TRDC_MbcSetNseUpdateConfig(TRDC_Type *base, const *trdc_mbc_nse_update_config_t* *config, uint8_t mbcIdx)

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be updateby NSE set/clear.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

void TRDC_MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

void TRDC_MbcWordNseClear(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

void TRDC_MbcNseClearAll(TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slave)

Clears all configuration words' NSE bits of the selected domain and memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

void TRDC_MbcSetMemoryAccessConfig(TRDC_Type *base, const *trdc_memory_access_control_config_t* *config, uint8_t mbcIdx, uint8_t rgdIdx)

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

void TRDC_MbcSetMemoryBlockConfig(TRDC_Type *base, const *trdc_mbc_memory_block_config_t* *config)

Sets the configuration for one of the memory block per domain per MBC instnce.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

enum _trdc_did_sel

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

Values:

enumerator kTRDC_DidMda

Use MDAn[2:0] as DID.

enumerator kTRDC_DidInput

Use the input DID (DID_in) as DID.

enumerator kTRDC_DidMdaAndInput

Use MDAn[2] concatenated with DID_in[1:0] as DID.

enumerator kTRDC_DidReserved

Reserved.

enum _trdc_secure_attr

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

Values:

enumerator kTRDC_ForceSecure

Force the bus attribute for this master to secure.

enumerator kTRDC_ForceNonSecure

Force the bus attribute for this master to non-secure.

enumerator kTRDC_MasterSecure

Use the bus master's secure/nonsecure attribute directly.

enumerator kTRDC_MasterSecure1

Use the bus master's secure/nonsecure attribute directly.

enum _trdc_pid_domain_hit_config

The configuration of domain hit evaluation of PID.

Values:

enumerator kTRDC_pidDomainHitNone0

No PID is included in the domain hit evaluation.

enumerator kTRDC_pidDomainHitNone1

No PID is included in the domain hit evaluation.

enumerator kTRDC_pidDomainHitInclusive

The PID is included in the domain hit evaluation when (PID & ~PIDM).

enumerator kTRDC_pidDomainHitExclusive

The PID is included in the domain hit evaluation when ~(PID & ~PIDM).

enum _trdc_privilege_attr

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

Values:

enumerator kTRDC_ForceUser

Force the bus attribute for this master to user.

enumerator kTRDC_ForcePrivilege
Force the bus attribute for this master to privileged.

enumerator kTRDC_MasterPrivilege
Use the bus master's attribute directly.

enumerator kTRDC_MasterPrivilege1
Use the bus master's attribute directly.

enum _trdc_pid_lock
PID lock configuration.

Values:

enumerator kTRDC_PidUnlocked0
The PID value can be updated by any secure priviledged write.

enumerator kTRDC_PidUnlocked1
The PID value can be updated by any secure priviledged write.

enumerator kTRDC_PidUnlocked2
The PID value can be updated by any secure priviledged write from the bus master that first configured this register.

enumerator kTRDC_PidLocked
The PID value is locked until next reset.

enum _trdc_controller
TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

Values:

enumerator kTRDC_MemBlockController0
Memory block checker 0.

enumerator kTRDC_MemBlockController1
Memory block checker 1.

enumerator kTRDC_MemBlockController2
Memory block checker 2.

enumerator kTRDC_MemBlockController3
Memory block checker 3.

enumerator kTRDC_MemRegionChecker0
Memory region checker 0.

enumerator kTRDC_MemRegionChecker1
Memory region checker 1.

enumerator kTRDC_MemRegionChecker2
Memory region checker 2.

enumerator kTRDC_MemRegionChecker3
Memory region checker 3.

enumerator kTRDC_MemRegionChecker4
Memory region checker 4.

enumerator kTRDC_MemRegionChecker5
Memory region checker 5.

enumerator kTRDC_MemRegionChecker6
Memory region checker 6.

enum _trdc_error_state
TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or TRDC_MRCn_DERR_W1[EST].

Values:

enumerator kTRDC_ErrorStateNone
No access violation detected.

enumerator kTRDC_ErrorStateNone1
No access violation detected.

enumerator kTRDC_ErrorStateSingle
Single access violation detected.

enumerator kTRDC_ErrorStateMulti
Multiple access violation detected.

enum _trdc_error_attr
TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

Values:

enumerator kTRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kTRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kTRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorSecurePrivilegeData
Secure privileged mode, data access.

enumerator kTRDC_ErrorNonSecureUserInst
NonSecure user mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecureUserData
NonSecure user mode, data access.

enumerator kTRDC_ErrorNonSecurePrivilegeInst
NonSecure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecurePrivilegeData
NonSecure privileged mode, data access.

enum _trdc_error_type
TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

Values:

enumerator kTRDC_ErrorTypeRead
Error occurs on read reference.

enumerator kTRDC_ErrorTypeWrite
Error occurs on write reference.

enum _trdc_region_descriptor

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

Values:

enumerator kTRDC_RegionDescriptor0
Region descriptor 0.

enumerator kTRDC_RegionDescriptor1
Region descriptor 1.

enumerator kTRDC_RegionDescriptor2
Region descriptor 2.

enumerator kTRDC_RegionDescriptor3
Region descriptor 3.

enumerator kTRDC_RegionDescriptor4
Region descriptor 4.

enumerator kTRDC_RegionDescriptor5
Region descriptor 5.

enumerator kTRDC_RegionDescriptor6
Region descriptor 6.

enumerator kTRDC_RegionDescriptor7
Region descriptor 7.

enumerator kTRDC_RegionDescriptor8
Region descriptor 8.

enumerator kTRDC_RegionDescriptor9
Region descriptor 9.

enumerator kTRDC_RegionDescriptor10
Region descriptor 10.

enumerator kTRDC_RegionDescriptor11
Region descriptor 11.

enumerator kTRDC_RegionDescriptor12
Region descriptor 12.

enumerator kTRDC_RegionDescriptor13
Region descriptor 13.

enumerator kTRDC_RegionDescriptor14
Region descriptor 14.

enumerator kTRDC_RegionDescriptor15
Region descriptor 15.

enum _trdc_MRC_domain

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

Values:

enumerator kTRDC_MrcDomain0
Domain 0.

enumerator kTRDC_MrcDomain1
Domain 1.

enumerator kTRDC_MrcDomain2
Domain 2.

enumerator kTRDC_MrcDomain3
Domain 3.

enumerator kTRDC_MrcDomain4
Domain 4.

enumerator kTRDC_MrcDomain5
Domain 5.

enumerator kTRDC_MrcDomain6
Domain 6.

enumerator kTRDC_MrcDomain7
Domain 7.

enumerator kTRDC_MrcDomain8
Domain 8.

enumerator kTRDC_MrcDomain9
Domain 9.

enumerator kTRDC_MrcDomain10
Domain 10.

enumerator kTRDC_MrcDomain11
Domain 11.

enumerator kTRDC_MrcDomain12
Domain 12.

enumerator kTRDC_MrcDomain13
Domain 13.

enumerator kTRDC_MrcDomain14
Domain 14.

enumerator kTRDC_MrcDomain15
Domain 15.

enum _trdc_MBC_domain
The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

Values:

enumerator kTRDC_MbcDomain0
Domain 0.

enumerator kTRDC_MbcDomain1
Domain 1.

enumerator kTRDC_MbcDomain2
Domain 2.

enumerator kTRDC_MbcDomain3
Domain 3.

enumerator kTRDC_MbcDomain4
Domain 4.

enumerator kTRDC_MbcDomain5
Domain 5.

enumerator kTRDC_MbcDomain6
Domain 6.

enumerator kTRDC_MbcDomain7
Domain 7.

enum _trdc_MBC_memory
The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

Values:

enumerator kTRDC_MbcSlaveMemory0
Memory 0.

enumerator kTRDC_MbcSlaveMemory1
Memory 1.

enumerator kTRDC_MbcSlaveMemory2
Memory 2.

enumerator kTRDC_MbcSlaveMemory3
Memory 3.

enum _trdc_MBC_bit
The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

Values:

enumerator kTRDC_MbcBit0
Bit 0.

enumerator kTRDC_MbcBit1
Bit 1.

enumerator kTRDC_MbcBit2
Bit 2.

enumerator kTRDC_MbcBit3
Bit 3.

enumerator kTRDC_MbcBit4
Bit 4.

enumerator kTRDC_MbcBit5
Bit 5.

enumerator kTRDC_MbcBit6
Bit 6.

enumerator kTRDC_MbcBit7
Bit 7.

enumerator kTRDC_MbcBit8
Bit 8.

enumerator kTRDC_MbcBit9
Bit 9.

enumerator kTRDC_MbcBit10
Bit 10.

enumerator kTRDC_MbcBit11
Bit 11.

enumerator kTRDC_MbcBit12
Bit 12.

enumerator kTRDC_MbcBit13
Bit 13.

enumerator kTRDC_MbcBit14
Bit 14.

enumerator kTRDC_MbcBit15
Bit 15.

enumerator kTRDC_MbcBit16
Bit 16.

enumerator kTRDC_MbcBit17
Bit 17.

enumerator kTRDC_MbcBit18
Bit 18.

enumerator kTRDC_MbcBit19
Bit 19.

enumerator kTRDC_MbcBit20
Bit 20.

enumerator kTRDC_MbcBit21
Bit 21.

enumerator kTRDC_MbcBit22
Bit 22.

enumerator kTRDC_MbcBit23
Bit 23.

enumerator kTRDC_MbcBit24
Bit 24.

enumerator kTRDC_MbcBit25
Bit 25.

enumerator kTRDC_MbcBit26
Bit 26.

enumerator kTRDC_MbcBit27
Bit 27.

enumerator kTRDC_MbcBit28
Bit 28.

enumerator kTRDC_MbcBit29
Bit 29.

enumerator kTRDC_MbcBit30
Bit 30.

enumerator kTRDC_MbcBit31

Bit 31.

typedef struct *_trdc_hardware_config* trdc_hardware_config_t

TRDC hardware configuration.

typedef struct *_trdc_slave_memory_hardware_config* trdc_slave_memory_hardware_config_t

Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *_trdc_did_sel* trdc_did_sel_t

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

typedef enum *_trdc_secure_attr* trdc_secure_attr_t

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

typedef enum *_trdc_pid_domain_hit_config* trdc_pid_domain_hit_config_t

The configuration of domain hit evaluation of PID.

typedef struct *_trdc_processor_domain_assignment* trdc_processor_domain_assignment_t

Domain assignment for the processor bus master.

typedef enum *_trdc_privilege_attr* trdc_privilege_attr_t

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

typedef struct *_trdc_non_processor_domain_assignment*
trdc_non_processor_domain_assignment_t

Domain assignment for the non-processor bus master.

typedef enum *_trdc_pid_lock* trdc_pid_lock_t

PID lock configuration.

typedef struct *_trdc_pid_config* trdc_pid_config_t

Process identifier(PID) configuration for processor cores.

typedef struct *_trdc_idau_config* trdc_idau_config_t

IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct *_trdc_flw_config* trdc_flw_config_t

FLW(Flash Logical Window) configuration.

typedef enum *_trdc_controller* trdc_controller_t

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

typedef enum *_trdc_error_state* trdc_error_state_t

TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or TRDC_MRCn_DERR_W1[EST].

typedef enum *_trdc_error_attr* trdc_error_attr_t

TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

typedef enum *_trdc_error_type* trdc_error_type_t

TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

typedef struct *_trdc_domain_error* trdc_domain_error_t

TRDC domain error definition.

typedef struct *_trdc_memory_access_control_config* trdc_memory_access_control_config_t
Memory access control configuration for MBC/MRC.

typedef struct *_trdc_mrc_region_descriptor_config* trdc_mrc_region_descriptor_config_t
The configuration of each region descriptor per domain per MRC instance.

typedef struct *_trdc_mbc_nse_update_config* trdc_mbc_nse_update_config_t
The configuration of MBC NSE update.

typedef struct *_trdc_mbc_memory_block_config* trdc_mbc_memory_block_config_t
The configuration of each memory block per domain per MBC instance.

FSL_TRDC_DRIVER_VERSION

struct _trdc_hardware_config
#include <fsl_trdc.h> TRDC hardware configuration.

Public Members

uint8_t masterNumber
Number of bus masters.

uint8_t domainNumber
Number of domains.

uint8_t mbcNumber
Number of MBCs.

uint8_t mrcNumber
Number of MRCs.

struct _trdc_slave_memory_hardware_config
#include <fsl_trdc.h> Hardware configuration of the two slave memories within each MBC(memory block checker).

Public Members

uint32_t blockNum
Number of blocks.

uint32_t blockSize
Block size.

struct _trdc_processor_domain_assignment
#include <fsl_trdc.h> Domain assignment for the processor bus master.

Public Members

uint32_t domainId
Domain ID.

uint32_t domainIdSelect
Domain ID select method, see trdc_did_sel_t.

uint32_t pidDomainHitConfig
The configuration of the domain hit evaluation for PID, see trdc_pid_domain_hit_config_t.

uint32_t pidMask

The mask combined with PID, so multiple PID can be included as part of the domain hit determination. Set to 0 to disable.

uint32_t secureAttr

Secure attribute, see trdc_secure_attr_t.

uint32_t pid

The process identifier, combined with pidMask to form the domain hit determination.

uint32_t ___pad0___

Reserved.

uint32_t lock

Lock the register.

uint32_t ___pad1___

Reserved.

struct __trdc__non__processor__domain__assignment

#include <fsl_trdc.h> Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId

Domain ID.

uint32_t privilegeAttr

Privileged attribute, see trdc_privilege_attr_t.

uint32_t secureAttr

Secure attribute, see trdc_secure_attr_t.

uint32_t bypassDomainId

Bypass domain ID.

uint32_t ___pad0___

Reserved.

uint32_t lock

Lock the register.

uint32_t ___pad1___

Reserved.

struct __trdc__pid__config

#include <fsl_trdc.h> Process identifier(PID) configuration for processor cores.

Public Members

uint32_t pid

The process identifier of the executing task. The highest bit can be used to define secure/nonsecure attribute of the task.

uint32_t ___pad0___

Reserved.

uint32_t lock

How to lock the register, see trdc_pid_lock_t.

uint32_t ___pad1___
Reserved.

struct __trdc__idau__config
#include <fsl_trdc.h> IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

Public Members

uint32_t ___pad0___
Reserved.

uint32_t lockSecureVTOR
Disable writes to secure VTOR(Vector Table Offset Register).

uint32_t lockNonsecureVTOR
Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32_t lockSecureMPU
Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32_t lockNonsecureMPU
Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32_t lockSAU
Disable writes to SAU(Security Attribution Unit) registers.

uint32_t ___pad1___
Reserved.

struct __trdc__flw__config
#include <fsl_trdc.h> FLW(Flash Logical Window) configuration.

Public Members

uint16_t blockCount
Block count of the Flash Logic Window in 32KByte blocks.

uint32_t arrayBaseAddr
Flash array base address of the Flash Logical Window.

bool lock
Disable writes to FLW registers.

bool enable
Enable FLW function.

struct __trdc__domain__error
#include <fsl_trdc.h> TRDC domain error definition.

Public Members

trdc_controller_t controller
Which controller captured access violation.

uint32_t address
: Access address that generated access violation.

trdc_error_state_t errorState
: Error state.

trdc_error_attr_t errorAttr
: Error attribute.

trdc_error_type_t errorType
: Error type.

uint8_t errorPort
: Error port.

uint8_t domainId
: Domain ID.

uint8_t slaveMemoryIdx
: The slave memory index. Only apply when violation in MBC.

struct __trdc_memory_access_control_config
: *#include <fsl_trdc.h>* Memory access control configuration for MBC/MRC.

Public Members

uint32_t nonsecureUsrX
: Allow nonsecure user execute access.

uint32_t nonsecureUsrW
: Allow nonsecure user write access.

uint32_t nonsecureUsrR
: Allow nonsecure user read access.

uint32_t ___pad0___
: Reserved.

uint32_t nonsecurePrivX
: Allow nonsecure privilege execute access.

uint32_t nonsecurePrivW
: Allow nonsecure privilege write access.

uint32_t nonsecurePrivR
: Allow nonsecure privilege read access.

uint32_t ___pad1___
: Reserved.

uint32_t secureUsrX
: Allow secure user execute access.

uint32_t secureUsrW
: Allow secure user write access.

uint32_t secureUsrR
: Allow secure user read access.

uint32_t ___pad2___
: Reserved.

uint32_t securePrivX
: 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.

2.118 Trdc_core

typedef struct *_TRDC_General_Type* TRDC__General__Type
: TRDC general configuration register definition.

typedef struct *_TRDC_FLW_Type* TRDC__FLW__Type
: TRDC flash logical control register definition.

typedef struct *_TRDC_DomainError_Type* TRDC__DomainError__Type
: TRDC domain error register definition.

typedef struct *_TRDC_DomainAssignment_Type* TRDC__DomainAssignment__Type
: TRDC master domain assignment register definition.

typedef struct *_TRDC_MBC_Type* TRDC__MBC__Type
: TRDC MBC control register definition.

typedef struct *_TRDC_MRC_Type* TRDC_MRC_Type
TRDC MRC control register definition. MRC_DOM0_RGD_W[region][word].

TRDC_GENERAL_BASE(base)
TRDC base address convert macro.

TRDC_FLW_BASE(base)

TRDC_DOMAIN_ERROR_BASE(base)

TRDC_DOMAIN_ASSIGNMENT_BASE(base)

TRDC_MBC_BASE(base, instance)

TRDC_MRC_BASE(base, instance)

struct _TRDC_General_Type
#include <fsl_trdc_core.h> TRDC general configuration register definition.

Public Members

___IO uint32_t TRDC_CR
TRDC Register, offset: 0x0

___I uint32_t TRDC_HWCFG0
TRDC Hardware Configuration Register 0, offset: 0xF0

___I uint32_t TRDC_HWCFG1
TRDC Hardware Configuration Register 1, offset: 0xF4

___I uint32_t TRDC_HWCFG2
TRDC Hardware Configuration Register 2, offset: 0xF8

___I uint32_t TRDC_HWCFG3
TRDC Hardware Configuration Register 3, offset: 0xFC

___I uint8_t DACFG [8]
Domain Assignment Configuration Register, array offset: 0x100, array step: 0x1

___IO uint32_t TRDC_IDAU_CR
TRDC IDAU Control Register, offset: 0x1C0

struct _TRDC_FLW_Type
#include <fsl_trdc_core.h> TRDC flash logical control register definition.

Public Members

___IO uint32_t TRDC_FLW_CTL
TRDC FLW Control, offset: 0x1E0

___I uint32_t TRDC_FLW_PBASE
TRDC FLW Physical Base, offset: 0x1E4

___IO uint32_t TRDC_FLW_ABASE
TRDC FLW Array Base, offset: 0x1E8

___IO uint32_t TRDC_FLW_BCNT
TRDC FLW Block Count, offset: 0x1EC

struct _TRDC_DomainError_Type
#include <fsl_trdc_core.h> TRDC domain error register definition.

Public Members

___IO uint32_t TRDC_FDID

TRDC Fault Domain ID, offset: 0x1FC

___I uint32_t TRDC_DERRLOC [16]

TRDC Domain Error Location Register, array offset: 0x200, array step: 0x4

struct _TRDC_DomainAssignment_Type

#include <fsl_trdc_core.h> TRDC master domain assignment register definition.

Public Members

___IO uint32_t PID [8]

Process Identifier, array offset: 0x700, array step: 0x4

struct _TRDC_MBC_Type

#include <fsl_trdc_core.h> TRDC MBC control register definition.

Public Members

___I uint32_t MBC_MEM_GLBCFG [4]

MBC Global Configuration Register, array offset: 0x10000, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_NSE_BLK_INDEX

MBC NonSecure Enable Block Index, array offset: 0x10010, array step: 0x2000

___O uint32_t MBC_NSE_BLK_SET

MBC NonSecure Enable Block Set, array offset: 0x10014, array step: 0x2000

___O uint32_t MBC_NSE_BLK_CLR

MBC NonSecure Enable Block Clear, array offset: 0x10018, array step: 0x2000

___O uint32_t MBC_NSE_BLK_CLR_ALL

MBC NonSecure Enable Block Clear All, array offset: 0x1001C, array step: 0x2000

___IO uint32_t MBC_MEMN_GLBAC [8]

MBC Global Access Control, array offset: 0x10020, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10040, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10140, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10180, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x101A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x101A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x101C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x101D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM0_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x101F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10240, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10340, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10380, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x103A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x103A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x103C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x103D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM1_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x103F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10440, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10540, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10580, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x105A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x105A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x105C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x105D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM2_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x105F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10640, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10740, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10780, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x107A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x107A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x107C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x107D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM3_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x107F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10840, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10940, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10980, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x109A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x109A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x109C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x109D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM4_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x109F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10A40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10B40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10B80, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10BA0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10BA8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10BC8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10BD0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM5_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10BF0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10C40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10D40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10D80, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10DA0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10DA8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10DC8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10DD0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM6_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10DF0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x10E40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x10F40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10F80, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10FA0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10FA8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10FC8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x10FD0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM7_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x10FF0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11040, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11140, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11180, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x111A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x111A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x111C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x111D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM8_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x111F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11240, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11340, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11380, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x113A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x113A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x113C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x113D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM9_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x113F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11440, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11540, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11580, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x115A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x115A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x115C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x115D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x115F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11640, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11740, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11780, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x117A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x117A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x117C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x117D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x117F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11840, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11940, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11980, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x119A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x119A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x119C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x119D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x119F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11A40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11B40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11B80, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11BA0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11BA8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11BC8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11BD0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM13_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11BF0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11C40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11D40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11D80, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11DA0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11DA8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11DC8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11DD0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM14_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11DF0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM0_BLK_CFG_W [64]

MBC Memory Block Configuration Word, array offset: 0x11E40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM0_BLK_NSE_W [16]

MBC Memory Block NonSecure Enable Word, array offset: 0x11F40, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM1_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11F80, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM1_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11FA0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM2_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11FA8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM2_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11FC8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM3_BLK_CFG_W [8]

MBC Memory Block Configuration Word, array offset: 0x11FD0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM15_MEM3_BLK_NSE_W [2]

MBC Memory Block NonSecure Enable Word, array offset: 0x11FF0, array step: index*0x2000, index2*0x4

struct _TRDC_MRC_Type

#include <fsl_trdc_core.h> TRDC MRC control register definition. MRC_DOM0_RGD_W[region][word].

Public Members

___I uint32_t MRC_GLBCFG

MRC Global Configuration Register, array offset: 0x14000, array step: 0x1000

___IO uint32_t MRC_NSE_RGN_INDIRECT

MRC NonSecure Enable Region Indirect, array offset: 0x14010, array step: 0x1000

___O uint32_t MRC_NSE_RGN_SET

MRC NonSecure Enable Region Set, array offset: 0x14014, array step: 0x1000

___O uint32_t MRC_NSE_RGN_CLR

MRC NonSecure Enable Region Clear, array offset: 0x14018, array step: 0x1000

___O uint32_t MRC_NSE_RGN_CLR_ALL

MRC NonSecure Enable Region Clear All, array offset: 0x1401C, array step: 0x1000

___IO uint32_t MRC_GLBAC [8]

MRC Global Access Control, array offset: 0x14020, array step: index*0x1000, index2*0x4

___IO uint32_t MRC_DOM0_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14040, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM0_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x140C0, array step: 0x1000

___IO uint32_t MRC_DOM1_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14140, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM1_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x141C0, array step: 0x1000

___IO uint32_t MRC_DOM2_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14240, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM2_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x142C0, array step: 0x1000

___IO uint32_t MRC_DOM3_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14340, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM3_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x143C0, array step: 0x1000

___IO uint32_t MRC_DOM4_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14440, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM4_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x144C0, array step: 0x1000

___IO uint32_t MRC_DOM5_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14540, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM5_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x145C0, array step: 0x1000

___IO uint32_t MRC_DOM6_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14640, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM6_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x146C0, array step: 0x1000

___IO uint32_t MRC_DOM7_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14740, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM7_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x147C0, array step: 0x1000

___IO uint32_t MRC_DOM8_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14840, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM8_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x148C0, array step: 0x1000

___IO uint32_t MRC_DOM9_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14940, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM9_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x149C0, array step: 0x1000

___IO uint32_t MRC_DOM10_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14A40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM10_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x14AC0, array step: 0x1000

___IO uint32_t MRC_DOM11_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14B40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM11_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x14BC0, array step: 0x1000

___IO uint32_t MRC_DOM12_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14C40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM12_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x14CC0, array step: 0x1000

___IO uint32_t MRC_DOM13_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14D40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM13_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x14DC0, array step: 0x1000

___IO uint32_t MRC_DOM14_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14E40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM14_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x14EC0, array step: 0x1000

___IO uint32_t MRC_DOM15_RGD_W [16][2]

MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14F40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM15_RGD_NSE

MRC Region Descriptor NonSecure Enable, array offset: 0x14FC0, array step: 0x1000

struct MBC_DERR

Public Members

___I uint32_t W0

MBC Domain Error Word0 Register, array offset: 0x400, array step: 0x10

___I uint32_t W1

MBC Domain Error Word1 Register, array offset: 0x404, array step: 0x10

___O uint32_t W3

MBC Domain Error Word3 Register, array offset: 0x40C, array step: 0x10

struct MRC_DERR

Public Members

___I uint32_t W0

MRC Domain Error Word0 Register, array offset: 0x480, array step: 0x10

___I uint32_t W1

MRC Domain Error Word1 Register, array offset: 0x484, array step: 0x10

___O uint32_t W3

MRC Domain Error Word3 Register, array offset: 0x48C, array step: 0x10

union ___unnamed132___

Public Members

struct *_TRDC_DomainAssignment_Type* MDA_DFMT0[8]

struct *_TRDC_DomainAssignment_Type* MDA_DFMT1[8]

struct MDA_DFMT0

Public Members

___IO uint32_t MDA_W_DFMT0 [8]

DAC Master Domain Assignment Register, array offset: 0x800, array step: index*0x20, index2*0x4

struct MDA_DFMT1

Public Members

___IO uint32_t MDA_W_DFMT1 [1]

DAC Master Domain Assignment Register, array offset: 0x800, array step: index*0x20, index2*0x4

2.119 Trdc_soc

FSL_TRDC_SOC_DRIVER_VERSION

Driver version 2.0.0.

TRDC_BASE_PTRS

TRDC base table.

TRDC_MBC_MEM_GLBCFG_NBLKS_MASK

TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_MASK

TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_SHIFT

TRDC_MBC_NSE_BLK_CLR_ALL_MEMSEL

TRDC_MBC_NSE_BLK_CLR_ALL_DID_SEL

FSL_FEATURE_TRDC_DOMAIN_COUNT

TRDC feature.

TRDC_MBC_COUNT

TRDC base address convert macro.

TRDC_MBC_OFFSET(x)

TRDC_MBC_ARRAY_STEP

FSL_COMPONENT_ID

2.120 Tsi_v6_driver

enum _tsi_main_clock_selection

TSI main clock selection.

These constants set the tsi main clock.

Values:

enumerator kTSI_MainClockSlection_0

Set TSI main clock frequency to 20.72MHz

enumerator kTSI_MainClockSlection_1
Set TSI main clock frequency to 16.65MHz

enumerator kTSI_MainClockSlection_2
Set TSI main clock frequency to 13.87MHz

enumerator kTSI_MainClockSlection_3
Set TSI main clock frequency to 11.91MHz

enum _tsi_sensing_mode_selection
TSI sensing mode selection.

These constants set the tsi sensing mode.

Values:

enumerator kTSI_SensingModeSlection_Self
Set TSI sensing mode to self-cap mode

enumerator kTSI_SensingModeSlection_Mutual
Set TSI sensing mode to mutual-cap mode

enum _tsi_dvolt_option
TSI DVOLT settings.

These bits indicate the comparator vp, vm and dvolt voltage.

Values:

enumerator kTSI_DvoltOption_0
DVOLT value option 0, the value may differ on different platforms

enumerator kTSI_DvoltOption_1
DVOLT value option 1, the value may differ on different platforms

enumerator kTSI_DvoltOption_2
DVOLT value option 2, the value may differ on different platforms

enumerator kTSI_DvoltOption_3
DVOLT value option 3, the value may differ on different platforms

enumerator kTSI_DvoltOption_4
DVOLT value option 4, the value may differ on different platforms

enumerator kTSI_DvoltOption_5
DVOLT value option 5, the value may differ on different platforms

enum _tsi_sensitivity_xdn_option
TSI sensitivity ajustment (XDN option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

Values:

enumerator kTSI_SensitivityXdnOption_0
Adjust sensitivity in self-cap mode, 1/16

enumerator kTSI_SensitivityXdnOption_1
Adjust sensitivity in self-cap mode, 1/8

enumerator kTSI_SensitivityXdnOption_2
Adjust sensitivity in self-cap mode, 1/4

enumerator kTSI_SensitivityXdnOption_3
Adjust sensitivity in self-cap mode, 1/2

enumerator kTSI_SensitivityXdnOption_4
Adjust sensitivity in self-cap mode, 1/1

enumerator kTSI_SensitivityXdnOption_5
Adjust sensitivity in self-cap mode, 2/1

enumerator kTSI_SensitivityXdnOption_6
Adjust sensitivity in self-cap mode, 4/1

enumerator kTSI_SensitivityXdnOption_7
Adjust sensitivity in self-cap mode, 8/1

enum _tsi_shield
TSI Shield setting (S_W_SHIELD option).

These constants define the shield pin used for HW shielding functionality. One or more shield pin can be selected. The involved bitfield is not fix can change from device to device (KE16Z7 and KE17Z7 support 3 shield pins, other KE serials only support 1 shield pin, MCXN devices has 4 shield pins).

Values:

enumerator kTSI_shieldAllOff
No pin used

enumerator kTSI_shield0On
Shield 0 pin used

enumerator kTSI_shield1On
Shield 1 pin used

enumerator kTSI_shield1and0On
Shield 0,1 pins used

enumerator kTSI_shield2On
Shield 2 pin used

enumerator kTSI_shield2and0On
Shield 2,0 pins used

enumerator kTSI_shield2and1On
Shield 2,1 pins used

enumerator kTSI_shield2and1and0On
Shield 2,1,0 pins used

enumerator kTSI_shield3On
Shield 3 pin used

enumerator kTSI_shield3and0On
Shield 3,0 pins used

enumerator kTSI_shield3and1On
Shield 3,1 pins used

enumerator kTSI_shield3and1and0On
Shield 3,1,0 pins used

enumerator kTSI_shield3and2On
Shield 3,2 pin used

enumerator kTSI_shield3and2and0On
Shield 3,2,0 pins used

enumerator kTSI_shield3and2and1On
Shield 3,2,1 pins used

enumerator kTSI_shieldAllOn
Shield 3,2,1,0 pins used

enum _tsi_sensitivity_ctrim_option
TSI sensitivity ajustment (CTRIM option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

Values:

enumerator kTSI_SensitivityCtrimOption_0
Adjust sensitivity in self-cap mode, 2.5p

enumerator kTSI_SensitivityCtrimOption_1
Adjust sensitivity in self-cap mode, 5.0p

enumerator kTSI_SensitivityCtrimOption_2
Adjust sensitivity in self-cap mode, 7.5p

enumerator kTSI_SensitivityCtrimOption_3
Adjust sensitivity in self-cap mode, 10.0p

enumerator kTSI_SensitivityCtrimOption_4
Adjust sensitivity in self-cap mode, 12.5p

enumerator kTSI_SensitivityCtrimOption_5
Adjust sensitivity in self-cap mode, 15.0p

enumerator kTSI_SensitivityCtrimOption_6
Adjust sensitivity in self-cap mode, 17.5p

enumerator kTSI_SensitivityCtrimOption_7
Adjust sensitivity in self-cap mode, 20.0p

enum _tsi_current_multiple_input
TSI current ajustment (Input current multiple).

These constants set the tsi input current multiple in self-cap mode.

Values:

enumerator kTSI_CurrentMultipleInputValue_0
Adjust input current multiple in self-cap mode, 1/8

enumerator kTSI_CurrentMultipleInputValue_1
Adjust input current multiple in self-cap mode, 1/4

enum _tsi_current_multiple_charge
TSI current ajustment (Charge/Discharge current multiple).

These constants set the tsi charge/discharge current multiple in self-cap mode.

Values:

enumerator kTSI_CurrentMultipleChargeValue_0
Adjust charge/discharge current multiple in self-cap mode, 1/16

enumerator kTSI_CurrentMultipleChargeValue_1
Adjust charge/discharge current multiple in self-cap mode, 1/8

enumerator kTSI_CurrentMultipleChargeValue_2
Adjust charge/discharge current multiple in self-cap mode, 1/4

enumerator kTSI_CurrentMultipleChargeValue_3
Adjust charge/discharge current multiple in self-cap mode, 1/2

enumerator kTSI_CurrentMultipleChargeValue_4
Adjust charge/discharge current multiple in self-cap mode, 1/1

enumerator kTSI_CurrentMultipleChargeValue_5
Adjust charge/discharge current multiple in self-cap mode, 2/1

enumerator kTSI_CurrentMultipleChargeValue_6
Adjust charge/discharge current multiple in self-cap mode, 4/1

enumerator kTSI_CurrentMultipleChargeValue_7
Adjust charge/discharge current multiple in self-cap mode, 8/1

enum __tsi_mutual_pre_current
TSI current used in vref generator.

These constants Choose the current used in vref generator.

Values:

enumerator kTSI_MutualPreCurrent_1uA
Vref generator current is 1uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_2uA
Vref generator current is 2uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_3uA
Vref generator current is 3uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_4uA
Vref generator current is 4uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_5uA
Vref generator current is 5uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_6uA
Vref generator current is 6uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_7uA
Vref generator current is 7uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_8uA
Vref generator current is 8uA, used in mutual-cap mode

enum __tsi_mutual_pre_resistor
TSI resistor used in pre-charge.

These constants Choose the resistor used in pre-charge.

Values:

enumerator kTSI_MutualPreResistor_1k
Vref generator resistor is 1k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_2k
Vref generator resistor is 2k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_3k
Vref generator resistor is 3k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_4k
Vref generator resistor is 4k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_5k
Vref generator resistor is 5k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_6k
Vref generator resistor is 6k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_7k
Vref generator resistor is 7k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_8k
Vref generator resistor is 8k, used in mutual-cap mode

enum _tsi_mutual_sense_resistor
TSI resistor used in I-sense generator.

These constants Choose the resistor used in I-sense generator.

Values:

enumerator kTSI_MutualSenseResistor_2k5
I-sense resistor is 2.5k , used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_5k
I-sense resistor is 5.0k , used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_7k5
I-sense resistor is 7.5k , used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_10k
I-sense resistor is 10.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_12k5
I-sense resistor is 12.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_15k
I-sense resistor is 15.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_17k5
I-sense resistor is 17.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_20k
I-sense resistor is 20.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_22k5
I-sense resistor is 22.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_25k
I-sense resistor is 25.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_27k5
I-sense resistor is 27.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_30k
I-sense resistor is 30.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_32k5
I-sense resistor is 32.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_35k
I-sense resistor is 35.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_37k5
I-sense resistor is 37.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_40k
I-sense resistor is 40.0k, used in mutual-cap mode

enum __tsi_mutual_tx_channel
TSI TX channel selection in mutual-cap mode.

These constants Choose the TX channel used in mutual-cap mode.

Values:

enumerator kTSI_MutualTxChannel_0
Select channel 0 as tx0, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_1
Select channel 1 as tx1, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_2
Select channel 2 as tx2, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_3
Select channel 3 as tx3, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_4
Select channel 4 as tx4, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_5
Select channel 5 as tx5, used in mutual-cap mode

enum __tsi_mutual_rx_channel
TSI RX channel selection in mutual-cap mode.

These constants Choose the RX channel used in mutual-cap mode.

Values:

enumerator kTSI_MutualRxChannel_6
Select channel 6 as rx6, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_7
Select channel 7 as rx7, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_8
Select channel 8 as rx8, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_9
Select channel 9 as rx9, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_10
Select channel 10 as rx10, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_11
Select channel 11 as rx11, used in mutual-cap mode

enum __tsi_mutual_sense_boost_current
TSI sensitivity boost current settings.

These constants set the sensitivity boost current.

Values:

enumerator kTSI_MutualSenseBoostCurrent_0uA
Sensitivity boost current is 0uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_2uA
Sensitivity boost current is 2uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_4uA
Sensitivity boost current is 4uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_6uA
Sensitivity boost current is 6uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_8uA
Sensitivity boost current is 8uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_10uA
Sensitivity boost current is 10uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_12uA
Sensitivity boost current is 12uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_14uA
Sensitivity boost current is 14uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_16uA
Sensitivity boost current is 16uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_18uA
Sensitivity boost current is 18uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_20uA
Sensitivity boost current is 20uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_22uA
Sensitivity boost current is 22uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_24uA
Sensitivity boost current is 24uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_26uA
Sensitivity boost current is 26uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_28uA
Sensitivity boost current is 28uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_30uA
Sensitivity boost current is 30uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_32uA
Sensitivity boost current is 32uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_34uA
Sensitivity boost current is 34uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_36uA
Sensitivity boost current is 36uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_38uA
Sensitivity boost current is 38uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_40uA
Sensitivity boost current is 40uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_42uA
Sensitivity boost current is 42uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_44uA
Sensitivity boost current is 44uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_46uA
Sensitivity boost current is 46uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_48uA
Sensitivity boost current is 48uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_50uA
Sensitivity boost current is 50uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_52uA
Sensitivity boost current is 52uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_54uA
Sensitivity boost current is 54uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_56uA
Sensitivity boost current is 56uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_58uA
Sensitivity boost current is 58uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_60uA
Sensitivity boost current is 60uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_62uA
Sensitivity boost current is 62uA, used in mutual-cap mode

enum __tsi_mutual_tx_drive_mode
TSI TX drive mode control.

These constants Choose the TX drive mode control setting.

Values:

enumerator kTSI_MutualTxDriveModeOption_0
TX drive mode is -5v ~ +5v, used in mutual-cap mode

enumerator kTSI_MutualTxDriveModeOption_1
TX drive mode is 0v ~ +5v, used in mutual-cap mode

enum __tsi_mutual_pmos_current_left
TSI Pmos current mirror selection on the left side.

These constants set the Pmos current mirror on the left side used in mutual-cap mode.

Values:

enumerator kTSI_MutualPmosCurrentMirrorLeft_4
Set Pmos current mirror left value as 4, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_8
Set Pmos current mirror left value as 8, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_12
Set Pmos current mirror left value as 12, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_16
Set Pmos current mirror left value as 16, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_20
Set Pmos current mirror left value as 20, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_24
Set Pmos current mirror left value as 24, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_28
Set Pmos current mirror left value as 28, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_32
Set Pmos current mirror left value as 32, used in mutual-cap mode

enum _tsi_mutual_pmos_current_right
TSI Pmos current mirror selection on the right side.

These constants set the Pmos current mirror on the right side used in mutual-cap mode.

Values:

enumerator kTSI_MutualPmosCurrentMirrorRight_1
Set Pmos current mirror right value as 1, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorRight_2
Set Pmos current mirror right value as 2, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorRight_3
Set Pmos current mirror right value as 3, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorRight_4
Set Pmos current mirror right value as 4, used in mutual-cap mode

enum _tsi_mutual_nmos_current
TSI Nmos current mirror selection.

These constants set the Nmos current mirror used in mutual-cap mode.

Values:

enumerator kTSI_MutualNmosCurrentMirror_1
Set Nmos current mirror value as 1, used in mutual-cap mode

enumerator kTSI_MutualNmosCurrentMirror_2
Set Nmos current mirror value as 2, used in mutual-cap mode

enumerator kTSI_MutualNmosCurrentMirror_3
Set Nmos current mirror value as 3, used in mutual-cap mode

enumerator kTSI_MutualNmosCurrentMirror_4
Set Nmos current mirror value as 4, used in mutual-cap mode

enum _tsi_sinc_cutoff_div
TSI SINC cutoff divider setting.

These bits set the SINC cutoff divider.

Values:

enumerator kTSI_SincCutoffDiv_1
Set SINC cutoff divider as 1

enumerator kTSI_SincCutoffDiv_2
Set SINC cutoff divider as 2

enumerator kTSI_SincCutoffDiv_4
Set SINC cutoff divider as 4

enumerator kTSI_SincCutoffDiv_8
Set SINC cutoff divider as 8

enumerator kTSI_SincCutoffDiv_16
Set SINC cutoff divider as 16

enumerator kTSI_SincCutoffDiv_32
Set SINC cutoff divider as 32

enumerator kTSI_SincCutoffDiv_64
Set SINC cutoff divider as 64

enumerator kTSI_SincCutoffDiv_128
Set SINC cutoff divider as 128

enum __tsi_sinc_filter_order
TSI SINC filter order setting.

These bits set the SINC filter order.

Values:

enumerator kTSI_SincFilterOrder_1
Use 1 order SINC filter

enumerator kTSI_SincFilterOrder_2
Use 1 order SINC filter

enum __tsi_sinc_decimation_value
TSI SINC decimation value setting.

These bits set the SINC decimation value.

Values:

enumerator kTSI_SincDecimationValue_1
The TSI_DATA[TSICH] bits is the counter value of 1 triger period.

enumerator kTSI_SincDecimationValue_2
The TSI_DATA[TSICH] bits is the counter value of 2 triger period.

enumerator kTSI_SincDecimationValue_3
The TSI_DATA[TSICH] bits is the counter value of 3 triger period.

enumerator kTSI_SincDecimationValue_4
The TSI_DATA[TSICH] bits is the counter value of 4 triger period.

enumerator kTSI_SincDecimationValue_5
The TSI_DATA[TSICH] bits is the counter value of 5 triger period.

enumerator kTSI_SincDecimationValue_6
The TSI_DATA[TSICH] bits is the counter value of 6 triger period.

enumerator kTSI_SincDecimationValue_7
The TSI_DATA[TSICH] bits is the counter value of 7 triger period.

enumerator kTSI_SincDecimationValue_8
The TSI_DATA[TSICH] bits is the counter value of 8 triger period.

enumerator kTSI_SincDecimationValue_9
The TSI_DATA[TSICH] bits is the counter value of 9 triger period.

enumerator kTSI_SincDecimationValue_10
The TSI_DATA[TSICH] bits is the counter value of 10 triger period.

enumerator kTSI_SincDecimationValue_11
The TSI_DATA[TSICH] bits is the counter value of 11 triger period.

enumerator kTSI_SincDecimationValue_12
The TSI_DATA[TSICH] bits is the counter value of 12 triger period.

enumerator kTSI_SincDecimationValue_13
The TSI_DATA[TSICH] bits is the counter value of 13 triger period.

enumerator kTSI_SincDecimationValue_14
The TSI_DATA[TSICH] bits is the counter value of 14 triger period.

enumerator kTSI_SincDecimationValue_15
The TSI_DATA[TSICH] bits is the counter value of 15 triger period.

enumerator kTSI_SincDecimationValue_16
The TSI_DATA[TSICH] bits is the counter value of 16 triger period.

enumerator kTSI_SincDecimationValue_17
The TSI_DATA[TSICH] bits is the counter value of 17 triger period.

enumerator kTSI_SincDecimationValue_18
The TSI_DATA[TSICH] bits is the counter value of 18 triger period.

enumerator kTSI_SincDecimationValue_19
The TSI_DATA[TSICH] bits is the counter value of 19 triger period.

enumerator kTSI_SincDecimationValue_20
The TSI_DATA[TSICH] bits is the counter value of 20 triger period.

enumerator kTSI_SincDecimationValue_21
The TSI_DATA[TSICH] bits is the counter value of 21 triger period.

enumerator kTSI_SincDecimationValue_22
The TSI_DATA[TSICH] bits is the counter value of 22 triger period.

enumerator kTSI_SincDecimationValue_23
The TSI_DATA[TSICH] bits is the counter value of 23 triger period.

enumerator kTSI_SincDecimationValue_24
The TSI_DATA[TSICH] bits is the counter value of 24 triger period.

enumerator kTSI_SincDecimationValue_25
The TSI_DATA[TSICH] bits is the counter value of 25 triger period.

enumerator kTSI_SincDecimationValue_26
The TSI_DATA[TSICH] bits is the counter value of 26 triger period.

enumerator kTSI_SincDecimationValue_27
The TSI_DATA[TSICH] bits is the counter value of 27 triger period.

enumerator kTSI_SincDecimationValue_28
The TSI_DATA[TSICH] bits is the counter value of 28 triger period.

enumerator kTSI_SincDecimationValue_29
The TSI_DATA[TSICH] bits is the counter value of 29 triger period.

enumerator kTSI_SincDecimationValue_30
The TSI_DATA[TSICH] bits is the counter value of 30 triger period.

enumerator kTSI_SincDecimationValue_31
The TSI_DATA[TSICH] bits is the counter value of 31 triger period.

enumerator kTSI_SincDecimationValue_32
The TSI_DATA[TSICH] bits is the counter value of 32 triger period.

enum _tsi_ssc_charge_num
TSI SSC output bit0's period setting(SSC0[CHARGE_NUM])
These bits set the SSC output bit0's period setting.
Values:

enumerator kTSI_SscChargeNumValue_1
The SSC output bit 0's period will be 1 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_2
The SSC output bit 0's period will be 2 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_3
The SSC output bit 0's period will be 3 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_4
The SSC output bit 0's period will be 4 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_5
The SSC output bit 0's period will be 5 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_6
The SSC output bit 0's period will be 6 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_7
The SSC output bit 0's period will be 7 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_8
The SSC output bit 0's period will be 8 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_9
The SSC output bit 0's period will be 9 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_10
The SSC output bit 0's period will be 10 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_11
The SSC output bit 0's period will be 11 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_12
The SSC output bit 0's period will be 12 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_13
The SSC output bit 0's period will be 13 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_14
The SSC output bit 0's period will be 14 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_15
The SSC output bit 0's period will be 15 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_16
The SSC output bit 0's period will be 16 clock cycle of system clock.

enum _tsi_ssc_nocharge_num
TSI SSC output bit1's period setting(SSC0[BASE_NOCHARGE_NUM])
These bits set the SSC output bit1's period setting.
Values:

enumerator kTSI_SscNoChargeNumValue_1
The SSC output bit 1's basic period will be 1 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_2
The SSC output bit 1's basic period will be 2 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_3
The SSC output bit 1's basic period will be 3 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_4
The SSC output bit 1's basic period will be 4 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_5
The SSC output bit 1's basic period will be 5 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_6
The SSC output bit 1's basic period will be 6 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_7
The SSC output bit 1's basic period will be 7 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_8
The SSC output bit 1's basic period will be 8 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_9
The SSC output bit 1's basic period will be 9 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_10
The SSC output bit 1's basic period will be 10 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_11
The SSC output bit 1's basic period will be 11 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_12
The SSC output bit 1's basic period will be 12 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_13
The SSC output bit 1's basic period will be 13 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_14
The SSC output bit 1's basic period will be 14 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_15
The SSC output bit 1's basic period will be 15 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_16
The SSC output bit 1's basic period will be 16 clock cycle of system clock.

enum _tsi_ssc_prbs_outsel
TSI SSC outsel choosing the length of the PRBS (Pseudo-RandomBinarySequence) method setting(SSC0[TSI_SSC0_PRBS_OUTSEL])

These bits set the SSC PRBS length.

Values:

enumerator kTSI_SscPrbsOutsel_2
The length of the PRBS is 2.

enumerator kTSI_SscPrbsOutsel_3
The length of the PRBS is 3.

enumerator kTSI_SscPrbsOutsel_4
The length of the PRBS is 4.

enumerator kTSI_SscPrbsOutsel_5
The length of the PRBS is 5.

enumerator kTSI_SscPrbsOutsel_6
The length of the PRBS is 6.

enumerator kTSI_SscPrbsOutsel_7
The length of the PRBS is 7.

enumerator kTSI_SscPrbsOutsel_8
The length of the PRBS is 8.

enumerator kTSI_SscPrbsOutsel_9
The length of the PRBS is 9.

enumerator kTSI_SscPrbsOutsel_10
The length of the PRBS is 10.

enumerator kTSI_SscPrbsOutsel_11
The length of the PRBS is 11.

enumerator kTSI_SscPrbsOutsel_12
The length of the PRBS is 12.

enumerator kTSI_SscPrbsOutsel_13
The length of the PRBS is 13.

enumerator kTSI_SscPrbsOutsel_14
The length of the PRBS is 14.

enumerator kTSI_SscPrbsOutsel_15
The length of the PRBS is 15.

enum _tsi_status_flags
TSI status flags.

Values:

enumerator kTSI_EndOfScanFlag
End-Of-Scan flag

enumerator kTSI_OutOfRangeFlag
Out-Of-Range flag

enum _tsi_interrupt_enable
TSI feature interrupt source.

Values:

enumerator kTSI_GlobalInterruptEnable
TSI module global interrupt

enumerator kTSI_OutOfRangeInterruptEnable
Out-Of-Range interrupt

enumerator kTSI_EndOfScanInterruptEnable
End-Of-Scan interrupt

enum _tsi_ssc_mode
TSI SSC mode selection.

These constants set the SSC mode.

Values:

enumerator kTSI_ssc_prbs_method
Using PRBS method generating SSC output bit.

enumerator kTSI_ssc_up_down_counter
Using up-down counter generating SSC output bit.

enumerator kTSI_ssc_dissable
SSC function is disabled.

enum _tsi_ssc_prescaler
TSI main clock selection.

These constants set select the divider ratio for the clock used for generating the SSC output bit.

Values:

enumerator kTSI_ssc_div_by_1
Set SSC divider to 00000000 div1(2^0)

enumerator kTSI_ssc_div_by_2
Set SSC divider to 00000001 div2(2^1)

enumerator kTSI_ssc_div_by_4
Set SSC divider to 00000011 div4(2^2)

enumerator kTSI_ssc_div_by_8
Set SSC divider to 00000111 div8(2^3)

enumerator kTSI_ssc_div_by_16
Set SSC divider to 00001111 div16(2^4)

enumerator kTSI_ssc_div_by_32
Set SSC divider to 00011111 div32(2^5)

enumerator kTSI_ssc_div_by_64
Set SSC divider to 00111111 div64(2^6)

enumerator kTSI_ssc_div_by_128
Set SSC divider to 01111111 div128(2^7)

enumerator kTSI_ssc_div_by_256
Set SSC divider to 11111111 div256(2^8)

typedef enum *_tsi_main_clock_selection* tsi_main_clock_selection_t
TSI main clock selection.

These constants set the tsi main clock.

typedef enum *_tsi_sensing_mode_selection* tsi_sensing_mode_selection_t
TSI sensing mode selection.

These constants set the tsi sensing mode.

typedef enum *_tsi_dvolt_option* tsi_dvolt_option_t
TSI DVOLT settings.

These bits indicate the comparator vp, vm and dvolt voltage.

typedef enum *_tsi_sensitivity_xdn_option* tsi_sensitivity_xdn_option_t
TSI sensitivity ajustment (XDN option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

typedef enum *_tsi_shield* tsi_shield_t

TSI Shield setting (S_W_SHIELD option).

These constants define the shield pin used for HW shielding functionality. One or more shield pin can be selected. The involved bitfield is not fix can change from device to device (KE16Z7 and KE17Z7 support 3 shield pins, other KE serials only support 1 shield pin, MCXN devices has 4 shield pins).

typedef enum *_tsi_sensitivity_ctrim_option* tsi_sensitivity_ctrim_option_t

TSI sensitivity ajustment (CTRIM option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

typedef enum *_tsi_current_multiple_input* tsi_current_multiple_input_t

TSI current ajustment (Input current multiple).

These constants set the tsi input current multiple in self-cap mode.

typedef enum *_tsi_current_multiple_charge* tsi_current_multiple_charge_t

TSI current ajustment (Charge/Discharge current multiple).

These constants set the tsi charge/discharge current multiple in self-cap mode.

typedef enum *_tsi_mutual_pre_current* tsi_mutual_pre_current_t

TSI current used in vref generator.

These constants Choose the current used in vref generator.

typedef enum *_tsi_mutual_pre_resistor* tsi_mutual_pre_resistor_t

TSI resistor used in pre-charge.

These constants Choose the resistor used in pre-charge.

typedef enum *_tsi_mutual_sense_resistor* tsi_mutual_sense_resistor_t

TSI resistor used in I-sense generator.

These constants Choose the resistor used in I-sense generator.

typedef enum *_tsi_mutual_tx_channel* tsi_mutual_tx_channel_t

TSI TX channel selection in mutual-cap mode.

These constants Choose the TX channel used in mutual-cap mode.

typedef enum *_tsi_mutual_rx_channel* tsi_mutual_rx_channel_t

TSI RX channel selection in mutual-cap mode.

These constants Choose the RX channel used in mutual-cap mode.

typedef enum *_tsi_mutual_sense_boost_current* tsi_mutual_sense_boost_current_t

TSI sensitivity boost current settings.

These constants set the sensitivity boost current.

typedef enum *_tsi_mutual_tx_drive_mode* tsi_mutual_tx_drive_mode_t

TSI TX drive mode control.

These constants Choose the TX drive mode control setting.

typedef enum *_tsi_mutual_pmos_current_left* tsi_mutual_pmos_current_left_t

TSI Pmos current mirror selection on the left side.

These constants set the Pmos current mirror on the left side used in mutual-cap mode.

typedef enum *_tsi_mutual_pmos_current_right* tsi_mutual_pmos_current_right_t

TSI Pmos current mirror selection on the right side.

These constants set the Pmos current mirror on the right side used in mutual-cap mode.

typedef enum *_tsi_mutual_nmos_current* tsi_mutual_nmos_current_t

TSI Nmos current mirror selection.

These constants set the Nmos current mirror used in mutual-cap mode.

typedef enum *_tsi_sinc_cutoff_div* tsi_sinc_cutoff_div_t

TSI SINC cutoff divider setting.

These bits set the SINC cutoff divider.

typedef enum *_tsi_sinc_filter_order* tsi_sinc_filter_order_t

TSI SINC filter order setting.

These bits set the SINC filter order.

typedef enum *_tsi_sinc_decimation_value* tsi_sinc_decimation_value_t

TSI SINC decimation value setting.

These bits set the SINC decimation value.

typedef enum *_tsi_ssc_charge_num* tsi_ssc_charge_num_t

TSI SSC output bit0's period setting(SSC0[CHARGE_NUM])

These bits set the SSC output bit0's period setting.

typedef enum *_tsi_ssc_nocharge_num* tsi_ssc_nocharge_num_t

TSI SSC output bit1's period setting(SSC0[BASE_NOCHARGE_NUM])

These bits set the SSC output bit1's period setting.

typedef enum *_tsi_ssc_prbs_outsel* tsi_ssc_prbs_outsel_t

TSI SSC outsel choosing the length of the PRBS (Pseudo-RandomBinarySequence) method setting(SSC0[TSI_SSC0_PRBS_OUTSEL])

These bits set the SSC PRBS length.

typedef enum *_tsi_status_flags* tsi_status_flags_t

TSI status flags.

typedef enum *_tsi_interrupt_enable* tsi_interrupt_enable_t

TSI feature interrupt source.

typedef enum *_tsi_ssc_mode* tsi_ssc_mode_t

TSI SSC mode selection.

These constants set the SSC mode.

typedef enum *_tsi_ssc_prescaler* tsi_ssc_prescaler_t

TSI main clock selection.

These constants set select the divider ratio for the clock used for generating the SSC output bit.

typedef struct *_tsi_calibration_data* tsi_calibration_data_t

TSI calibration data storage.

typedef struct *_tsi_common_config* tsi_common_config_t

TSI common configuration structure.

This structure contains the common settings for TSI self-cap or mutual-cap mode, configurations including the TSI module main clock, sensing mode, DVOLT options, SINC and SSC configurations.

typedef struct *_tsi_selfCap_config* tsi_selfCap_config_t

TSI configuration structure for self-cap mode.

This structure contains the settings for the most common TSI self-cap configurations including the TSI module charge currents, sensitivity configuration and so on.

typedef struct *_tsi_mutualCap_config* tsi_mutualCap_config_t

TSI configuration structure for mutual-cap mode.

This structure contains the settings for the most common TSI mutual-cap configurations including the TSI module generator settings, sensitivity related current settings and so on.

const *clock_ip_name_t* s_tsiClock[]

TSI_Type *const s_tsiBases[]

uint32_t TSI_GetInstance(TSI_Type *base)

Get the TSI instance from peripheral base address.

Parameters

- base – TSI peripheral base address.

Returns

TSI instance.

void TSI_InitSelfCapMode(TSI_Type *base, const *tsi_selfCap_config_t* *config)

Initialize hardware to Self-cap mode.

Initialize the peripheral to the targeted state specified by parameter config, such as sets sensitivity adjustment, current settings.

Parameters

- base – TSI peripheral base address.
- config – Pointer to TSI self-cap configuration structure.

Returns

none

void TSI_InitMutualCapMode(TSI_Type *base, const *tsi_mutualCap_config_t* *config)

Initialize hardware to Mutual-cap mode.

Initialize the peripheral to the targeted state specified by parameter config, such as sets Vref generator setting, sensitivity boost settings, Pmos/Nmos settings.

Parameters

- base – TSI peripheral base address.
- config – Pointer to TSI mutual-cap configuration structure.

Returns

none

void TSI_Deinit(TSI_Type *base)

De-initialize hardware.

De-initialize the peripheral to default state.

Parameters

- base – TSI peripheral base address.

Returns

none

void TSI_GetSelfCapModeDefaultConfig(*tsi_selfCap_config_t* *userConfig)

Get TSI self-cap mode user configure structure. This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to a value:

```
userConfig->commonConfig.mainClock     = kTSI_MainClockSlection_0;
userConfig->commonConfig.mode          = kTSI_SensingModeSlection_Self;
userConfig->commonConfig.dvolt         = kTSI_DvoltOption_2;
userConfig->commonConfig.cutoff        = kTSI_SincCutoffDiv_1;
userConfig->commonConfig.order         = kTSI_SincFilterOrder_1;
userConfig->commonConfig.decimation    = kTSI_SincDecimationValue_8;
userConfig->commonConfig.chargeNum     = kTSI_SscChargeNumValue_3;
userConfig->commonConfig.prbsOutsel    = kTSI_SscPrbsOutsel_2;
userConfig->commonConfig.noChargeNum   = kTSI_SscNoChargeNumValue_2;
userConfig->commonConfig.ssc_mode      = kTSI_ssc_prbs_method;
userConfig->commonConfig.ssc_prescaler = kTSI_ssc_div_by_1;
userConfig->enableSensitivity          = true;
userConfig->enableShield               = false;
userConfig->xdn                        = kTSI_SensitivityXdnOption_1;
userConfig->ctrim                      = kTSI_SensitivityCtrimOption_7;
userConfig->inputCurrent               = kTSI_CurrentMultipleInputValue_0;
userConfig->chargeCurrent              = kTSI_CurrentMultipleChargeValue_1;
```

Parameters

- userConfig – Pointer to TSI user configure structure.

void TSI_GetMutualCapModeDefaultConfig(*tsi_mutualCap_config_t* *userConfig)

Get TSI mutual-cap mode default user configure structure. This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to a value:

```
userConfig->commonConfig.mainClock     = kTSI_MainClockSlection_1;
userConfig->commonConfig.mode          = kTSI_SensingModeSlection_Mutual;
userConfig->commonConfig.dvolt         = kTSI_DvoltOption_0;
userConfig->commonConfig.cutoff        = kTSI_SincCutoffDiv_1;
userConfig->commonConfig.order         = kTSI_SincFilterOrder_1;
userConfig->commonConfig.decimation    = kTSI_SincDecimationValue_8;
userConfig->commonConfig.chargeNum     = kTSI_SscChargeNumValue_4;
userConfig->commonConfig.prbsOutsel    = kTSI_SscPrbsOutsel_2;
userConfig->commonConfig.noChargeNum   = kTSI_SscNoChargeNumValue_5;
userConfig->commonConfig.ssc_mode      = kTSI_ssc_prbs_method;
userConfig->commonConfig.ssc_prescaler = kTSI_ssc_div_by_1;
userConfig->preCurrent                 = kTSI_MutualPreCurrent_4uA;
userConfig->preResistor                = kTSI_MutualPreResistor_4k;
userConfig->senseResistor              = kTSI_MutualSenseResistor_10k;
userConfig->boostCurrent               = kTSI_MutualSenseBoostCurrent_0uA;
userConfig->txDriveMode                = kTSI_MutualTxDriveModeOption_0;
userConfig->pmosLeftCurrent            = kTSI_MutualPmosCurrentMirrorLeft_32;
userConfig->pmosRightCurrent           = kTSI_MutualPmosCurrentMirrorRight_1;
userConfig->enableNmosMirror           = true;
userConfig->nmosCurrent                = kTSI_MutualNmosCurrentMirror_1;
```

Parameters

- userConfig – Pointer to TSI user configure structure.

void TSI_SelfCapCalibrate(TSI_Type *base, *tsi_calibration_data_t* *calBuff)

Hardware base counter value for calibration.

Calibrate the peripheral to fetch the initial counter value of the enabled channels. This API is mostly used at initial application setup, it shall be called after the TSI_Init API, then user can use the calibrated counter values to setup applications(such as to determine

under which counter value we can confirm a touch event occurs).

Note: This API is mainly used for self-cap mode;

Note: The calibration work in mutual-cap mode shall be done in applications due to different board layout.

Parameters

- base – TSI peripheral base address.
- calBuff – Data buffer that store the calibrated counter value.

Returns
none

void TSI_EnableInterrupts(TSI_Type *base, uint32_t mask)

Enables TSI interrupt requests.

Parameters

- base – TSI peripheral base address.
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kTSI_GlobalInterruptEnable
  - kTSI_EndOfScanInterruptEnable
  - kTSI_OutOfRangeInterruptEnable

void TSI_DisableInterrupts(TSI_Type *base, uint32_t mask)

Disables TSI interrupt requests.

Parameters

- base – TSI peripheral base address.
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kTSI_GlobalInterruptEnable
  - kTSI_EndOfScanInterruptEnable
  - kTSI_OutOfRangeInterruptEnable

static inline uint32_t TSI_GetStatusFlags(TSI_Type *base)

Get interrupt flag. This function get tsi interrupt flags.

Parameters

- base – TSI peripheral base address.

Returns
The mask of these status flags combination.

void TSI_ClearStatusFlags(TSI_Type *base, uint32_t mask)

Clear interrupt flag.

This function clear tsi interrupt flag, automatically cleared flags can not be cleared by this function.

Parameters

- base – TSI peripheral base address.
- mask – The status flags to clear.

static inline uint32_t TSI_GetScanTriggerMode(TSI_Type *base)

Get TSI scan trigger mode.

Parameters

- base – TSI peripheral base address.

Returns

Scan trigger mode.

static inline void TSI_EnableModule(TSI_Type *base, bool enable)

Enables the TSI Module or not.

Parameters

- base – TSI peripheral base address.
- enable – Choose whether to enable or disable module;
  - true Enable TSI module;
  - false Disable TSI module;

Returns

none.

static inline void TSI_EnableLowPower(TSI_Type *base, bool enable)

Sets the TSI low power STOP mode enable or not. This enables TSI module function in low power modes.

Parameters

- base – TSI peripheral base address.
- enable – Choose to enable or disable STOP mode.
  - true Enable module in STOP mode;
  - false Disable module in STOP mode;

Returns

none.

static inline void TSI_EnableHardwareTriggerScan(TSI_Type *base, bool enable)

Enable the hardware trigger scan or not.

Parameters

- base – TSI peripheral base address.
- enable – Choose to enable hardware trigger or software trigger scan.
  - true Enable hardware trigger scan;
  - false Enable software trigger scan;

Returns

none.

static inline void TSI_StartSoftwareTrigger(TSI_Type *base)

Start one sotware trigger measurement (trigger a new measurement).

Parameters

- base – TSI peripheral base address.

Returns

none.

static inline void TSI_SetSelfCapMeasuredChannel(TSI_Type *base, uint8_t channel)

Set the measured channel number for self-cap mode.

Note: This API can only be used in self-cap mode!

Parameters

- base – TSI peripheral base address.
- channel – Channel number 0 ... 24.

Returns

none.

static inline uint8_t TSI_GetSelfCapMeasuredChannel(TSI_Type *base)

Get the current measured channel number, in self-cap mode.

Note: This API can only be used in self-cap mode!

Parameters

- base – TSI peripheral base address.

Returns

uint8_t Channel number 0 ... 24.

static inline void TSI_EnableEndOfScanDmaTransferOnly(TSI_Type *base, bool enable)

Decide whether to enable End of Scan DMA transfer request only.

Parameters

- base – TSI peripheral base address.
- enable – Choose whether to enable End of Scan DMA transfer request only.
  - true Enable End of Scan DMA transfer request only;
  - false Both End-of-Scan and Out-of-Range can generate DMA transfer request.

Returns

none.

static inline uint16_t TSI_GetCounter(TSI_Type *base)

Gets the conversion counter value.

Parameters

- base – TSI peripheral base address.

Returns

Accumulated scan counter value ticked by the reference clock.

static inline void TSI_SetLowThreshold(TSI_Type *base, uint16_t low_threshold)

Set the TSI wake-up channel low threshold.

Parameters

- base – TSI peripheral base address.
- low_threshold – Low counter threshold.

Returns

none.

static inline void TSI_SetHighThreshold(TSI_Type *base, uint16_t high_threshold)

Set the TSI wake-up channel high threshold.

Parameters

- base – TSI peripheral base address.
- high_threshold – High counter threshold.

Returns

none.

static inline void TSI_SetMainClock(TSI_Type *base, *tsi_main_clock_selection_t* mainClock)

Set the main clock of the TSI module.

Parameters

- base – TSI peripheral base address.
- mainClock – clock option value.

Returns

none.

static inline void TSI_SetSensingMode(TSI_Type *base, *tsi_sensing_mode_selection_t* mode)

Set the sensing mode of the TSI module.

Parameters

- base – TSI peripheral base address.
- mode – Mode value.

Returns

none.

static inline *tsi_sensing_mode_selection_t* TSI_GetSensingMode(TSI_Type *base)

Get the sensing mode of the TSI module.

Parameters

- base – TSI peripheral base address.

Returns

Currently selected sensing mode.

static inline void TSI_SetDvolt(TSI_Type *base, *tsi_dvolt_option_t* dvolt)

Set the DVOLT settings.

Parameters

- base – TSI peripheral base address.
- dvolt – The voltage rails.

Returns

none.

static inline void TSI_EnableNoiseCancellation(TSI_Type *base, bool enableCancellation)

Enable self-cap mode noise cancellation function or not.

Parameters

- base – TSI peripheral base address.
- enableCancellation – Choose whether to enable noise cancellation in self-cap mode
  - true Enable noise cancellation;
  - false Disable noise cancellation;

Returns

none.

static inline void TSI_SetMutualCapTxChannel(TSI_Type *base, *tsi_mutual_tx_channel_t* txChannel)

Set the mutual-cap mode TX channel.

Parameters

- base – TSI peripheral base address.
- txChannel – Mutual-cap mode TX channel number

Returns

none.

static inline *tsi_mutual_tx_channel_t* TSI_GetTxMutualCapMeasuredChannel(TSI_Type *base)

Get the current measured TX channel number, in mutual-cap mode.

Note: This API can only be used in mutual-cap mode!

Parameters

- base – TSI peripheral base address;

Returns

Tx Channel number 0 ... 5;

static inline void TSI_SetMutualCapRxChannel(TSI_Type *base, *tsi_mutual_rx_channel_t* rxChannel)

Set the mutual-cap mode RX channel.

Parameters

- base – TSI peripheral base address.
- rxChannel – Mutual-cap mode RX channel number

Returns

none.

static inline *tsi_mutual_rx_channel_t* TSI_GetRxMutualCapMeasuredChannel(TSI_Type *base)

Get the current measured RX channel number, in mutual-cap mode.

Note: This API can only be used in mutual-cap mode!

Parameters

- base – TSI peripheral base address;

Returns
Rx Channel number 6 ... 11;

static inline void TSI_SetSscMode(TSI_Type *base, *tsi_ssc_mode_t* mode)
Set the SSC clock mode of the TSI module.

Parameters

- base – TSI peripheral base address.
- mode – SSC mode option value.

Returns
none.

static inline void TSI_SetSscPrescaler(TSI_Type *base, *tsi_ssc_prescaler_t* prescaler)
Set the SSC prescaler of the TSI module.

Parameters

- base – TSI peripheral base address.
- prescaler – SSC prescaler option value.

Returns
none.

static inline void TSI_SetUsedTxChannel(TSI_Type *base, *tsi_mutual_tx_channel_t* txChannel)
Set used mutual-cap TX channel.

Parameters

- base – TSI peripheral base address.
- txChannel – Mutual-cap mode TX channel number

Returns
none.

static inline void TSI_ClearUsedTxChannel(TSI_Type *base, *tsi_mutual_tx_channel_t* txChannel)
Clear used mutual-cap TX channel.

Parameters

- base – TSI peripheral base address.
- txChannel – Mutual-cap mode TX channel number

Returns
none.

FSL_TSI_DRIVER_VERSION
TSI driver version.

ALL_FLAGS_MASK
TSI status flags macro collection.

struct _tsi_calibration_data
#include <fsl_tsi_v6.h> TSI calibration data storage.

Public Members

uint16_t calibratedData[1]
TSI calibration data storage buffer

struct __tsi__common__config

#include <fsl_tsi_v6.h> TSI common configuration structure.

This structure contains the common settings for TSI self-cap or mutual-cap mode, configurations including the TSI module main clock, sensing mode, DVOLT options, SINC and SSC configurations.

Public Members

tsi_main_clock_selection_t mainClock

Set main clock.

tsi_sensing_mode_selection_t mode

Choose sensing mode.

tsi_dvolt_option_t dvolt

DVOLT option value.

tsi_sinc_cutoff_div_t cutoff

Cutoff divider.

tsi_sinc_filter_order_t order

SINC filter order.

tsi_sinc_decimation_value_t decimation

SINC decimation value.

tsi_ssc_charge_num_t chargeNum

SSC High Width (t1), SSC output bit0's period setting.

tsi_ssc_prbs_outsel_t prbsOutsel

SSC High Random Width (t2), length of PRBS(Pseudo-RandomBinarySequence),SSC output bit2's period setting.

tsi_ssc_nocharge_num_t noChargeNum

SSC Low Width (t3), SSC output bit1's period setting.

tsi_ssc_mode_t ssc__mode

Clock mode selection (basic - from main clock by divider,advanced - using SSC(Switching Speed Clock) by three configurable intervals.

tsi_ssc_prescaler_t ssc__prescaler

Set clock divider for basic mode.

struct __tsi__selfCap__config

#include <fsl_tsi_v6.h> TSI configuration structure for self-cap mode.

This structure contains the settings for the most common TSI self-cap configurations including the TSI module charge currents, sensitivity configuration and so on.

Public Members

tsi_common_config_t commonConfig

Common settings.

bool enableSensitivity

Enable sensitivity boost of self-cap or not.

tsi_shield_t enableShield

Enable shield of self-cap mode or not.

tsi_sensitivity_xdn_option_t xdn
Sensitivity XDN option.

tsi_sensitivity_ctrim_option_t ctrim
Sensitivity CTRIM option.

tsi_current_multiple_input_t inputCurrent
Input current multiple.

tsi_current_multiple_charge_t chargeCurrent
Charge/Discharge current multiple.

struct _tsi_mutualCap_config
#include <fsl_tsi_v6.h> TSI configuration structure for mutual-cap mode.

This structure contains the settings for the most common TSI mutual-cap configurations including the TSI module generator settings, sensitivity related current settings and so on.

Public Members

tsi_common_config_t commonConfig
Common settings.

tsi_mutual_pre_current_t preCurrent
Vref generator current.

tsi_mutual_pre_resistor_t preResistor
Vref generator resistor.

tsi_mutual_sense_resistor_t senseResistor
I-sense generator resistor.

tsi_mutual_sense_boost_current_t boostCurrent
Sensitivity boost current setting.

tsi_mutual_tx_drive_mode_t txDriveMode
TX drive mode control setting.

tsi_mutual_pmos_current_left_t pmosLeftCurrent
Pmos current mirror on the left side.

tsi_mutual_pmos_current_right_t pmosRightCurrent
Pmos current mirror on the right side.

bool enableNmosMirror
Enable Nmos current mirror setting or not.

tsi_mutual_nmos_current_t nmosCurrent
Nmos current mirror setting.

2.121 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.

2.122 UTICK: MictoTick Timer Driver

void UTICK__Init(UTICK_Type *base)
Initializes an UTICK by turning its bus clock on.

void UTICK__Deinit(UTICK_Type *base)
Deinitializes a UTICK instance.

This function shuts down Utick bus clock

Parameters

- base – UTICK peripheral base address.

uint32_t UTICK__GetStatusFlags(UTICK_Type *base)
Get Status Flags.

This returns the status flag

Parameters

- base – UTICK peripheral base address.

Returns
status register value

void UTICK_ClearStatusFlags(UTICK_Type *base)

Clear Status Interrupt Flags.

This clears intr status flag

Parameters

- base – UTICK peripheral base address.

Returns

none

void UTICK_SetTick(UTICK_Type *base, *utick_mode_t* mode, uint32_t count, *utick_callback_t* cb)

Starts UTICK.

This function starts a repeat/onetime countdown with an optional callback

Parameters

- base – UTICK peripheral base address.
- mode – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
- count – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
- cb – UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

Returns

none

void UTICK_HandleIRQ(UTICK_Type *base, *utick_callback_t* cb)

UTICK Interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in UTICK_SetTick()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – UTICK peripheral base address.
- cb – callback scheduled for this instance of UTICK

Returns

none

FSL_UTICK_DRIVER_VERSION

UTICK driver version 2.0.5.

enum _utick_mode

UTICK timer operational mode.

Values:

enumerator kUTICK_Onetime

Trigger once

enumerator kUTICK_Repeat

Trigger repeatedly

typedef enum *_utick_mode* utick_mode_t

UTICK timer operational mode.

typedef void (*utick_callback_t)(void)

UTICK callback function.

2.123 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.

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.

2.124 WUU: Wakeup Unit driver

void WUU_SetExternalWakeUpPinsConfig(WUU_Type *base, uint8_t pinIndex, const *wuu_external_wakeup_pin_config_t* *config)

Enables and Configs External WakeUp Pins.

This function enables/disables the external pin as wakeup input. What's more this function configs pins options, including edge detection wakeup event and operate mode.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin. See Reference Manual for the details.
- config – Pointer to wuu_external_wakeup_pin_config_t structure.

void WUU_ClearExternalWakeupPinsConfig(WUU_Type *base, uint8_t pinIndex)

Disable and clear external wakeup pin settings.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin.

static inline uint32_t WUU_GetExternalWakeUpPinsFlag(WUU_Type *base)

Gets External Wakeup pin flags.

This function return the external wakeup pin flags.

Parameters

- base – WUU peripheral base address.

Returns
Wakeup flags for all external wakeup pins.

static inline void WUU_ClearExternalWakeUpPinsFlag(WUU_Type *base, uint32_t mask)

Clears External WakeUp Pin flags.

This function clears external wakeup pins flags based on the mask.

Parameters

- base – WUU peripheral base address.
- mask – The mask of Wakeup pin index to be cleared.

void WUU_SetInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex, *wuu_internal_wakeup_module_event_t* event)

Config Internal modules' event as the wake up soures.

This function configs the internal modules event as the wake up sources.

Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – Select interrupt or DMA/Trigger of the internal module as the wake up source.

void WUU_ClearInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex, *wuu_internal_wakeup_module_event_t* event)

Disable an on-chip internal modules' event as the wakeup sources.

Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – The event(interrupt or DMA/trigger) of the internal module to disable.

void WUU_SetPinFilterConfig(WUU_Type *base, uint8_t filterIndex, const *wuu_pin_filter_config_t* *config)

Configs and Enables Pin filters.

This function configs Pin filter, including pin select, filer operate mode filer wakeup event and filter edge detection.

Parameters

- base – WUU peripheral base address.
- filterIndex – The index of the pin filer.
- config – Pointer to wuu_pin_filter_config_t structure.

bool WUU_GetPinFilterFlag(WUU_Type *base, uint8_t filterIndex)

Gets the pin filter configuration.

This function gets the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index, which starts from 1.

Returns
True if the flag is a source of the existing low-leakage power mode.

void WUU_ClearPinFilterFlag(WUU_Type *base, uint8_t filterIndex)

Clears the pin filter configuration.

This function clears the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

bool WUU_GetExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0. return True if the specific pin is a wakeup source.

void WUU_ClearExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0.

FSL_WUU_DRIVER_VERSION

Defines WUU driver version 2.4.1.

enum _wuu_external_pin_edge_detection

External WakeUp pin edge detection enumeration.

Values:

enumerator kWUU_ExternalPinDisable
External input Pin disabled as wake up input.

enumerator kWUU_ExternalPinRisingEdge
External input Pin enabled with the rising edge detection.

enumerator kWUU_ExternalPinFallingEdge
External input Pin enabled with the falling edge detection.

enumerator kWUU_ExternalPinAnyEdge
External input Pin enabled with any change detection.

enum _wuu_external_wakeup_pin_event

External input wake up pin event enumeration.

Values:

enumerator kWUU_ExternalPinInterrupt
External input Pin configured as interrupt.

enumerator kWUU_ExternalPinDMARequest
External input Pin configured as DMA request.

enumerator kWUU_ExternalPinTriggerEvent
External input Pin configured as Trigger event.

enum _wuu_external_wakeup_pin_mode

External input wake up pin mode enumeration.

Values:

enumerator kWUU_ExternalPinActiveDSPD

External input Pin is active only during Deep Sleep/Power Down Mode. NOTE: This enumerations has been deprecated, please switch to kWUU_ExternalPinActiveLowLeakage.

enumerator kWUU_ExternalPinActiveLowLeakageMode

External input Pin is active only during low-leakage power modes.

enumerator kWUU_ExternalPinActiveAlways

External input Pin is active during all power modes.

enum _wuu_internal_wakeup_module_event

Internal module wake up event enumeration.

Values:

enumerator kWUU_InternalModuleInterrupt

Internal modules' interrupt as a wakeup source.

enumerator kWUU_InternalModuleDMATrigger

Internal modules' DMA/Trigger as a wakeup source.

enum _wuu_filter_edge

Pin filter edge enumeration.

Values:

enumerator kWUU_FilterDisabled

Filter disabled.

enumerator kWUU_FilterPosedgeEnable

Filter posedge detect enabled.

enumerator kWUU_FilterNegedgeEnable

Filter negedge detect enabled.

enumerator kWUU_FilterAnyEdge

Filter any edge detect enabled.

enum _wuu_filter_event

Pin Filter event enumeration.

Values:

enumerator kWUU_FilterInterrupt

Filter output configured as interrupt.

enumerator kWUU_FilterDMARequest

Filter output configured as DMA request.

enumerator kWUU_FilterTriggerEvent

Filter output configured as Trigger event.

enum _wuu_filter_mode

Pin filter mode enumeration.

Values:

enumerator kWUU_FilterActiveDSPD

External input pin filter is active only during Deep Sleep/Power Down Mode. NOTE: This enumerations has been deprecated, please switch to kWUU_FilterActiveLowLeakage.

enumerator kWUU_FilterActiveLowLeakageMode

External input pin filter is active only during low-leakage power modes.

enumerator kWUU_FilterActiveAlways

External input Pin filter is active during all power modes.

typedef enum *_wuu_external_pin_edge_detection* wuu_external_pin_edge_detection_t

External WakeUp pin edge detection enumeration.

typedef enum *_wuu_external_wakeup_pin_event* wuu_external_wakeup_pin_event_t

External input wake up pin event enumeration.

typedef enum *_wuu_external_wakeup_pin_mode* wuu_external_wakeup_pin_mode_t

External input wake up pin mode enumeration.

typedef enum *_wuu_internal_wakeup_module_event* wuu_internal_wakeup_module_event_t

Internal module wake up event enumeration.

typedef enum *_wuu_filter_edge* wuu_filter_edge_t

Pin filter edge enumeration.

typedef enum *_wuu_filter_event* wuu_filter_event_t

Pin Filter event enumeration.

typedef enum *_wuu_filter_mode* wuu_filter_mode_t

Pin filter mode enumeration.

typedef struct *_wuu_external_wakeup_pin_config* wuu_external_wakeup_pin_config_t

External WakeUp pin configuration.

typedef struct *_wuu_pin_filter_config* wuu_pin_filter_config_t

Pin Filter configuration.

struct __wuu_external_wakeup_pin_config

#include <fsl_wuu.h> External WakeUp pin configuration.

Public Members

wuu_external_pin_edge_detection_t edge

External Input pin edge detection.

wuu_external_wakeup_pin_event_t event

External Input wakeup Pin event

wuu_external_wakeup_pin_mode_t mode

External Input wakeup Pin operate mode.

struct __wuu_pin_filter_config

#include <fsl_wuu.h> Pin Filter configuration.

Public Members

uint32_t pinIndex

The index of wakeup pin to be muxxed into filter.

wuu_filter_edge_t edge
The edge of the pin digital filter.

wuu_filter_event_t event
The event of the filter output.

wuu_filter_mode_t mode
The mode of the filter operate.

2.125 WWDT: Windowed Watchdog Timer Driver

void WWDT_GetDefaultConfig(*wwdt_config_t* *config)
Initializes WWDT configure structure.

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

```
config->enableWwdt = true;
config->enableWatchdogReset = false;
config->enableWatchdogProtect = false;
config->enableLockOscillator = false;
config->windowValue = 0xFFFFFFU;
config->timeoutValue = 0xFFFFFFU;
config->warningValue = 0;
```

See also:

wwdt_config_t

Parameters

- config – Pointer to WWDT config structure.

void WWDT_Init(WWDT_Type *base, const *wwdt_config_t* *config)
Initializes the WWDT.

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
wwdt_config_t config;
WWDT_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
WWDT_Init(wwdt_base,&config);
```

Parameters

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

void WWDT_Deinit(WWDT_Type *base)
Shuts down the WWDT.

This function shuts down the WWDT.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Enable(WWDT_Type *base)

Enables the WWDT module.

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Disable(WWDT_Type *base)

Disables the WWDT module.

Deprecated:

Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

- base – WWDT peripheral base address

static inline uint32_t WWDT_GetStatusFlags(WWDT_Type *base)

Gets all WWDT status flags.

This function gets all status flags.

Example for getting Timeout Flag:

```
uint32_t status;
status = WWDT_GetStatusFlags(wwdt_base) & kWWDT_TimeoutFlag;
```

Parameters

- base – WWDT peripheral base address

Returns

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

void WWDT_ClearStatusFlags(WWDT_Type *base, uint32_t mask)

Clear WWDT flag.

This function clears WWDT status flag.

Example for clearing warning flag:

```
WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
```

Parameters

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

static inline void WWDT_SetWarningValue(WWDT_Type *base, uint32_t warningValue)

Set the WWDT warning value.

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

- base – WWDT peripheral base address
- warningValue – WWDT warning value.

static inline void WWDT_SetTimeoutValue(WWDT_Type *base, uint32_t timeoutCount)

Set the WWDT timeout value.

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is TWDCLK*256*4. If enableWatchdogProtect flag is true in wwdt_config_t config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

- base – WWDT peripheral base address
- timeoutCount – WWDT timeout value, count of WWDT clock tick.

static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)

Sets the WWDT window value.

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

- base – WWDT peripheral base address
- windowValue – WWDT window value.

void WWDT_Refresh(WWDT_Type *base)

Refreshes the WWDT timer.

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WWDT peripheral base address

FSL_WWDT_DRIVER_VERSION

Defines WWDT driver version.

WWDT_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WWDT_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

enum _wwdt_status_flags_t

WWDT status flags.

This structure contains the WWDT status flags for use in the WWDT functions.

Values:

enumerator kWWDT_TimeoutFlag

Time-out flag, set when the timer times out

enumerator kWWDT_WarningFlag

Warning interrupt flag, set when timer is below the value WDWARNINT

typedef struct *_wwdt_config* wwdt_config_t

Describes WWDT configuration structure.

struct _wwdt_config

#include <fsl_wwdt.h> Describes WWDT configuration structure.

Public Members

bool enableWwdt

Enables or disables WWDT

bool enableWatchdogReset

true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset

bool enableWatchdogProtect

true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time

bool enableLockOscillator

true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator

uint32_t windowValue

Window value, set this to 0xFFFFFF if windowing is not in effect

uint32_t timeoutValue

Timeout value

uint32_t warningValue

Watchdog time counter value that will generate a warning interrupt. Set this to 0 for no warning

uint32_t clockFreq_Hz

Watchdog clock source frequency.

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	
Hexagon	
RISC-V	
Xtensa	
Generate Integration Test	

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

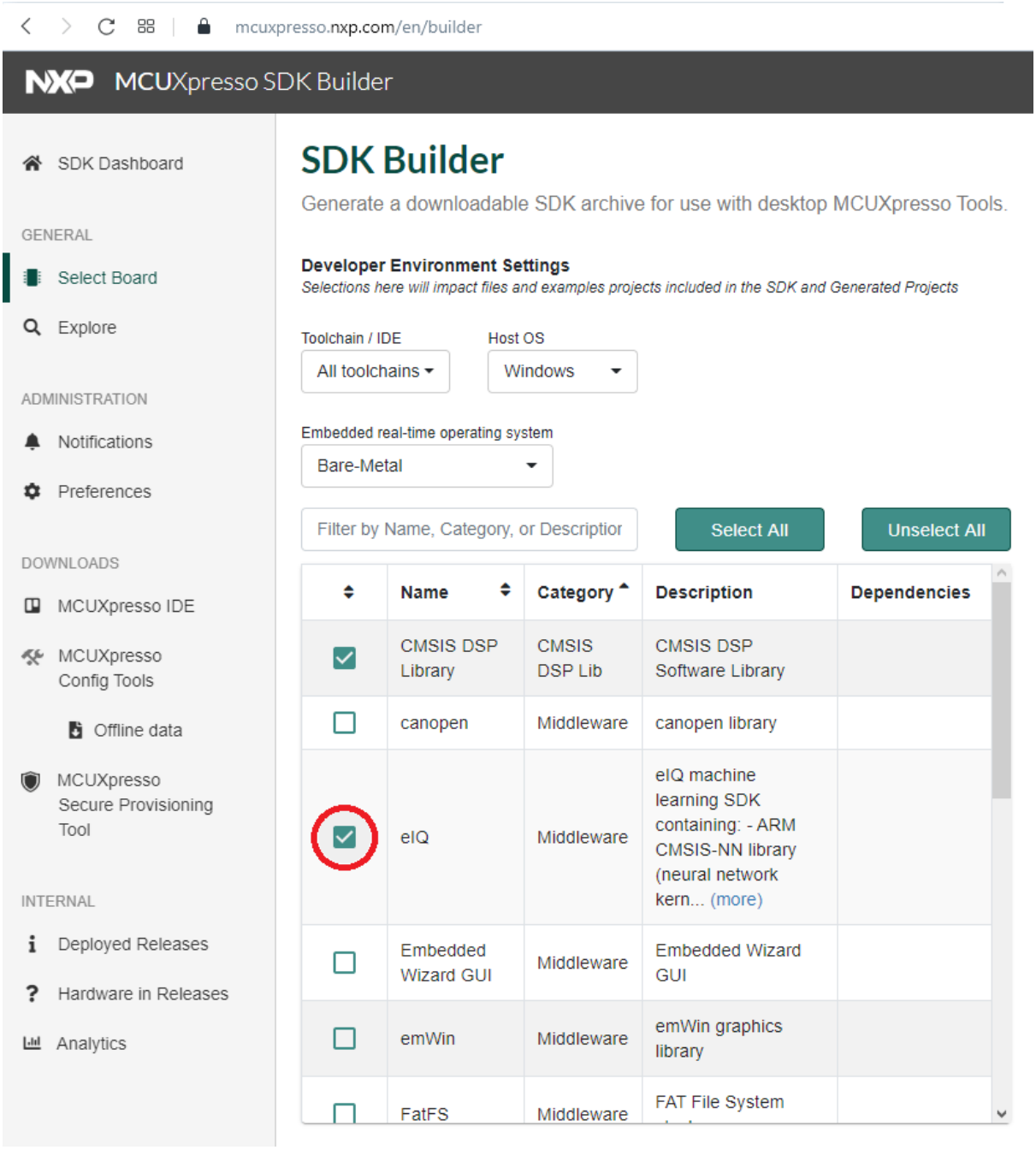
- *Continuous Integration*
- *Benchmarks*
- *Profiling*
- *Memory Management*
- *Logging*
- *Porting Reference Kernels from TfLite to TFLM*
- *Optimized Kernel Implementations*
- *New Platform Support*
- Platform/IP support
  - *Arm IP support*
- *Software Emulation with Renode*
- *Software Emulation with QEMU*

- *Python Dev Guide*
- *Automatically Generated Files*
- *Python Interpreter Guide*

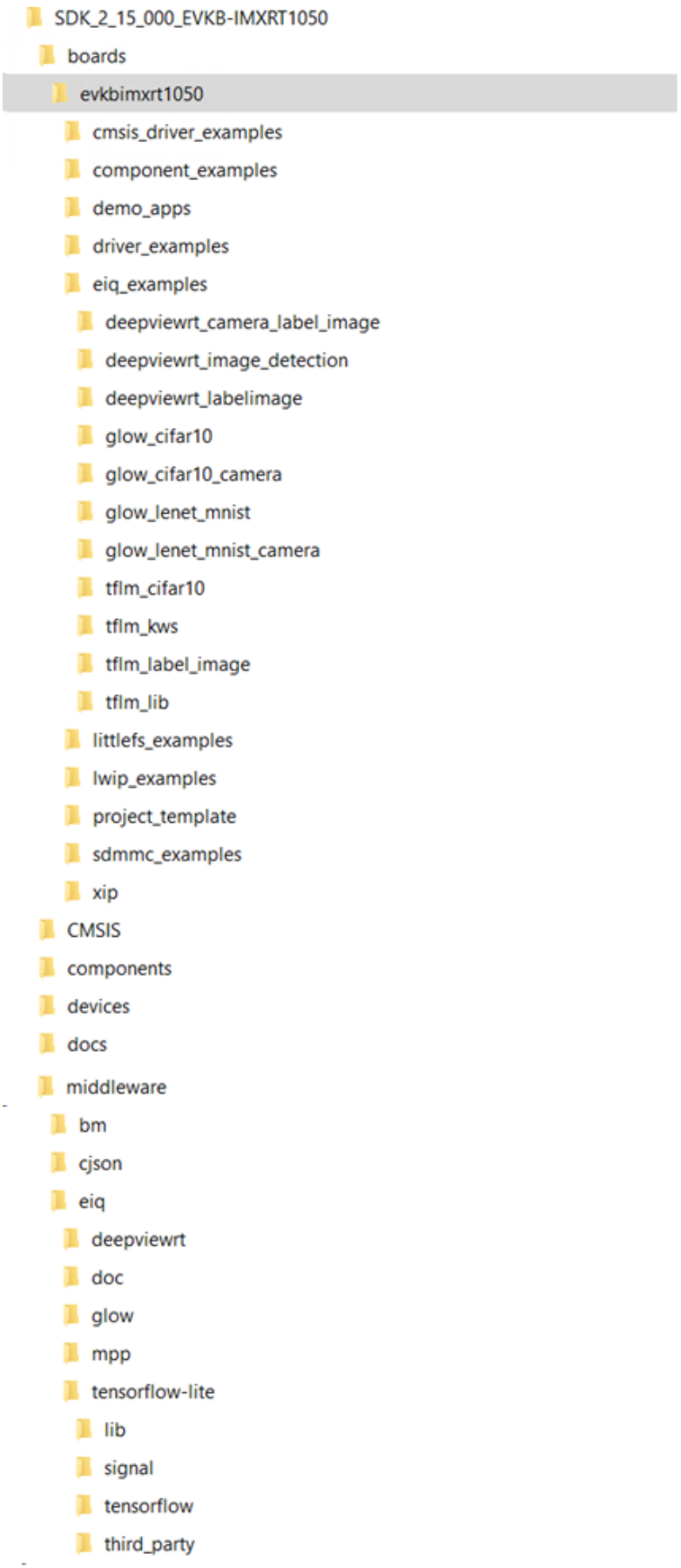
RFCs

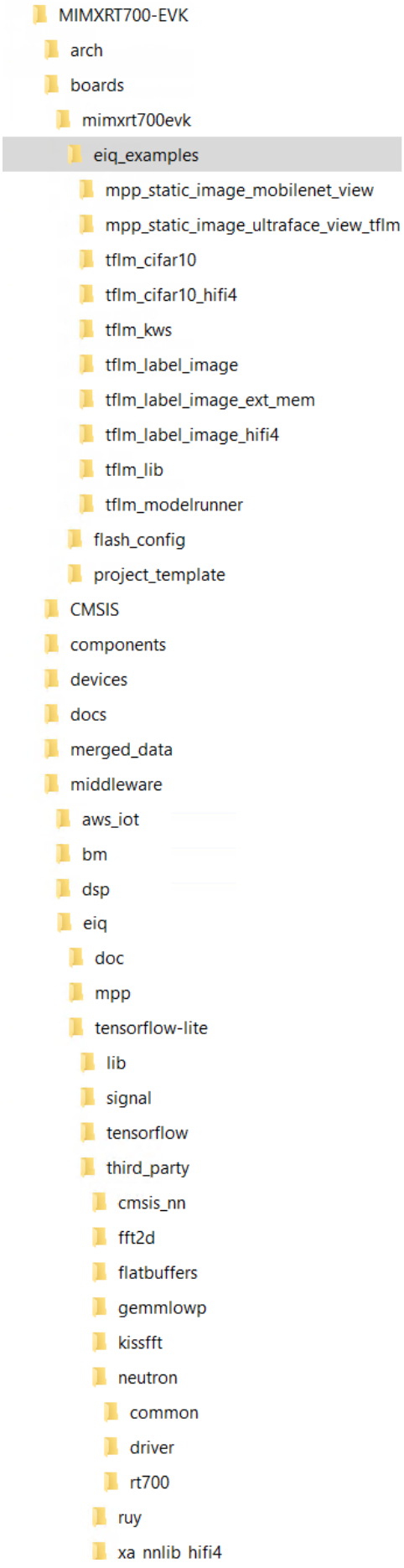
1. *Pre-allocated tensors*
2. *TensorFlow Lite for Microcontrollers Port of 16x8 Quantized Operators*

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

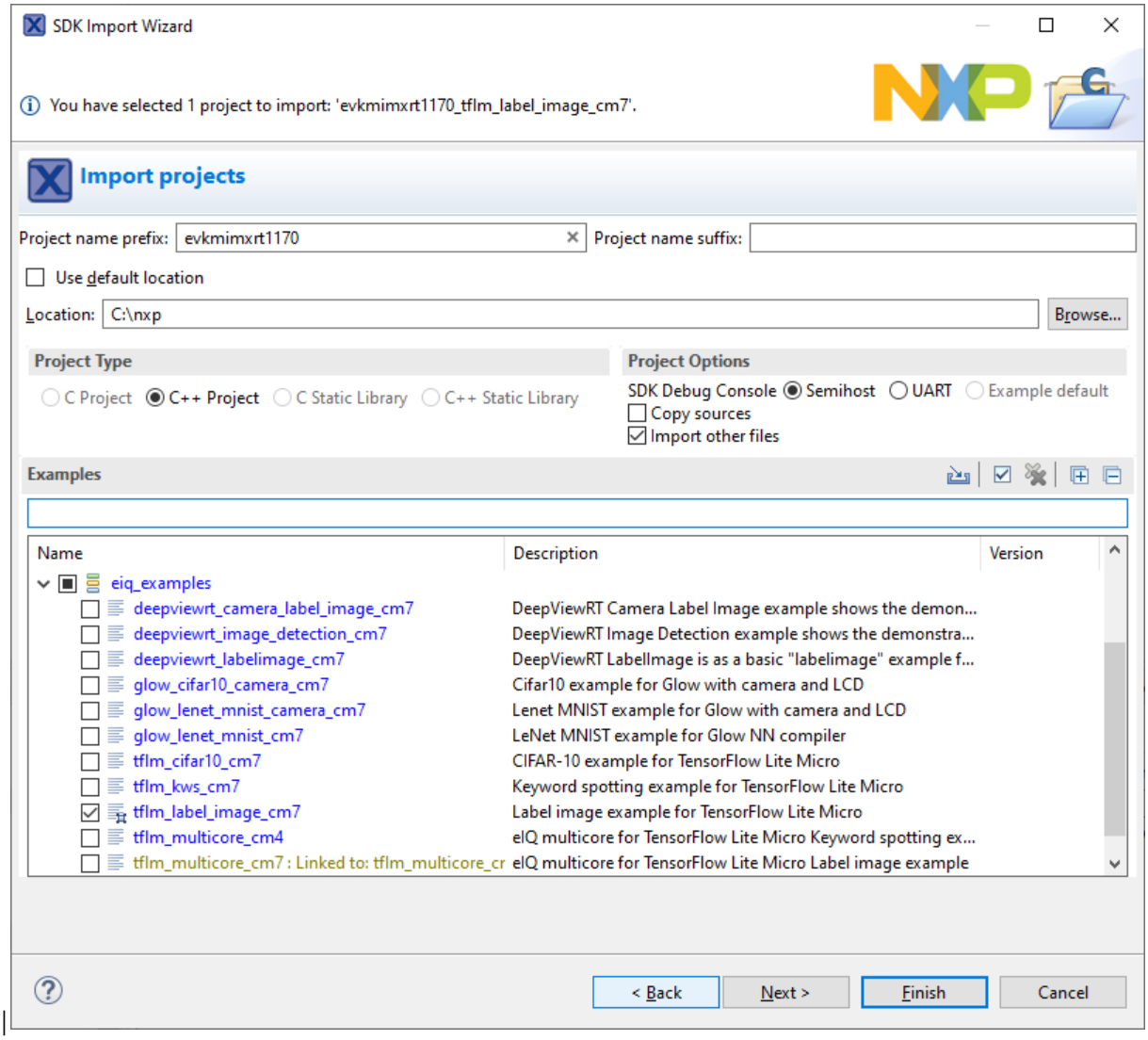
|

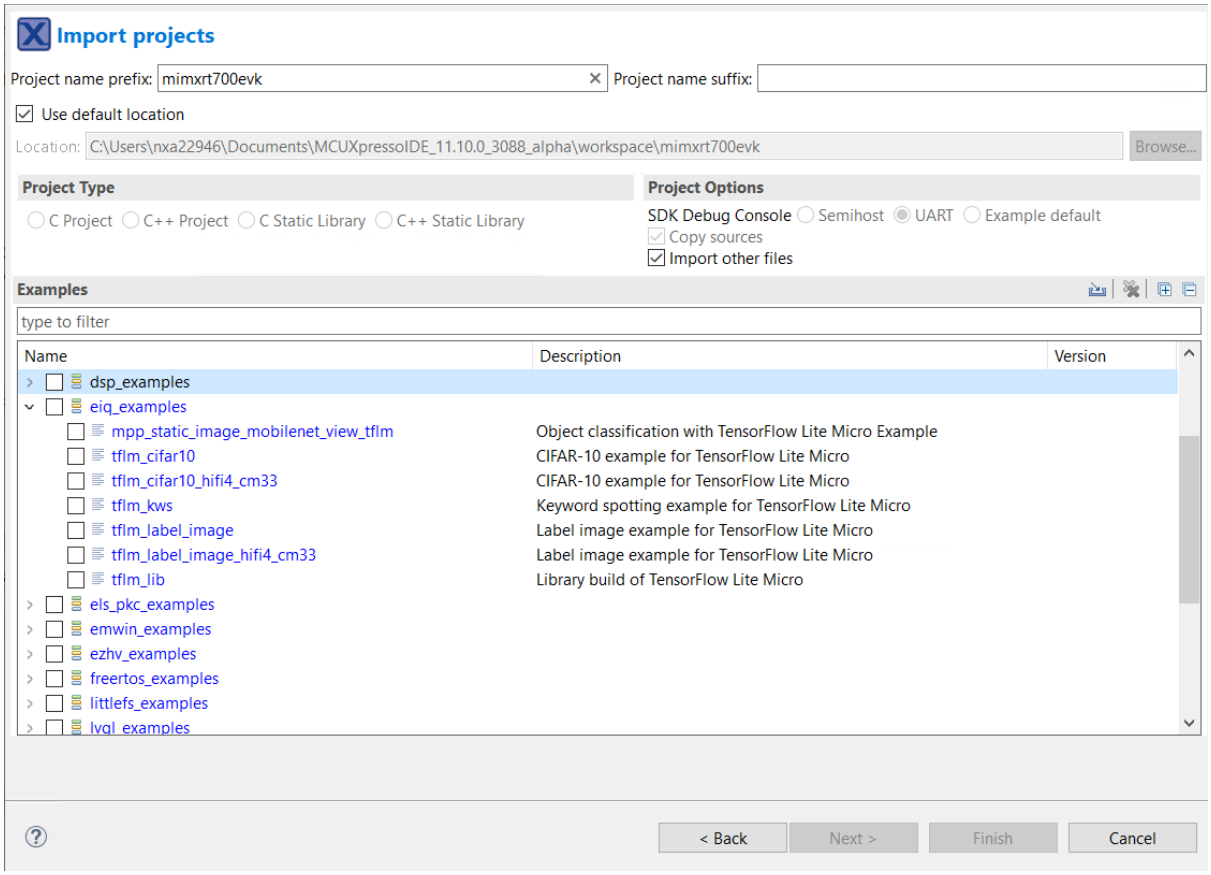
The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

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.

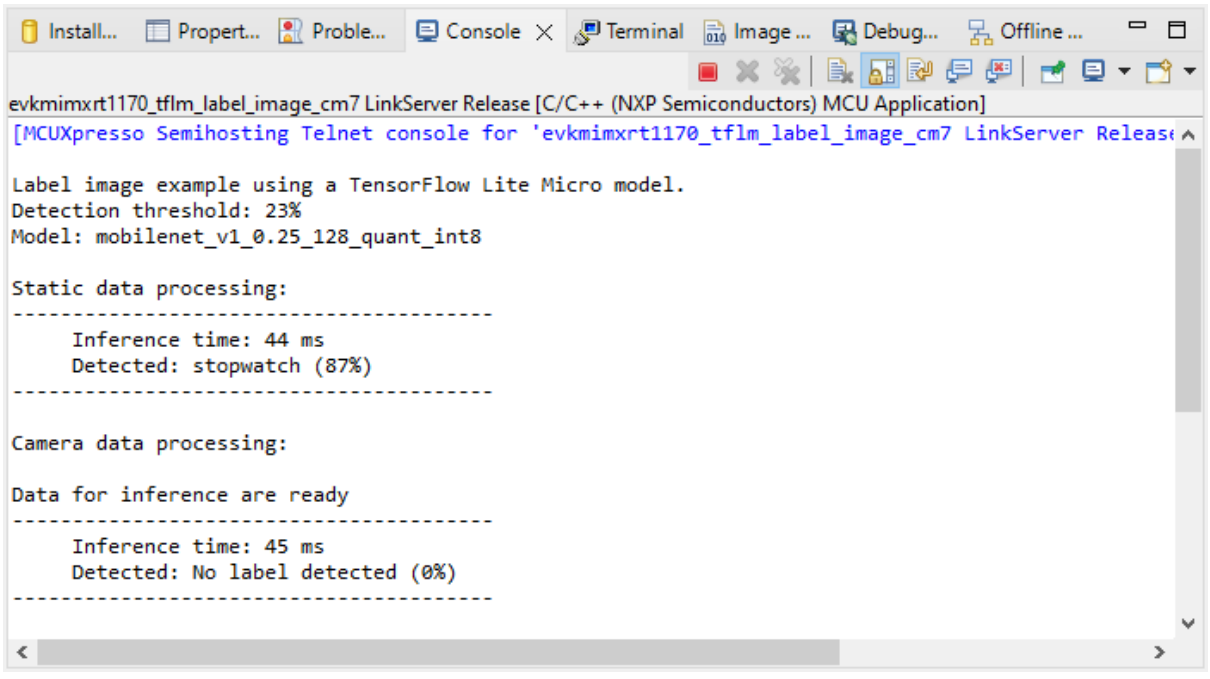
Name	Description	Availability
tflm_c	CIFAR-10 classification of 32 × 32 RGB pixel images into 10 categories using a small Convolutional Neural Network (CNN).	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (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.

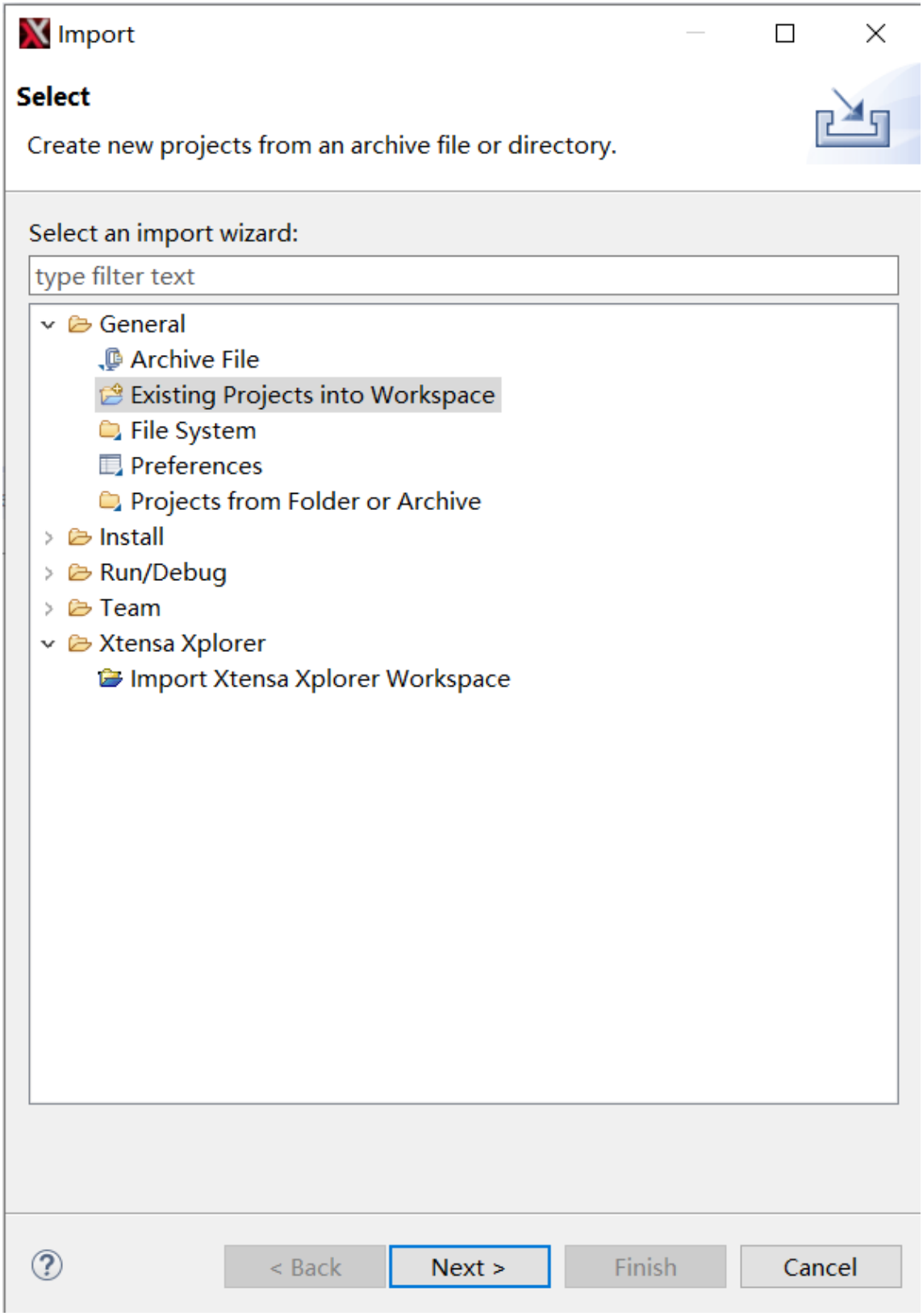
Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

Prepare DSP core for the examples

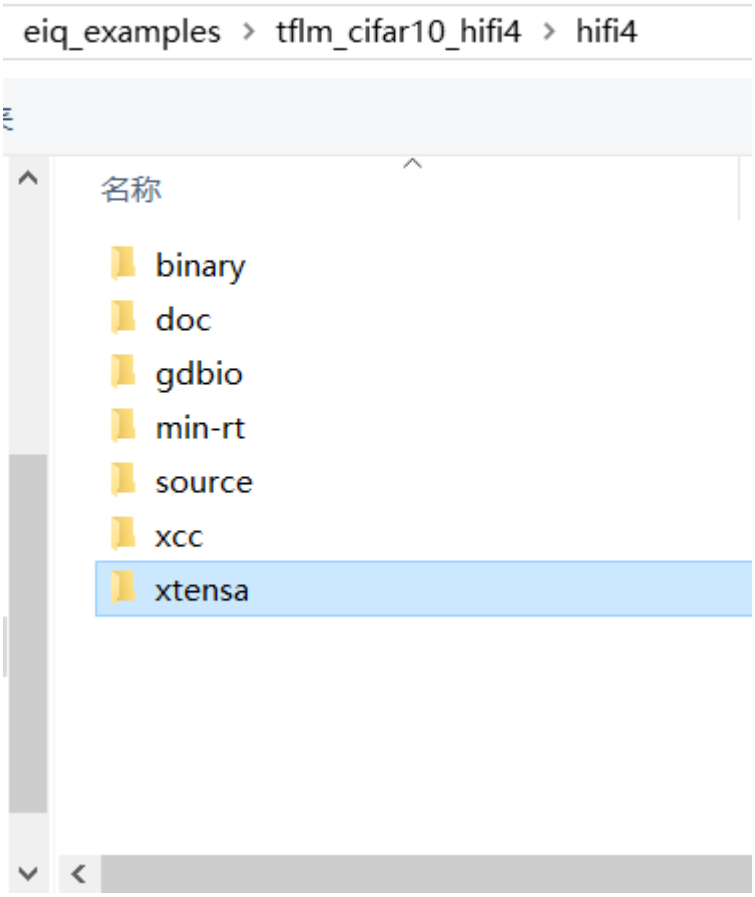
The projects for different supported toolchains are built. The “xcc” project builds on the command line and the “xtensa” directory is an Xplorer IDE project.

To run the tflm_cifar10_hifi4 example, import the SDK sources into the Xplorer IDE.

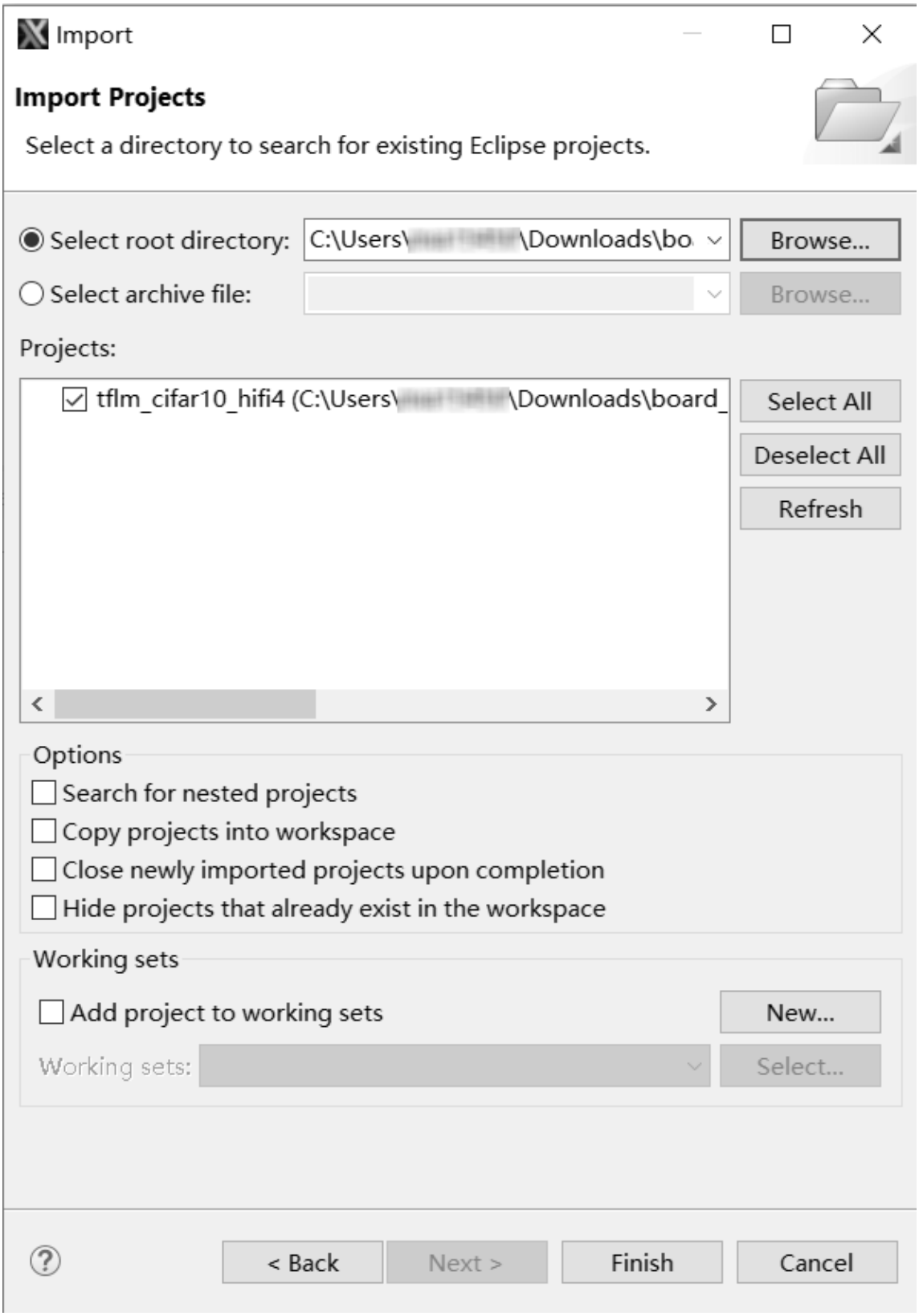
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Click **Next**.
3. Select the SDK directory/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/xtensa as the root directory.

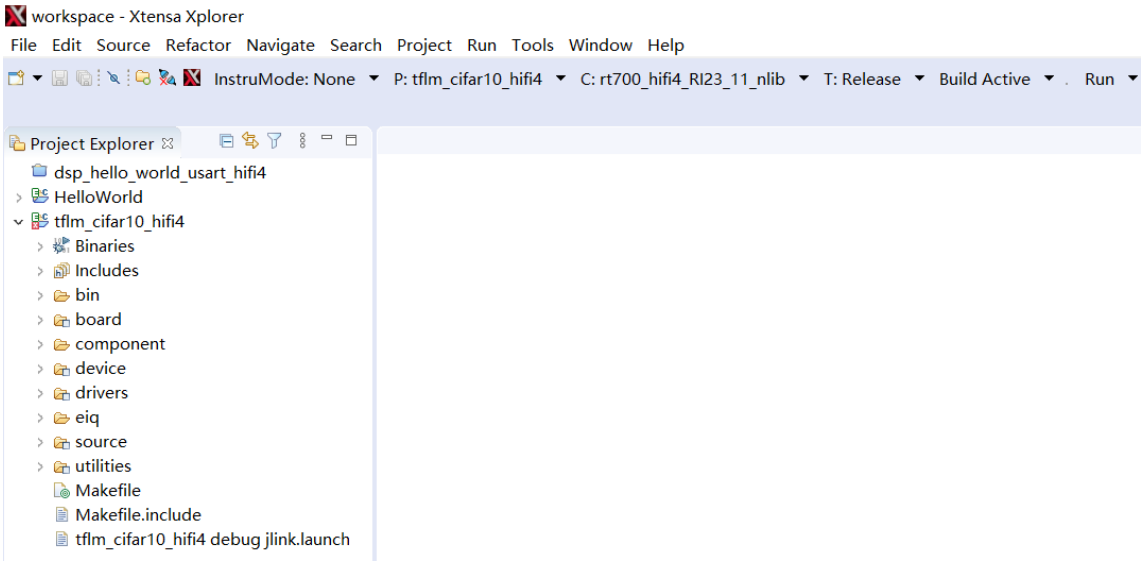


4. Click **Select Folder**.
5. Leave all the other options check boxes blank.

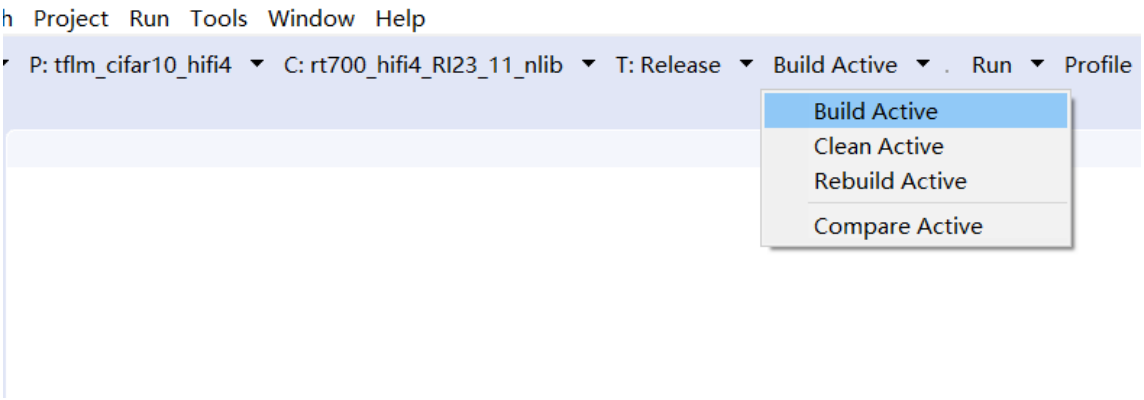


Once imported, the tflm_cifar10_hifi4 example appears in the **Project Explorer**.

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.

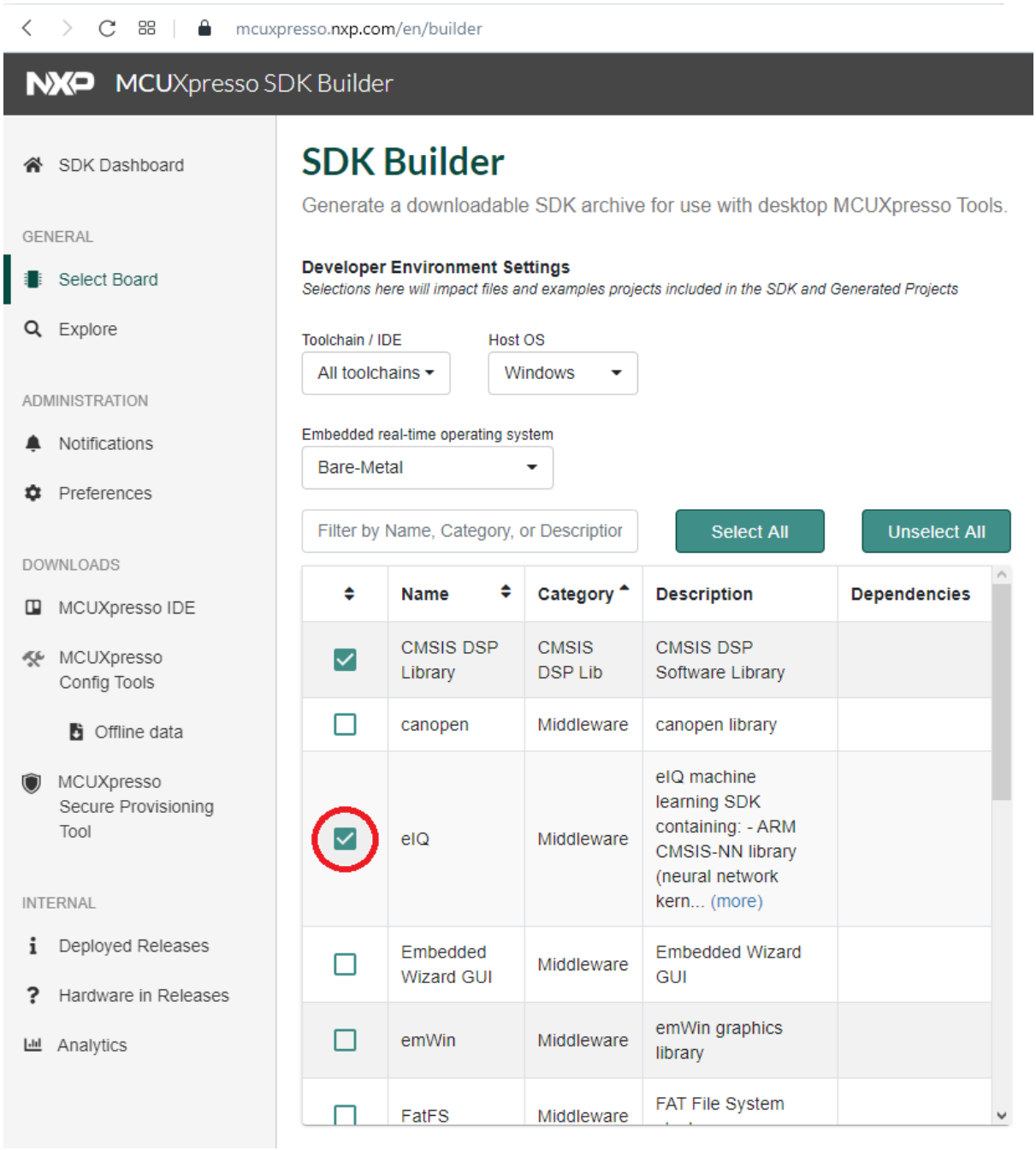
You can obtain the MCUXpresso SDK from MCUXpresso SDK Builder including the IDE. See the *getting_mcuxpress* for details.

In the MCUXpresso SDK, there are 2 projects available related to ExecuTorch:

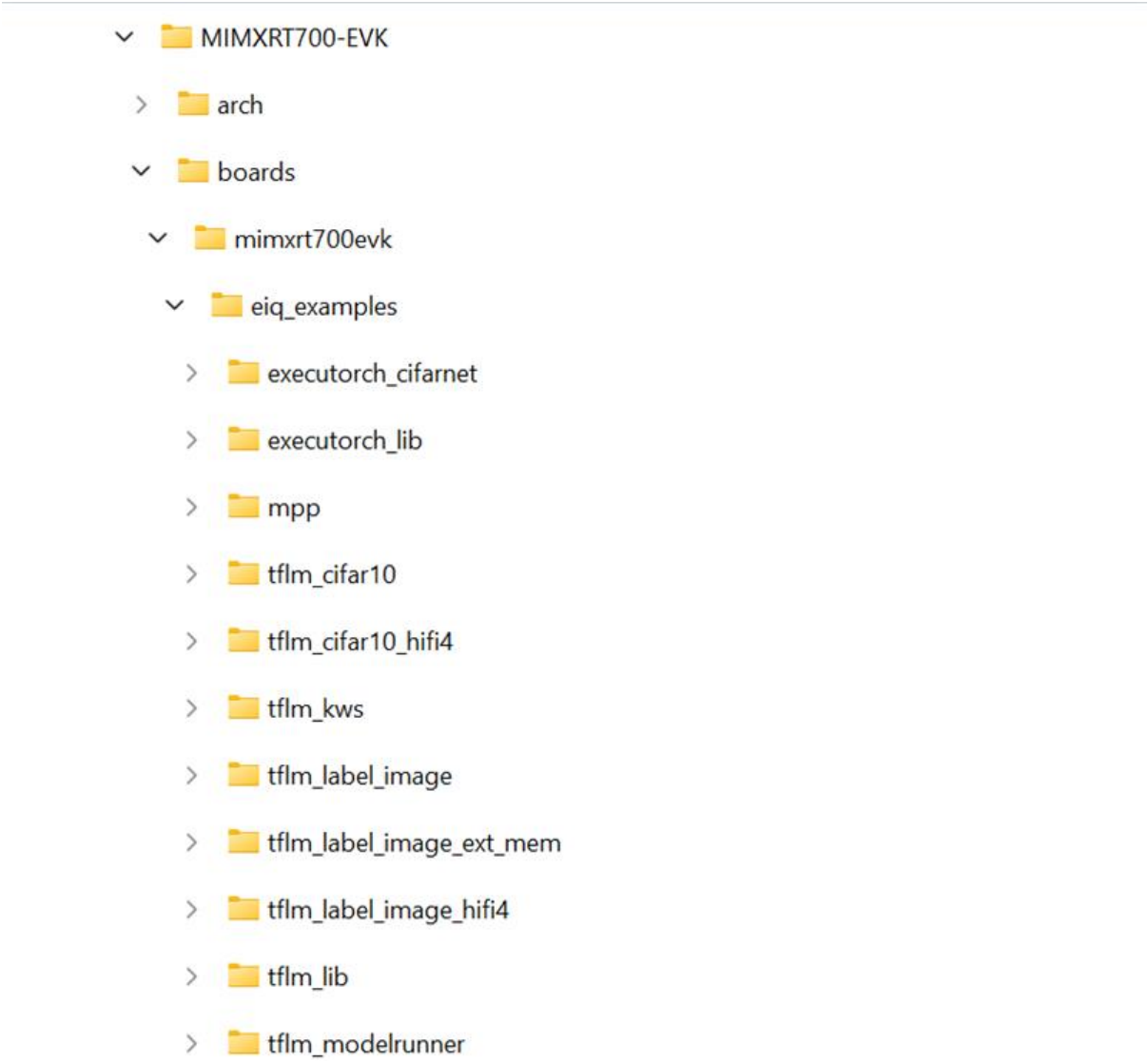
- executorch_lib
- executorch_cifarnet

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

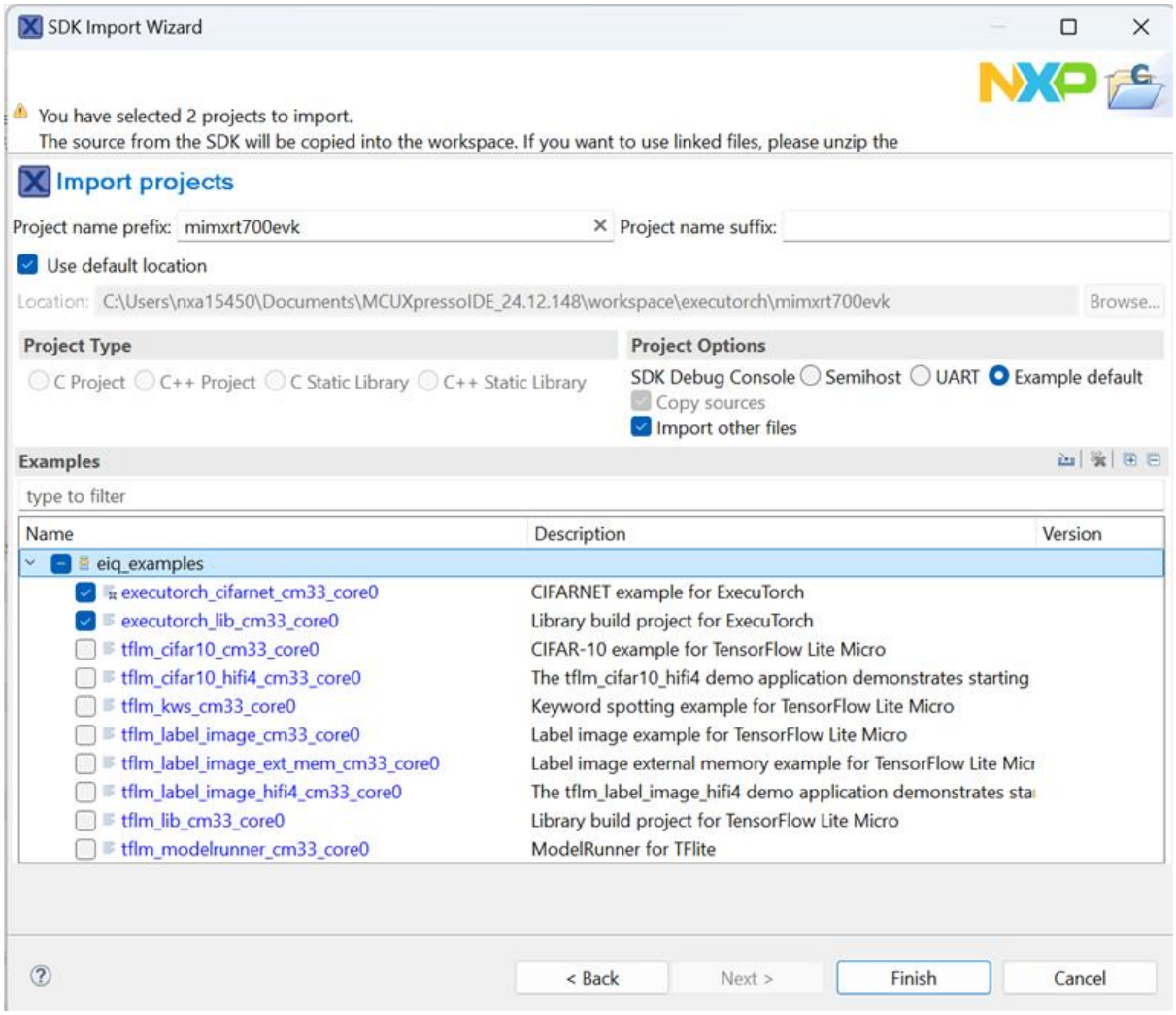
1. When using ARMGCC toolchain, the example application can be built as below. After building the example application, download it to the target with JLink as shown in *Figure 3*, an output message displays on the connected terminal as *Figure 4*.

```
$ 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.
```

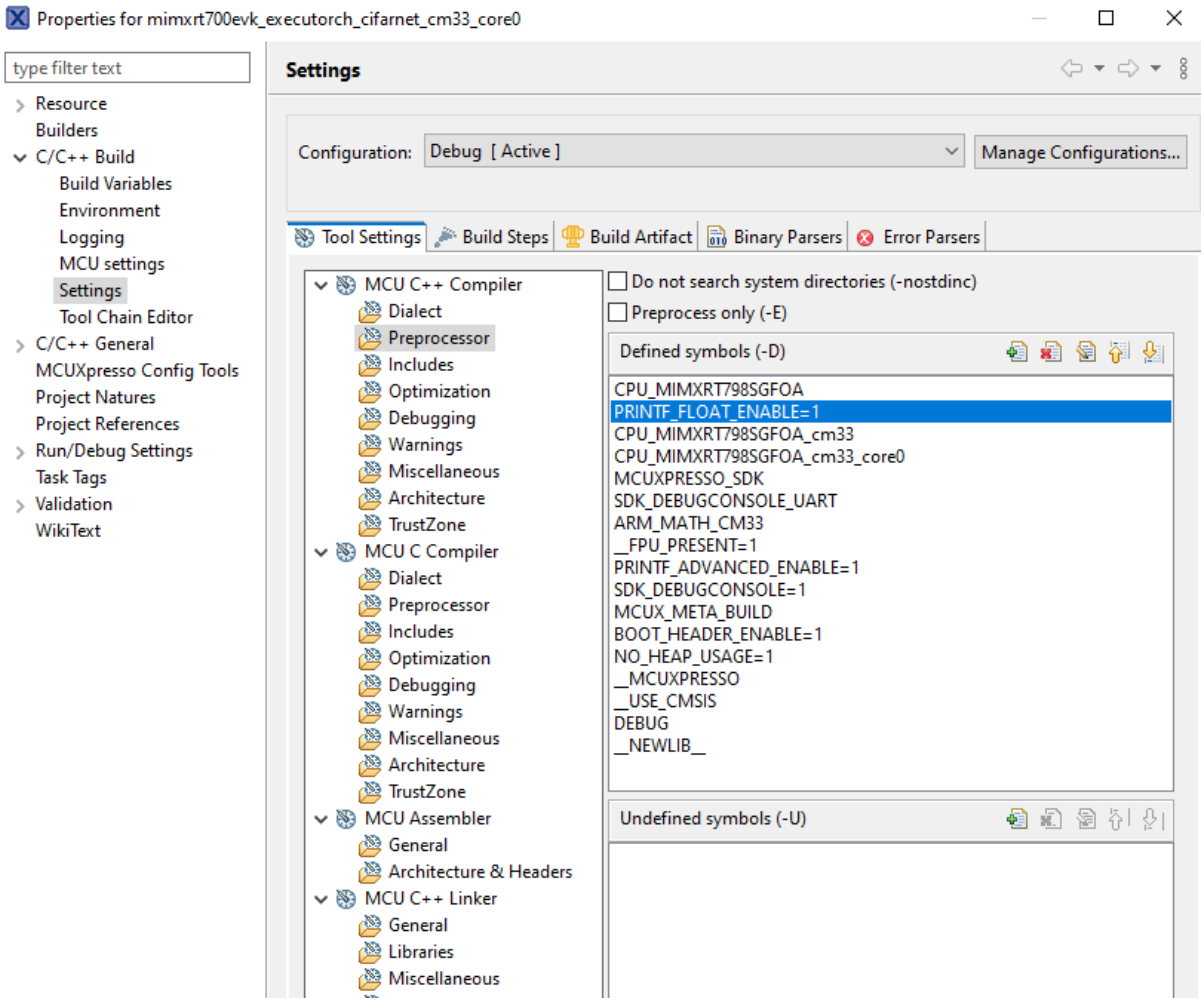
2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 5*.



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 Multimedia

3.4.1 Audio Voice

Maestro Audio Framework

MCUXpresso SDK : Maestro

Overview This repository is for MCUXpresso SDK maestro middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

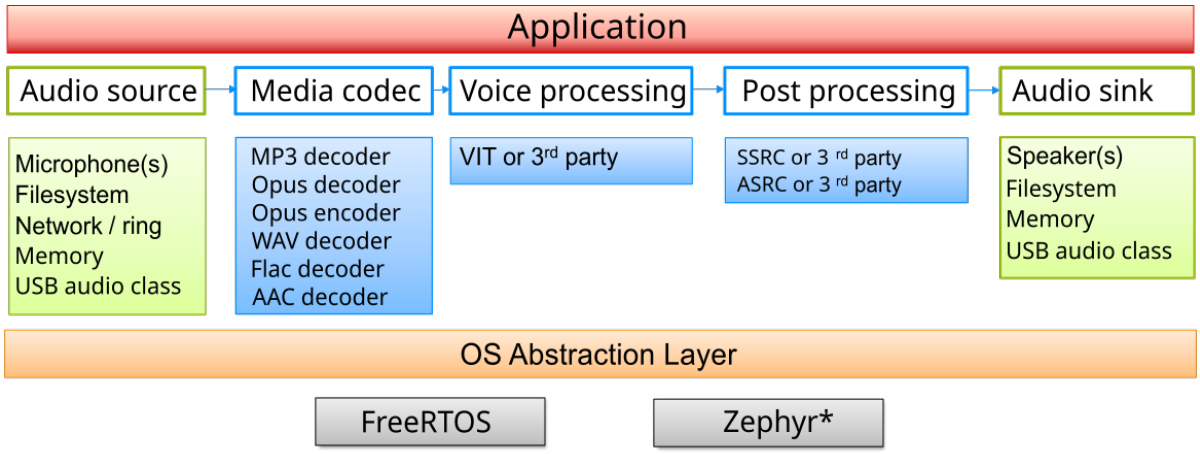
Visit Maestro - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution We welcome and encourage the community to submit patches directly to the Maestro project placed on github. Contributing can be managed via pull-requests.

Introduction Maestro audio framework intends to enable chaining of basic audio processing blocks, called *elements*. These blocks then form stream processing objects, called *pipeline*. This pipeline can be used for multiple audio processing use cases.

The processing blocks can include (but are not limited to) different audio sources (for example file or microphone), decoders or encoders, filters or effects, and audio sinks. Framework overview is depicted in the following picture:



*not all elements and libraries are supported in Zephyr port. For more information, see *Maestro on Zephyr*

The Maestro audio framework is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license. It is running on RTOS (Zephyr or FreeRTOS), abstracted by OSA layer.

For detailed description of the audio Maestro framework, please refer to the *programmer's guide*.

To see what is new, see *changelog*.

Maestro on Zephyr Getting started guide and further information for Maestro on Zephyr may be found *here*.

Maestro on FreeRTOS Maestro on FreeRTOS is supported in NXP's SDK. To get started, see mcuxsdk doc.

Supported examples The current version of the Maestro audio framework supports several optional *features*, some of which are used in these examples:

- *maestro_playback*
- *maestro_record*
- *maestro_usb_mic*
- *maestro_usb_speaker*

The examples can be found in the **audio_examples** folder of the desired board. The demo applications are based on FreeRTOS and use multiple tasks to form the application functionality.

Example applications overview To set up the audio framework properly, it is necessary to create a streamer with streamer_create API. It is also essential to set up the desired hardware peripherals using the functions described in streamer_pcm.h. The Maestro example projects consist of several files regarding the audio framework. The initial file is main.c with code to create multiple tasks. For features including SD card (in the maestro_playback examples, reading a file from SD card is supported and in maestro_record writing to SD card is currently supported) the APP_SDCARD_Task is created. The command prompt and connected functionalities are handled by APP_Shell_Task.

One of the most important parts of the configuration is the streamer_pcm.c where the initialization of the hardware peripherals, input and output buffer management can be found. For further information please see also streamer_pcm.h

In the Maestro USB examples (maestro_usb_mic and maestro_usb_speaker), the USB configuration is located in the usb_device_descriptor.c, audio_microphone.c and audio_speaker.c files. For further information please see also usb_device_descriptor.h, audio_microphone.h and audio_speaker.h.

In order to be able to get the messages from the audio framework, it is necessary to create a thread for receiving the messages from the streamer, which is usually called a Message Task. The message thread is placed in the app_streamer.c file, reads the streamer message queue, and reacts to the following messages:

- STREAM_MSG_ERROR - stops the streamer and exits the message thread
- STREAM_MSG_EOS - stops the streamer and exits the message thread
- STREAM_MSG_UPDATE_DURATION - prints info about the stream duration
- STREAM_MSG_UPDATE_POSITION - prints info about current stream position
- STREAM_MSG_CLOSE_TASK - exits the message thread

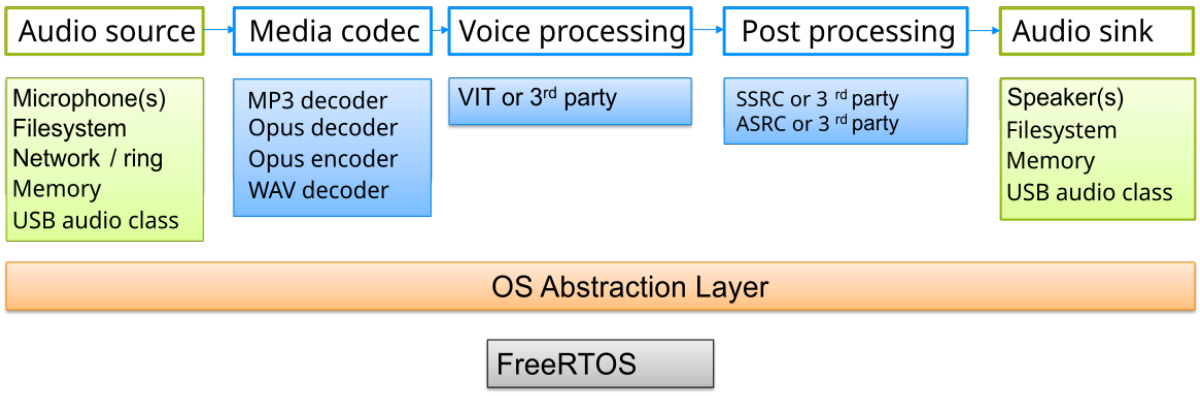
File structure

Folder	Description
src	Maestro audio framework sources
src/inc	Maestro include files
src/core	Maestro core sources
src/cci	Common decoder interface sources
src/cei	Common encoder interface sources
src/elements	Maestro elements sources
src/devices	External audio devices implementation (audio source & audio sink elements)
src/utils	Helper utilities utilized by Maestro
docs	Generated documentation
doxygen	Documentation sources
components	Glue for audio libraries, so they can be used in elements
tests	Maestro tests
zephyr/	Zephyr related files
zephyr/samples/	Zephyr samples
zephyr/tests/	Zephyr tests
zephyr/audioTracks/	Audio tracks for testing
zephyr/wrappers/	Zephyr NXP SDK Wrappers
zephyr/doc/	Zephyr documentation configuration for Sphinx
zephyr/scripts/	Zephyr helper scripts, mostly for testing

Maestro Audio Framework Programmer's Guide

Introduction Maestro audio framework provides instruments for playback and capture of different audio streams. In order to do that the framework uses API for creating various audio and voice pipelines with the support of media and track information. This document describes the framework in its detail, and the usage of API for pipeline creation using different elements. The framework needs an operating system in order to create different tasks for audio processing and communication with the application.

Architecture overview A high-level block diagram of the streamer used in Maestro is shown below. An element is the most important class of objects in the streamer (see streamer_element.c). A chain of elements will be created and linked together when a *pipeline* is created. Data flows through this chain of elements in form of data buffers. An element has one specific function, which can be the reading of data from a file, decoding of this data, or outputting this data to a sink device. By chaining together several such elements, a pipeline is created that can do a specific task, for example, the playback.



Pipeline

The pipeline is created within the streamer_create API using the streamer_create_pipeline call. In the example applications provided in the MCUXpresso SDK the pipeline is created in the app_streamer.c file. In order to create a pipeline user needs to provide a PipelineElements structure consisting of array of element indexes ElementIndex and the number of elements in the pipeline. Then the pipeline is built automatically and user can specify the properties of the elements using the streamer_set_property API. All the element properties can be found in the streamer_element_properties.h file.

The streamer can handle up to two pipelines within a single task. The first pipeline with index 0 can be created using the streamer_create function as described above. Then the streamer_create_pipeline function should be used to create the second pipeline (pipeline with index 1). Both pipelines are processed sequentially, so after the first pipeline is processed, the second pipeline is processed.

After the pipeline is sucessfully created, all elements and entire pipeline are in STATE_NULL state. A user can start the streamer by setting the pipeline state to STATE_PLAYING using the streamer_set_state function. The pipeline can also be paused or stopped using the same function. Use the STATE_PAUSED to pause and use STATE_NULL to stop. The function changes the state of each element that is in the pipeline in turn, and after all the elements have obtained the desired state, the state of entire pipeline is changed.

Elements The current version of the Maestro framework supports several types of elements (StreamElementType). In each pipeline should be used one source element (elements with the _SRC suffix) and one sink element (elements with the _SINK suffix). A decoder, encoder or audio_proc element can be connected between these two elements. The audio_proc element can be used more than once within the same pipeline.

Each element type (StreamElementType) has several functions that are determined by a unique element index (ElementIndex). These indexes are used to create a pipeline, and each element index can only be used once in the same pipeline. The type_lookup_table shows which StreamElementType supports which ElementIndex.

Each element index (ElementIndex) has its own properties and a list of these properties can be found in the streamer_element_properties.h file. These properties are divided into groups and each group is identified by a property mask (e.g. for speaker it is PROP_SPEAKER_MASK). Then the property_lookup_table in the streamer_msg.c file determines which property group relates to which element index (ElementIndex). When an element is created and added to the pipeline, its properties are set to their defalut values. Default values can be seen in the initialization function of a particular element. The initialization functions are specified in the element_list array in the streamer_element.c file (e.g. for the audio_proc element it is the audio_proc_init_element function). The user can get the value of the property using the streamer_get_property function or change its value using the streamer_set_property function.

The source code of the elements can be found in the middleware\audio_voice\maestro\src\elements\ folder.

Add a new element type The user can add a new element type (StreamElementType) to the Maestro audio framework. For this, the following steps need to be done.

- Add a new element type to the StreamElementType enum type in the streamer_api.h.
- Create a new *.c and *.h files for the new element type in the middleware\audio_voice\maestro\src\elements\ folder. All necessary structures and functions (functions for src pads, sink pads and element itself) needs to be defined in these files. Inspiration can be found in other elements.
- Link the initialization function to the element type in the element_list array in the streamer_element.c file. To do this, a new definition that enables the element needs to be created (e.g. there is a STREAMER_ENABLE_AUDIO_PROC definition for the audio_proc element).

- Associate the newly created element type with an element index (ElementIndex) by adding a new pair to the type_lookup_table in the streamer.c file.
- If the user wants to use the newly created element in an application, the definiton that enables the element must be defined at the project level.

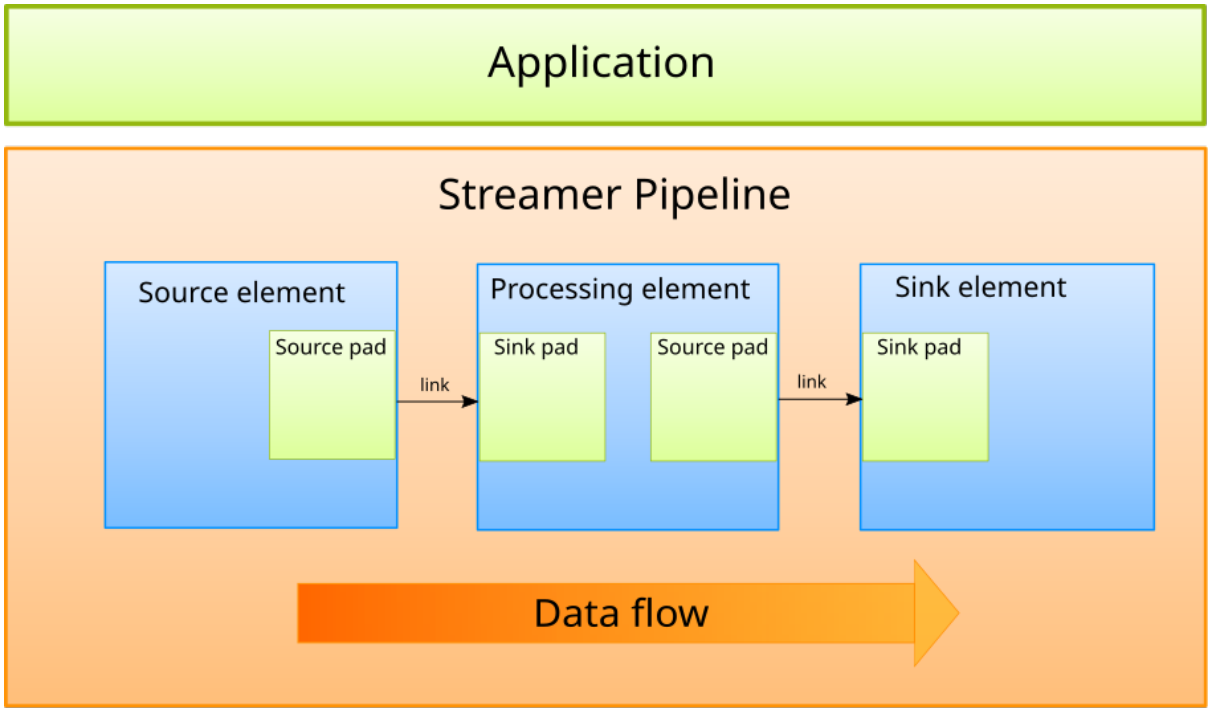
Mostly the user doesn't need to create a new element type, but just create an element index.

Add a new element index To create a new element index in the Maestro audio framework, follow these steps:

- Add a new element index to the ElementIndex enum type in the streamer_api.h.
- Create the required properties for the newly created element index in the streamer_element_properties.h file.
- Associate the newly created property group with newly created element index by adding a new pair to the property_lookup_table in the streamer_msg.c file.
- Associate the newly created element index with an element type (StreamElementType) by adding a new pair to the type_lookup_table in the streamer.c file.
- Add support for the created properties to functions of the associated element type. These functions are defined in files that correspond to a particular element type. The files are located in the middleware\audio_voice\maestro\src\elements\ folder.

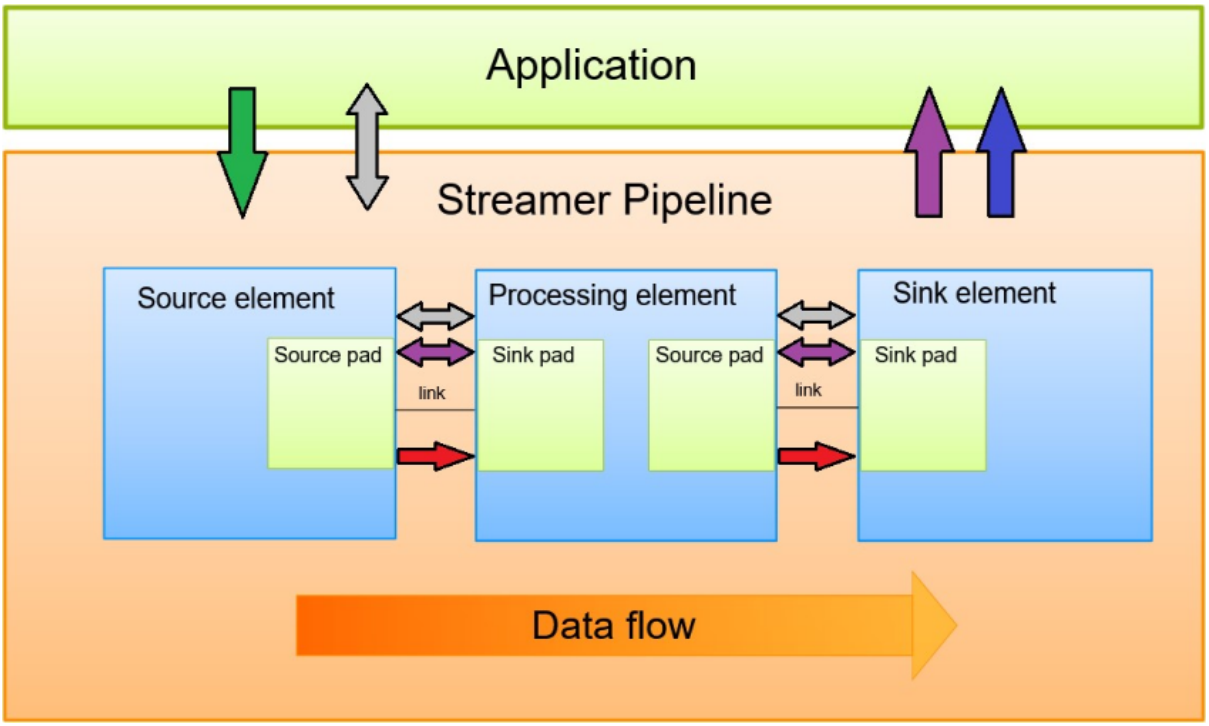
It is important to know that each element type (StreamElementType) can be associated with more than one element index (ElementIndex), but each element index (ElementIndex) can be associated with only one element type (StreamElementType).

Pads Pads are elements' inputs and outputs. A pad can be viewed as a "plug" or "port" on an element where links may be made with other elements, and through which data can flow to or from those elements. Data flows out of an element through a source pad, and elements accept incoming data through a sink pad. Source and sink elements have only source and sink pads, respectively. For detailed information about pads, please see the API reference from pad.c.



Internal communication The streamer (the core of the framework) provides several mechanisms for communication and data exchange between the application, a pipeline, and pipeline elements:

- Buffers are objects for passing streaming data between elements in the pipeline. Buffers always travel from sources to sinks (downstream).
- Messages are objects sent from the application to the streamer task to construct, configure, and control a streamer pipeline.
- Callbacks are used to transmit information such as errors, tags, state changes, etc. from the pipeline and elements to the application.
- Events are objects sent between elements. Events can travel upstream and downstream. Events may also be sent to the application
- Queries allow applications to request information such as duration or current playback position from the pipeline. Elements can also use queries to request information from their peer elements (such as the file size or duration). They can be used both ways within a pipeline, but upstream queries are more common



Decoders and encoders Maestro framework uses a common codec interface for decoding purposes and a common encoder interface for encoding. Those interfaces encapsulate the usage of specific codecs. Reference codecs are available in audio-voice-components repository which should be in \middleware\audio_voice\components\ folder.

Common codec interface The Common Codec Interface is the intended interface for all used **decoders**. The framework will integrate a CCI decoder element into the streamer to interface with all decoders.

Using the CCI to interface with Metadata

- cci_extract_meta_data must be called before any other Codec Interface APIs. This API extracts the metadata information of the codec and fills this information in the

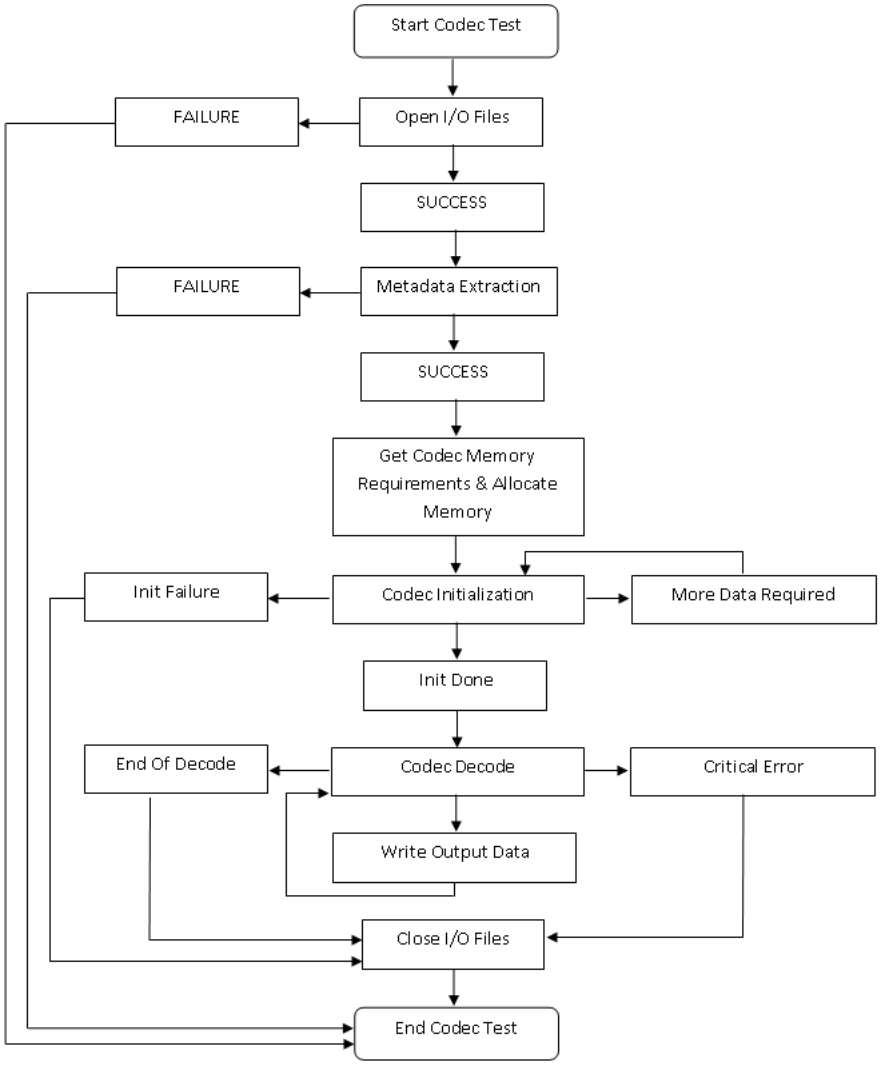
file_meta_data_t structure. The file_meta_data_t structure must be allocated by the application.

- This function first extracts the input file extension and based on that it calls the specific codec's metadata extraction function. If it finds an invalid extension or unsupported extension then it returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.
- If this API finds the valid metadata then it returns with META_DATA_FOUND code. If this API does not find any metadata information then it returns with META_DATA_NOT_FOUND code. It also returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.

Using the CCI to interface with Decoders

- codec_get_mem_info gets the memory requirement based on the specific decoder stream type. It returns the size in bytes of the specific codec. The user of the decoders must allocate memory of this size and this memory is used by the initialization API. The user or application must pass this allocated memory pointer to the init API.
- codec_init must be called before the codec's decode API. This API calls the codec-specific initialization function based on the codec stream type. This API allocates the memory to the codec main structure and also initializes the codec main structure parameters. It also registers the call back functions to the codec which will be used by the codec to read or seek the input stream.
- codec_decode is the main decoding API of the codec. This API calls the codec-specific decoding function based on the codec stream type. This API decodes the input raw stream and fills the PCM output samples into codec output PCM buffer. This API gives the information about the number of samples produced by the codec and also gives the pointer of the codec output PCM samples buffer.
- codec_get_pcm_samples must be called after the codec's decode API. This API calls the codec specific Get PCM Sample API based on the codec stream type. This API gets the PCM samples from the codec in constant block size and fills them into the output PCM buffer. It returns the number of samples get from the codec and also gives the pointer of the output PCM buffer.
- codec_reset calls the codec specific reset API base on stream type and resets the codec.
- codec_seek accepts the seek bytes offset converted from the time by application. This API calls the decoder's internal seek API to calculate the actual seek offset which frame boundary aligns. This API returns the actual seek offset.

The basic sequence to use a decoder with the CCI is shown below:



Adding new decoders to the CCI This section explains how to integrate a new decoder in the Common Codec Interface. The CCI assumes the decoder library to be used is in the \middleware\audio_voice\audiocomponents\decoders*decoder*\libs\ folder of the maestro framework. The CCI is just a wrapper around a specific implementation. The decoder is expected to be extended as needed to meet the APIs described above.

- Register Decoder Top level APIs in Common Codec Interface
  - Place the decoder lib in libs folder.
  - Add prototypes of the decoder top level APIs in codec_interface.h file (located at maestro\src\cci\inc\ folder).
  - In codec_interface.c file (located at maestro\src\cci\src\), add top level Decoder APIs in decoder function table.
  - Pseudo code for this is as described below.

```
const codec_interface_function_table_t g_codec_function_table[STREAM_TYPE_COUNT] = {
#ifdef VORBIS_CODEC
    {
        &VORBISDecoderGetMemorySize,
        &VORBISDecoderInit,
        &VORBISDecoderDecode,
        NULL,
```

(continues on next page)

(continued from previous page)

```
        NULL,
        &VORBISDecoderSeek,
        &VORBISDecoderGetIOFrameSize,
    },
#else
    {
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
    }
#endif
};
```

- Enable or Disable Decoder
  - Define VORBIS_CODEC macro in audio_cfg.h file.
  - Comment this macro if you want to disable VORBIS Decoder otherwise keep it defined in order to enable the decoder.
- Add Extract Metadata API for the decoder
  - Add extract metadata API source file for the decoder at streamer/cci/metadata/src/vorbis folder.
  - Add this code in extract metadata lib project space.
  - Build the extract metadata lib and copy that lib to libs folder.
  - Add the desired stream type into ccidec_extract_meta_data API (in codecextractmetadata.c file) to call VORBIS Decoder extract metadata API.
- Add stream type of the new decoder in the stream type enum audio_stream_type_t in codec_interface_public_api.h
  - Stream type of the decoder in stream type enum and decoder APIs in decoder function table must be in the same sequence.

Common encoder interface Please see the following section about the *cei*.

Maestro performance

Memory information The memory usage of the framework components using reference codecs (data obtained from GNU ARM compiler) in bytes is:

text	data	bss	component
48790	2752	4	aac decoder
4348	16400	212	asrc
15512	0	4	flac decoder
76462	16	5013	maestro
34211	0	4	mp3 decoder
211974	0	0	opus
65446	0	4	ssrc
5850	16	12	wav decoder

Maestro framework uses dynamic allocation of audio buffers. The total amount of memory allocated for the pipeline depends on the following parameters:

- Number of elements in the pipeline
- Element types
- Audio stream properties
  - Sampling rate
  - Bit width
  - Channel number
  - Frame size

CPU usage The performance of the pipeline was measured using the real hardware platform (RT1060).

- CPU core clock in MHz: 600.

Pipeline type	Performance MIPS of pipeline (in MHz)
audio source -> audio sink	~10.26 MHz
audio source -> file sink	~9.84 MHz
file source (8-channel PCM) -> audio sink	~16.5 MHz

For performance details about the supported codecs please see audio-voice-components repository documentation.

CEI encoder The Maestro streamer contains an element adapting an extensible set of audio encoders in the form of functions conforming to the CEI (Common Encoder Interface). This element enables the user to choose and configure a suitable encoder at runtime.

Header files CEI itself and the CEI encoders are using following header files, in which you may be interested:

- cei.h - contains types used by the element itself and an encoder implementing the CEI
- cei_enctypes.h - contains a list of possible encoders and types used for interfacing with a CEI encoder
- cei_table.h - contains a table of functions implementing integrated CEI encoders

Instantiating the element This element's index is ELEMENT_ENCODER_INDEX and its type is TYPE_ELEMENT_ENCODER, as defined in streamer_api.h. It has one source pad (data input) and one sink pad (data output). It is initialized like any other element, meaning that it is instantiated and inserted into the pipeline using the create_element, add_element_pipeline and link_elements functions. Inversely, for destroying the element, the unlink_elements, remove_element_pipeline and destroy_element are used. This element alone does not depend on any additional software layers other than these required by the Maestro streamer itself, so no pre-initialization before this element instantiation is necessary.

Element properties Use Maestro streamer property API (streamer_set_property and streamer_get_property) for setting or getting these. The constants are defined in streamer_element_properties.h.

- PROP_ENCODER_CHUNK_SIZE

  - **Synopsis:** Determines the length of a chunk pulled from the sibling of the source pad and essentially influences the size of allocated buffers. If the actual amount of data pulled is smaller, the rest is zero-filled.
  - **Type:** unsigned 32-bit integer
  - **Default value:** 1920
  - **Constraints:**
    * Must be bigger than zero, otherwise STREAM_ERR_INVALID_ARGS is returned.
    * Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS is returned.

- PROP_ENCODER_TYPE
  - **Synopsis:** Determines the exact encoder (CEI implementation) to be used.
  - **Type:** CeiEncoderType (cei_enctypes.h)
  - **Default value:** CEIENC_LAST
  - **Constraints:**
    * Must not be equal to CEIENC_LAST, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    * Selected encoder must be implemented, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    * Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS will be returned.
  - **Behaviour influenced:** The encoder element process function will return FLOW_ERROR if this property isn't set.

- PROP_ENCODER_CONFIG
  - **Synopsis:** Determines encoder-specific configuration (application, bitrate, ...).
  - **Type:** Pointer to the encoder-specific configuration structure.
  - **Default value:** Determined by the encoder.
  - **Constraints**:
    * The encoder has to be configurable. If it is not, STREAM_ERR_ERR_GENERAL will be returned on any access.
    * The structure has to conform to the encoder requirements. If the encoder returns an error code, STREAM_ERR_GENERAL will be returned.

- PROP_ENCODER_BITSTREAMINFO
  - **Synopsis:** Specifies information about the incoming bitstream (sample rate, sample depth, ...).
  - **Type:** Pointer to CeiBitstreamInfo (cei_enctypes.h).
  - **Default value:**

```
(CeiBitstreamInfo) {
    .sample_rate = 0,
    .num_channels = 0,
    .endian = AF_LITTLE_ENDIAN,
    .sign = TRUE,
    .sample_size = 0,
    .interleaved = TRUE
}
```

- **Constraints:**
  - Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS will be returned.
  - As of now, only bitstreams containing 16-bit interleaved (if 2 or more channels will be encoded) samples are supported. If anything else was set to the sample_size and interleaved members, STREAM_ERR_INVALID_ARGS will be returned.
- **Behaviour influenced:**
  - Given the characteristics of some elements available, different packets of data (header and payload, referred to as "chunk" above) may be pulled by this element. Each packet can contain a different header, which may or may not contain useful information about the bitstream. If a packet with the AudioPacketHeader (todofile.h) is pulled at first and any other iteration of the streamer pipeline, the bitstream parameters configured by this property are implicitly available and are not expected to be specified by the user. Other packet header types (such as RawPacketHeader) don't contain any bitstream parameters and require the user to specify the parameters manually using this property. Failure to do so will result in the element's process function returning FLOW_ERROR. Same situation will occur if a packet with the AudioPacketHeader is received and its contents differ from the already acquired bitstream parameters.
  - As of now, CEI is defined to work with 16-bit signed little-endian (s16le) samples, which are interleaved if the bitstream contains more than one channels. This element handles endianness and unsigned to signed conversion.

CEI definition - implementing your own encoder The CEI defines following function pointer types:

- CeiFnGetMemorySize: Returns number of bytes required for encoder state for a given number of channels.
- CeiFnEncoderInit: Initialize an encoder for a given sample rate and channel count.
- CeiFnEncoderGetConfig: Copy current or default configuration to a given structure pointer.
- CeiFnEncoderSetConfig: Configure the encoder from a given structure pointer.
- CeiFnEncode: Encode a given buffer to a given output buffer.

Detailed descriptions of function behaviour, parameters and expected return values are available as docblocks in the cei.h file.

Each encoder is implemented as a set of pointers pointing to functions conforming to these types, grouped in the CeiEncoderFunctions structure. Specifying the CeiEncoderGetConfig fnGetConfig and CeiFnEncoderSetConfig fnSetConfig members is optional, as an encoder does not have to be configurable. If so desired, specify NULL. Implementation of the remaining functions is mandatory, however. If at least one of these functions isn't implemented and NULL is specified instead, the encoder will be considered as not implemented.

To register an implemented encoder with the element, add a new entry to the CeiEncoderType enum and add the CeiEncoderFunctions struct value to the table CeiEncoderFunctions ceiEncTable[] located in the cei_table.h header file. Note and match the order of items in that table, as a CeiEncoderType value is used as an index. Same goes for the size_t ceiEncConfigSizeTable[]. If configuration is not applicable, specify 0 at the appropriate index. If configuration is applicable, describe the configuration structure in the cei_enctypes.h header file and add its size to that table.

Maestro playback example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*
- *Processing Time*

Overview The Maestro playback example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console and the audio files are read from the SD card.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback of encoded files from an SD card with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-channel** - The mode demonstrates playback of raw PCM files from an SD card with 2 or 8 channels, 96kHz sample rate and 32 bit width. The decoders and synchronous sample rate converter are not supported in this mode. The Multi-channel mode is only supported on selected platforms, see the table below. The *Example configuration* section contains information on how to enable it.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Note:
  - *LPCXPresso55s69* - MCUXpresso IDE project default debug console is semihost
- Decoder:
  - **AAC:**
    - The reference decoder is supported only in the MCUXpresso IDE and ARMGCC.
  - **FLAC:**
    - *LPCXpresso55s69* - When playing FLAC audio files with too small frame size (block size), the audio output may be distorted because the board is not fast enough.
  - **OPUS:**
    - *LPCXpresso55s69* - The decoder is disabled due to insufficient memory may be distorted because the board is not fast enough.
- Sample rate converter:
  - **SSRC:**

        * *LPCXpresso55s69* - When a memory allocation ERROR occurs, it is necessary to disable the SSRC element due to insufficient memory.

Known issues:

- Decoder:
  - **MP3:**
    * The reference decoder has issues with some of the files. One of the channels can be sometimes distorted or missing parts of the signal.
  - **OPUS:**
    * The decoder doesn't support all the combinations of frame sizes and sample rates. The application might crash when playing an unspupported file.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- SD card with supported audio files
- Personal computer
- Optional:
  - Audio expansion board AUD-EXP-42448 (REV B)

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
  1. Please remove below resistors if on board wifi chip is not DNP:
     - R228, R229, R232, R234
  2. Please make sure R136 is weld for GPIO card detect.

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board.
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.

5. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**********************************
Maestro audio playback demo start
**********************************

[APP_Main_Task] started

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console (*If multi-channel playback mode is enabled, the description is slightly different*):

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"file": Perform audio file decode and playback

 USAGE: file [stop|pause|volume|seek|play|list|info]
   stop              Stops actual playback.
   pause             Pause actual track or resume if already paused.
   volume <volume>   Set volume. The volume can be set from 0 to 100.
   seek <seek_time>  Seek currently paused track. Seek time is absolute time in milliseconds.
   play <filename>   Select audio track to play.
   list              List audio files available on mounted SD card.
   info              Prints playback info.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**
  - Add the MULTICHANNEL_EXAMPLE symbol to preprocessor defines on project level.
  - Connect AUD-EXP-42448 (see the point below).
- **Connect AUD-EXP-42448:**
  - *EVKC-MIMXRT1060:*
    1. Disconnect the power supply for safety reasons.
    2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
    3. Uninstall J99.
    4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file

5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.

Functionality The file play <filename> command calls the STREAMER_file_Create or STREAMER_PCM_Create function from the app_streamer.c file depending on the selected mode.

- When the *Standard* mode is enabled, the command calls the STREAMER_file_Create function that creates a pipeline with the following elements:
  - ELEMENT_FILE_SRC_INDEX
  - ELEMENT_DECODER_INDEX
  - ELEMENT_SRC_INDEX (If SSRC_PROC is defined)
  - ELEMENT_SPEAKER_INDEX
- When the *Multi-channel* mode is enabled, the command calls STREAMER_PCM_Create function, which creates a pipeline with the following elements:
  - ELEMENT_FILE_SRC_INDEX (PCM format only)
  - ELEMENT_SPEAKER_INDEX
  - *Note:*
    - If the input file is an 8 channel PCM file, output to all 8 channels is available. The properties of the PCM file are set in the app_streamer.c file using file source properties sent to the streamer:
      - PROP_FILESRC_SET_SAMPLE_RATE - default value is 96000 [Hz]
      - PROP_FILESRC_SET_NUM_CHANNELS - default value is 8
      - PROP_FILESRC_SET_BIT_WIDTH - default value is 32

Playback itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

EXT_PROCESS_DESC_T ssrc_proc = {SSRC_Proc_Init, SSRC_Proc_Execute, SSRC_Proc_Deinit,␣
↪&get_app_data()->proc_args};

prop.prop = PROP_SRC_PROC_FUNCPTR;
prop.val  = (uintptr_t)&ssrc_proc;

if (streamer_set_property(streamer, 0, prop, true) != 0)
{
    return -1;
}

prop.prop = PROP_AUDIOSINK_SET_VOLUME;
prop.val  = volume;
streamer_set_property(streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 3 different states:

- Idle

- Running
- Paused

In each state, each command can have a different behavior. For more information, see *Commands in detail* section.

Commands in detail The applicatin is controlled by commands from the shell interface and the available commands for the selected mode can be displayed using the help command. Commands are processed in the cmd.c file.

- *help, version*
- *file stop*
- *file pause*
- *file volume <volume>*
- *file seek <seek_time>*
- *file play <filename>*
- *file list*
- *file info*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> D[Write help or version]:::function
    B((Running)):::state --> D
    C((Paused)):::state --> D
    D-->E((No state
    change)):::state
```

file stop

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D
```

```
B((Idle)):::state --> B
C((Running)):::state -->E((Idle)):::state
D((Paused)):::state -->E
```

file pause

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> B
    C((Running)):::state -->E((Paused)):::state
    D((Paused)):::state -->F((Running)):::state
```

file volume <volume>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> M[Error: Play a track first]:::error
    C((Running)):::state --> G{Volume
    parameter
    empty?}:::condition
    D((Paused)):::state --> G
    G -- Yes -->H[Error: Enter volume parameter]:::error
    G -- No -->I{Volume
    in range?}:::condition
    I -- No -->J[Error: invalid value]:::error
    I -- Yes -->K[Set volume]:::function
    J --> L((No state
    change)):::state
    K --> L
    H--> L
```

file seek <seek_time>

The seek argument is only supported in the Standard mode.

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> E[Error: First select
    an audio track to play]:::error
    E-->B
    C((Running)):::state --> F[Error: First
    pause the track]:::error
    F --> C
    D((Paused)):::state --> G{Seek
    parameter
    empty?}:::condition
    G --No --> H{AAC file?}:::condition
```

```
G --Yes --> I[Error: Enter
a seek time value]:::error
I-->N((Paused)):::state;
H --Yes -->J[Error: The AAC decoder
does not support
the seek command]:::error
J-->N
H --No -->K{Seek
parameter
positive?}:::condition
K --No -->L[Error: The seek
time must be
a positive value]:::error
L-->N
K --Yes -->M[Seek the file]:::function
M-->N
```

file play <filename>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    C((Running)):::state --> Z[Error: First stop
    current track]:::error
    D((Paused)):::state --> Z
    B((Idle)):::state --> E{SD Card
    inserted?}:::condition
    E -- No -->F[Error: Insert SD
    card]:::error
    E -- Yes -->G{File
    name
    empty?}:::condition
    G -- Yes -->H[Error: Enter
    file name]:::error
    G -- No -->I{File exists?}:::condition
    I -- No -->O[Error: File
    doesn't exist]:::error
    I -- Yes -->J{Supported
    format?}:::condition
    J -- Yes -->K[Play the track]:::function
    J -- No -->L[Error: Unsupported
    file]:::error
    K -->M((Running)):::state
    L --> W((No state
    change)):::state
    O --> W
    H --> W
    F --> W
    Z --> W
```

file list

```
flowchart TD
    classDef function fill:#69CA00
```

```
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> G{SD Card
inserted?}:::condition
C((Running)):::state --> G
D((Paused)):::state --> G
G -- Yes -->H[List supported files]:::function
G -- No -->I[Error: Insert SD card]:::error
I --> J((No state
change)):::state
H --> J
```

file info

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state -->E[Write file info]:::function
    C((Running)):::state -->E
    D((Paused)):::state -->E
    E --> F((No state
        change)):::state
```

Processing Time Typical streamer pipeline execution times and their individual elements for the EVKC-MIMXRT1060 development board are presented in the following tables. The time spent on output buffers is not included in the traversal measurements. However, file reading time is accounted for. In the case of the WAV codec, the audio file was accessed in every pipeline run. Therefore, during each run, the file was read from the SD card. However, for the MP3 codec, where data must be processed in complete MP3 frames, the file was not read in every run. Instead, it was read periodically only when the codec buffer did not contain a complete frame of data.

For further details, please refer to the *Processing Time* document.

WAV	streamer	file_src	codec	SSRC_proc	speaker
48kHz	1.1 ms	850 µs	150 µs	70 µs	40 µs
44kHz	1.75 ms	850 µs	180 µs	670 µs	40 µs

MP3	streamer	file_src	codec	SSRC_proc	speaker
48 kHz with file read	2.9 ms	2.3 µs	450 µs	60 µs	50 µs
48 kHz without file read	0.5 ms	x	400 µs	40 µs	40 µs
44 kHz with file read	3.2 ms	2.3 µs	440 µs	400 µs	50 µs
44 kHz without file read	0.9 ms	x	440 µs	390 µs	40 µs

Maestro record example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*
- *Processing Time*

Overview The Maestro record example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Loopback** - The application demonstrates a loopback from the microphone to the speaker without any audio processing. Mono, stereo or multichannel mode can be used, depending on the hardware, see *table* below.
- **File recording** - The application takes audio samples from the microphone inputs and stores them to an SD card as an PCM file. The PCM file has following parameters:
  - Mono and stereo : 2 channels, 16kHz, 16bit width
  - Multi-channel (AUD-EXP-42448): 6 channels, 16kHz, 32bit width
- **Voice control** - The application takes audio samples from the microphone input and uses the VIT library to recognize wake words and voice commands. If a wake word or a voice command is recognized, the application write it to the serial terminal.
- **Encoding** - The application takes PCM samples from memory and sends them to the Opus encoder. The encoded data is stored in memory and compared to a reference. The result of the comparison is finally written into the serial terminal.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Note:
  - *LPCXPresso55s69* - MCUXpresso IDE project default debug console is semihost
- Addition labraries
  - **VIT:**
    - The VIT is supported only in the MCUXpresso IDE and ARMGCC.
    - *LPCXpresso55s69* - The VIT is disabled by default due to insufficient memory. To enable it, see the *Example configuration* section.

        * *EVK-MCXN5XX* - Some VIT models can't fit into memory. In order to free some space it is necessary to disable SD card handling and opus encoder. To disable it, see the *Example configuration* section.

- Encoder
  - **OPUS:**
    * *LPCXpresso55s69* - The encoder is not supported due to insufficient memory.
- The File recording mode is not supported on *RW612BGA* development board due to missing SD card slot.

Known issues:

- *EVKB-MIMXRT1170* - After several tens of runs (the number of runs is not deterministic), the development board restarts because a power-up sequence is detected on the RESET pin (due to a voltage drop).

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- Personal computer
- Optional:
  - SD card for file output
  - Audio expansion board AUD-EXP-42448 (REV B)
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
  1. Please remove below resistors if on board wifi chip is not DNP:
     - R228, R229, R232, R234
  2. Please make sure R136 is weld for GPIO card detect.
- *EVK-MCXN5XX:*
  - Short: JP7 2-3, JP8 2-3, JP10 2-3, JP11 2-3
- *RW612BGA:*
  - Connect: JP50; Disconnect JP9, JP11

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate

- 8 data bits
- No parity
- One stop bit
- No flow control

3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.
5. *LPCXpresso55s69:*
   - Insert source of sound to audio Line-In connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
******************************
Maestro audio record demo start
******************************

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"record_mic": Record MIC audio and perform one (or more) of following actions:
- playback on codec
- perform voice recognition (VIT)
- store samples to a file.

USAGE: record_mic [audio|file|<file_name>|vit] 20 [<language>]
The number defines length of recording in seconds.

Please see the project defined symbols for the languages supported.
Then specify one of: en/cn/de/es/fr/it/ja/ko/pt/tr as the language parameter.
For voice recognition say supported WakeWord and in 3s frame supported command.
Please note that this VIT demo is near-field and uses 1 on-board microphone.

NOTES: This command returns to shell after the recording is finished.
       To store samples to a file, the "file" option can be used to create a file
       with a predefined name, or any file name (without whitespaces) can be specified
       instead of the "file" option.

"opus_encode": Initializes the streamer with the Opus memory-to-memory pipeline and
encodes a hardcoded buffer.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. There are several options how to configure the example settings, depending on the environment. For configuration using west and Kconfig, please follow the instructions here. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Connect AUD-EXP-42448:**
  - *EVKC-MIMXRT1060:*
    1. Disconnect the power supply for safety reasons.
    2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
    3. Uninstall J99.
    4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file
    5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.
  - *Note:*
    * The audio stream is as follows:
      - Stereo INPUT 1 (J12) -> LINE 1&2 OUTPUT (J6)
      - Stereo INPUT 2 (J15) -> LINE 3&4 OUTPUT (J7)
      - MIC1 & MIC2 (P1, P2) -> LINE 5&6 OUTPUT (J8)
      - Insert the headphones into the different line outputs to hear the inputs.
      - To use the Stereo INPUT 1, 2, connect an audio source LINE IN jack.
- **Enable VIT:**
  - *LPCXpresso55s69 and MCX-N5XX:*
    * In MCUXPresso IDE (SDK package):
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level.
      2. Add VIT_PROC symbol to preprocessor defines on project level:
         - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
    * In armgcc in SDK package:
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
      2. Remove OPUS_ENCODE=1 and STREAMER_ENABLE_ENCODER preprocessor defines in flags.cmake file.
      3. Add VIT_PROC symbol to preprocessor defines in flags.cmake file.
      4. Remove sdmmc_config.c,.h files from CMakeLists.txt file.
    * In Kconfig:
      1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
      2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd and MCUX_COMPONENT_middleware.sdmmc.host.usdhc
      3. Make sure sdmmc_config files (.c, .h) is excluded from project build

        - remove mcux_add_source function that adds the sources in reconfig.cmake in maestro_record/cm33_core0 folder
    4. Disable fatfs MCUX_COMPONENT_middleware.fatfs and MCUX_COMPONENT_middleware.fatfs.sd
    5. Disable file utils MCUX_COMPONENT_middleware.audio_voice.maestro.file_utils.enable
    6. Make sure Opus encoder is disabled MCUX_COMPONENT_middleware.audio_voice.maestro.element.encoder.opus.enable
    7. Make sure VIT_PROC symbol is defined
        - remove mcux_remove_macro function that removes the VIT_PROC preprocessor definition in reconfig.cmake in maestro_record folder
    8. Make sure VIT processing is enabled MCUX_PRJSEG_middleware.audio_voice.components.vit

- **VIT model generation:**
  - For custom VIT model generation (defining own wake words and voice commands) please use https://vit.nxp.com/
- **Disable SD card handling:**
  - In MCUXPresso IDE:
    - Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level:
      - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
  - In armgcc in SDK package:
    - Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
  - In Kconfig:
    1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
    2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd

Functionality The record_mic or opus_encode command calls the STREAMER_mic_Create or STREAMER_opusmem2mem_Create function from the app_streamer.c file depending on the selected mode.

- When the *Loopback* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements:
  - ELEMENT_MICROPHONE_INDEX
  - ELEMENT_SPEAKER_INDEX
- When the *File recording* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_FILE_SINK_INDEX
- When the *Voice control* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_VIT_INDEX

- When the Encoding mode is selected, the command calls the STREAMER_opusmem2mem_Create function that creates a pipeline with the following elements: - ELEMENT_MEM_SRC_INDEX - ELEMENT_ENCODER_INDEX - ELEMENT_MEM_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val  = DEMO_MIC_CHANNEL_NUM;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_BITS_PER_SAMPLE;
prop.val  = DEMO_AUDIO_BIT_WIDTH;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
prop.val  = DEMO_MIC_FRAME_SIZE;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val  = DEMO_AUDIO_SAMPLE_RATE;
streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *record_mic audio <time>*
- *record_mic file <time>*
- *record_mic <file_name> <time>*
- *record_mic vit <time> <language>*
- *opus_encode*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
    C --> E((No state
    change)):::state
```

record_mic audio <time>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> D{time
    > 0 ?}:::condition
    D -- Yes --> F[recording]:::function
    D -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    F --> C((Running)):::state
    C -->G{time
    expired?}:::condition
    G -- No --> C
    G -- Yes --> B
```

record_mic file <time>/record_mic <file_name> <time>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> C{time
    > 0 ?}:::condition
    C -- Yes --> D{SD card
    inserted?}:::condition
    C -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    D -- Yes --> G{Custom
    file name?}:::condition
    G -- Yes --> H[Create custom
    file name]:::function
    G -- No --> I[Create default
    file name]:::function
    H --> J[Recording]:::function
    I --> J
    J --> K((Running)):::state
```

```
K -->L{time
expired?}:::condition
L -- No --> K
L -- Yes --> B
D -- No --> F[Error: Insert SD
card first]:::error
F --> B
```

record_mic vit <time> <language>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> C{time
    > 0 ?}:::condition
    C -- Yes --> E{Selected
    language?}:::condition
    C -- No --> D[Error: Record length
    must be greater than 0]:::error
    D --> B
    E -- Yes --> G{Supported
    language?}:::condition
    E -- No --> F[Error: Language
    not selected]:::error
    F -->B
    G -- Yes -->I[Recording with
    voice recognition]:::function
    G -- No -->H[Error: Language not supported]:::error
    H --> B
    I --> J((Running)):::state
    J -->K{time
    expired?}:::condition
    K -- No --> J
    K -- Yes --> B
```

opus_encode

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state -->C[Encode file]:::function
    C -->D[Check result]:::function
    D -->B
```

Processing Time Typical execution times of the streamer pipeline for the EVKC-MIMXRT1060 development board are detailed in the following table. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured

pipelines were considered. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone and VIT blocks.

For further details of execution times on individual elements, please refer to the *Processing Time* document.

	streamer
microphone -> speaker 1 channel	40 µs
microphone -> speaker 2 channels	115 µs
microphone -> VIT	7.4 ms

Maestro USB microphone example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*

Overview The Maestro USB microphone example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the microphone inputs and sends them to the USB host via the USB bus. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- *Note:*
  1. When connected to MacBook, change the PCM format from (0x02,0x00,) to (0x01,0x00,) in the g_config_descriptor[CONFIG_DESC_SIZE] in the usb_descriptor.c file. Otherwise, it can't be enumerated and noise is present when recording with the QuickTime player because the sampling frequency and bit resolution do not match.

2. When device functionality is changed, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.
3. If you're having audio problems on Windows 10 for recorder, please disable signal enhancement as the following if it is enabled and have a try again.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. *LPCXpresso55s69:*
   - Insert source of sound to Audio Line-In connector (headphone jack) on the development board.
5. Connect the second micro USB cable between the PC host and the USB port on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**************************************************
Maestro audio USB microphone solutions demo start
**************************************************

Copyright 2022 NXP
```

(continues on next page)

(continued from previous page)

```
[APP_Shell_Task] start

>> usb_mic -1

Starting maestro usb microphone application
The application will run until the board restarts
[STREAMER] Message Task started
Starting recording
[STREAMER] start usb microphone
Set Cur Volume : 1f00
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_mic":  Record MIC audio and playback to the USB port as an audio 2.0
        microphone device.

  USAGE: usb_mic <seconds>
    <seconds>  Time in seconds how long the application should run.
            When you enter a negative number the application will
            run until the board restarts.
  EXAMPLE:   The application will run for 20 seconds: usb_mic 20
```

Details of commands can be found *here*.

Example configuration The example only supports one mode and do not support any additional libraries, so the example can't be configured by user.

Functionality The usb_mic command calls the STREAMER_mic_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_USB_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
```

(continues on next page)

(continued from previous page)

```
    prop.val  = 1;

    streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *usb_mic <seconds>*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
    C --> E((No state
    change)):::state
```

usb_mic <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{seconds
    == 0?}:::condition
    C -- No --> E{seconds
    < 0?}:::condition
    C -- Yes --> D[Error: Incorrect
```

command parameter]:::error
D -->B
E -- Yes --> G[recording]:::function
G --> H((Running)):::state
H --> H
E -- No --> F[recording]:::function
F --> I((Running)):::state
I --> J{seconds
expired?}:::condition
J -- No -->I
J -- Yes --> B

Maestro USB speaker example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*

Overview The Maestro USB speaker example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the USB host and sends them to the audio Line-Out port. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-Channel** - In this mode the device is enumerated as a UAC 5.1. This mode is disabled by default. See the *Example configuration* section to see how to enable the mode.
  - When playing an 5.1 audio file, the example sends only the front-left and front-right channels to the audio Line-Out port (the other channels are ignored), since this example only supports on-board codecs with stereo audio output.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- *Note:*
  - If the USB device audio speaker example uses an ISO IN feedback endpoint, please attach the device to a host like PC which supports feedback function. Otherwise, there might be attachment issue or other problems.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- Headphones with 3.5 mm stereo jack

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Connect the second micro USB cable between the PC host and the USB port on the development board.
5. Insert the headphones into Line-Out connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
*********************************************
Maestro audio USB speaker solutions demo start
*********************************************

Copyright 2022 NXP
[APP_Shell_Task] start

>> usb_speaker -1
```

(continues on next page)

(continued from previous page)

```
Starting maestro usb speaker application
The application will run until the board restarts
[STREAMER] Message Task started
Starting playing
[STREAMER] start usb speaker
Set Cur Volume : fbd5
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_speaker": Play data from the USB port as an audio 2.0
               speaker device.

  USAGE:      usb_speaker <seconds>
   <seconds>  Time in seconds how long the application should run.
              When you enter a negative number the application will
              run until the board restarts.
   EXAMPLE:   The application will run for 20 seconds: usb_speaker 20
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**
  - The feature can be enabled by set the USB_AUDIO_CHANNEL5_1 macro to 1U in the usb_device_descriptor.h file.
  - *Note:* When device functionality is changed, such as UAC 5.1, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.

Functionality The Usb_speaker command calls the STREAMER_speaker_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_USB_SRC_INDEX - ELEMENT_SPEAKER_INDEX

Playback itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_USB_SRC_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_NUM_CHANNELS;
prop.val  = 2;
```

(continues on next page)

(continued from previous page)

```
    streamer_set_property(handle->streamer, 0, prop, true);

    prop.prop = PROP_USB_SRC_SET_FRAME_MS;
    prop.val  = 1;

    streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *usb_speaker <seconds>*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
    C --> E((No state
    change)):::state
```

usb_speaker <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{Duration
    == 0?}:::condition
```

C -- No --> E{Duration
< 0?}:::condition
C -- Yes --> D[Error: Incorrect
command parameter]:::error
D -->B
E -- Yes --> G[playing]:::function
G --> H((Running)):::state
H --> H
E -- No --> F[playing]:::function
F --> I((Running)):::state
I --> J{Duration
expired?}:::condition
J -- No -->I
J -- Yes --> B

Supported features The current version of the audio framework supports several optional features. These can be limited to some MCU cores or development boards variants. More information about support can be found on the specific example page:

- *maestro_playback*
- *maestro_record*
- *maestro_usb_mic*
- *maestro_usb_speaker*

Some features are delivered as prebuilt library and the binaries can be found in the \middleware\audio_voice\components*component*\libs folder. The source code of some features can be found in the \middleware\audio_voice\maestro\src folder.

Decoders Supported decoders and its options are:

Decoder	Sample rates [kHz]	Number of channels	Bit depth
AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
FLAC	8, 11.025, 12, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	16
MP3	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
OPUS	8, 16, 24, 48	1, 2 (mono/stereo)	16
WAV	8, 11.025, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	8, 16, 24

For more details about the reference decoders please see audio-voice-components repository documentation \middleware\audio_voice\components\.

Encoders

- **OPUS encoder** - The current verion of the audio framework only supports a OPUS encoder. For more details about the encoder please see the following link.

Sample rate converters

- **SSRC** - Synchronous sample rate converter. More details about SSRC are available in the User Guide, which is located in middleware\audio_voice\components\ssrc\doc\.
- **ASRC** - Asynchronous sample rate converter is not used in our examples, but it is part of the maestro middleware and can be enabled. To enable ASRC, the maestro_framework_asrc and CMSIS_DSP_Library_Source components must be added to the project. Furthermore, it is necessary to switch from Redlib to Newlib (semihost) library and add a platform definition

to the project (e.g. for RT1170: PLATFORM_RT1170_CORTEXM7). Supported platforms can be found in the PL_platformTypes.h file. More details about ASRC are available in the User Guide, which is located in middleware\audio_voice\components\asrc\doc\.

Additional libraries

- **VIT** - Voice Intelligent Technology (VIT) Wake Word and Voice Command Engines provide free, ready to use voice UI enablement for developers. It enables customer-defined wake words and commands using free online tools. More details about VIT are available in the VIT package, which is located in middleware\audio_voice\components\vit\{platform}\Doc\(depending on the platform) or via following link.

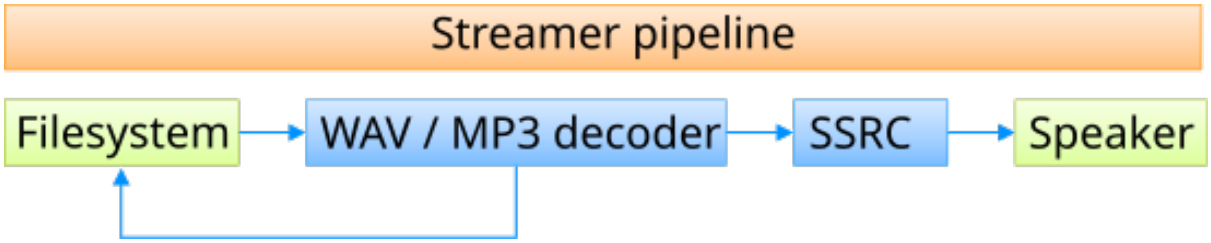
Processing Time

Table of content

- *Maestro playback example*
- *Maestro record example*

The individual time measurements were conducted using a logic analyzer by monitoring changes in the GPIO port levels on the EVKC-MIMXRT1060 development board. These measurements were executed for each individual pipeline run, capturing the timing at each corresponding element, and, when relevant, the interconnections between these elements.

Maestro playback example For the Maestro playback example the following reference audio file was used: test_48khz_16bit_2ch.wav. In this example, the pipeline depicted in the diagram was considered. Media codecs WAV and MP3 were taken into account. To compare the times spent on the SSRC block, sampling rates for both codecs were selected: 44.1 kHz and 48 kHz.



The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines(): streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.

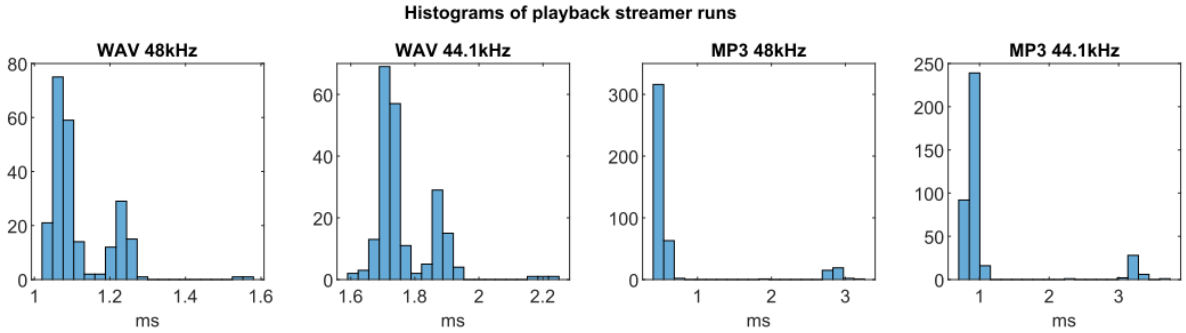
In the scenario involving the WAV codec, the audio file was accessed in every iteration of the streamer pipeline. Meaning, during each run, the file was read directly from the SD card. However, in the case of the MP3 codec, where data processing necessitates complete MP3 frames, the file wasn't read during every run. Rather, it was accessed periodically, triggered when the codec buffer lacked a complete MP3 frame of data. The total time spent on codec processing varies significantly depending on the type and implementation of the codec. For certain types of codecs, like FLAC, there may be multiple file accesses during a single pipeline run. The provided values are specific to the reference implementation. For details about the codecs please see see audio-voice-components documentation middleware\audio_voice\components\.

The duration of the streamer pipeline illustrates that with a sampling frequency of 48 kHz, there is no resampling occurring at the SSRC element. Consequently, the overall pipeline time is lower than in the case of 44.1 kHz audio, where resampling takes place.

To enhance comprehension of the system's behavior, histograms of the pipeline run times and its elements are included. The greater time variance with the MP3 codec is precisely due to

the absence of file reads in every run. In clusters with shorter times, there are no file accesses, while in clusters with longer times, file reads occur. This indicates that the majority of runs do not involve file access.

	WAV 48 kHz	WAV 44 kHz	MP3 48 kHz file read	MP3 48 kHz w/o file read	MP3 44 kHz file read	MP3 44 kHz w/o file read
mean	1.11 ms	1.76 ms	2.87 ms	0.51 ms	3.22 ms	0.89 ms
min	1.03 ms	1.60 ms	2.74 ms	0.41 ms	2.33 ms	0.74 ms
max	1.29 ms	2.23 ms	3.24 ms	1.83 ms	3.73 ms	1.12 ms



Time on each element In the tables and histograms below, the timings for individual elements and their connections are provided. Given that the file reading function was invoked during the codec's operation, the tables for individual elements display the total time on the codec element, the time on the codec element before the file read, and the time on the codec element after the file read. The individual blocks in the tables are as follows:

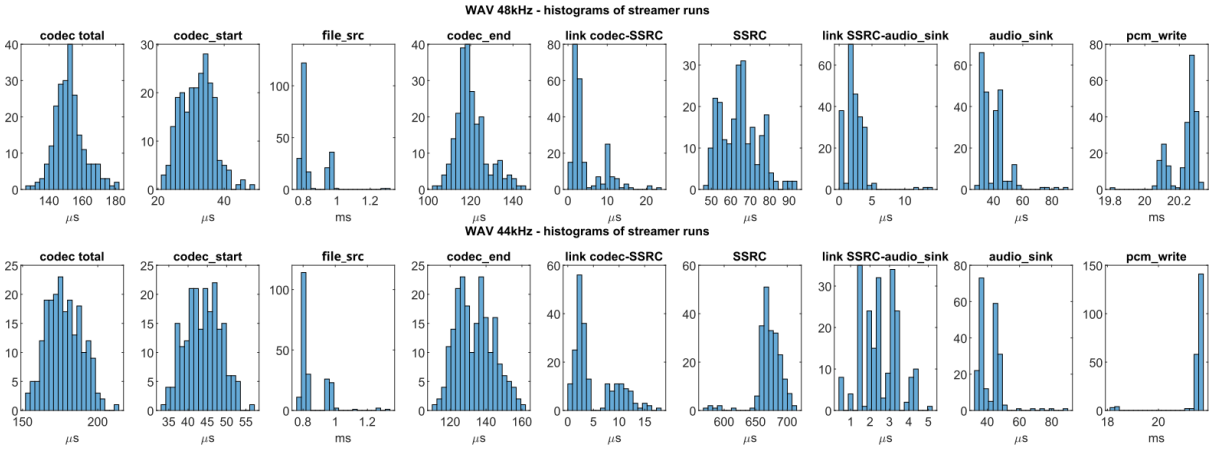
- **streamer** - total time of one pipeline run without time on output buffers
- **codec start** - time on decoder before file read
- **codec end** - time on decoder after file read
- **codec total** - codec_start+codec_end
- **file_src** - file reading time
- **SSRC_proc** - time on SSRC element
- **audio_sink** - time on audio sink without ouput buffers
- **pcm_write** - time on output buffers
- **link** - time on element links

The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **codec** - decoder_sink_pad_process_handler():decoder_pads.c
- **file_src** - filesrc_read():file_src_rtos.c
- **SSRC_proc** - SSRC_Proc_Execute():ssrc_proc.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

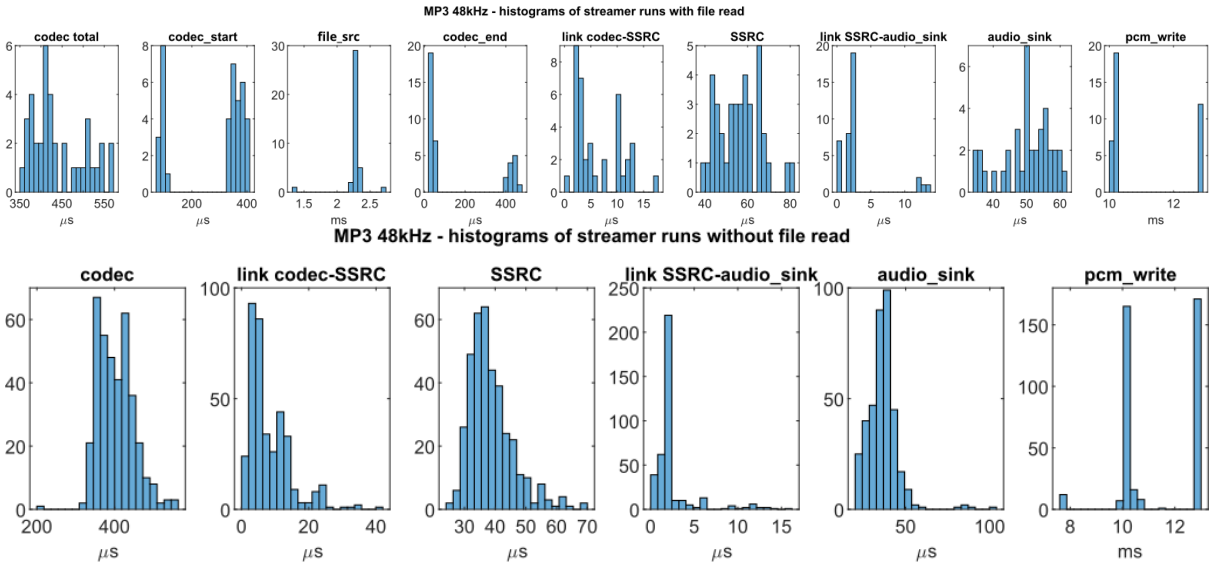
WAV 48kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.119 ms	152 µs	31 µs	0.843 ms	120 µs	5 µs	64 µs	2 µs	40 µs	20.228 ms
min	1.026 ms	125 µs	21 µs	0.773 ms	104 µs	<1 µs	47 µs	<1 µs	30 µs	19.805 ms
max	1.290 ms	193 µs	49 µs	1.311 ms	144 µs	23 µs	93 µs	14 µs	91 µs	20.324 ms

WAV 44kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.765 ms	178 µs	44 µs	0.853 ms	134 µs	5 µs	671 µs	3 µs	42 µs	21.472 ms
min	1.604 ms	145 µs	33 µs	0.770 ms	112 µs	<1 µs	574 µs	<1 µs	33 µs	18.163 ms
max	2.233 ms	218 µs	57 µs	1.335 ms	161 µs	18 µs	715 µs	5 µs	89 µs	21.746 ms



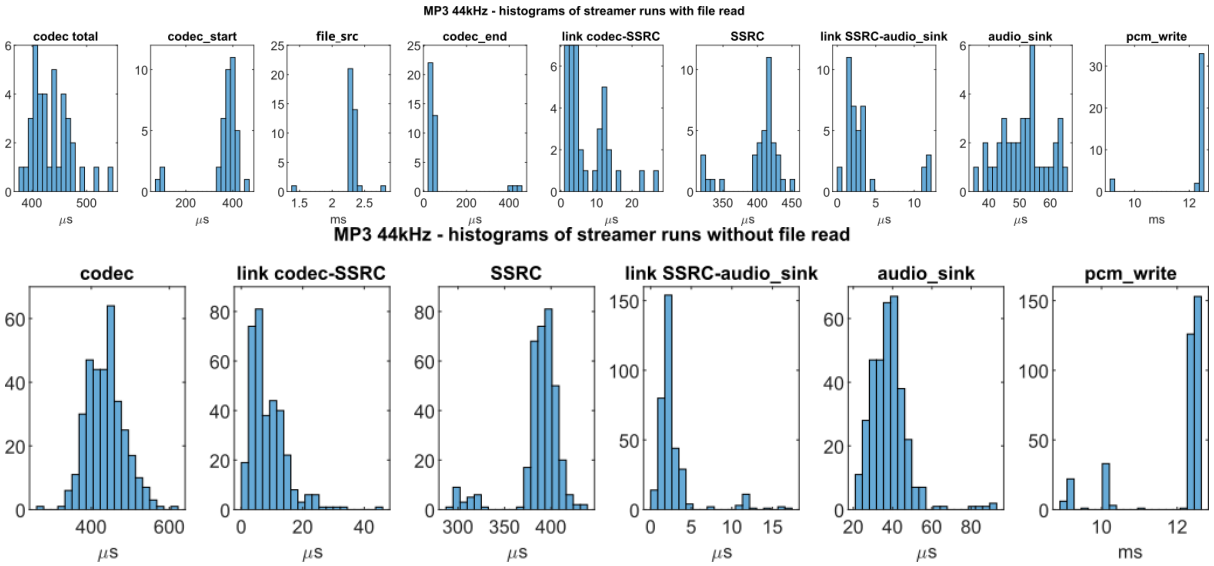
MP3 48 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_|	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	2.871 ms	441 µs	279 µs	2.271 ms	162 µs	6 µs	56 µs	3 µs	50 µs	11.019 ms
min	2.739 ms	353 µs	74 µs	1.353 ms	26 µs	<1 µs	40 µs	<1 µs	34 µs	10.091 ms
max	3.244 ms	570 µs	409 µs	2.728 ms	467 µs	18 µs	80 µs	14 µs	62 µs	12.910 ms

MP3 48 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.508 ms	403 µs	x	x	x	8 µs	39 µs	3 µs	36 µs	11.326 ms
min	0.407 ms	208 µs	x	x	x	<1 µs	25 µs	<1 µs	21 µs	7.715 ms
max	1.834 ms	563 µs	x	x	x	41 µs	69 µs	16 µs	104 µs	12.941 ms

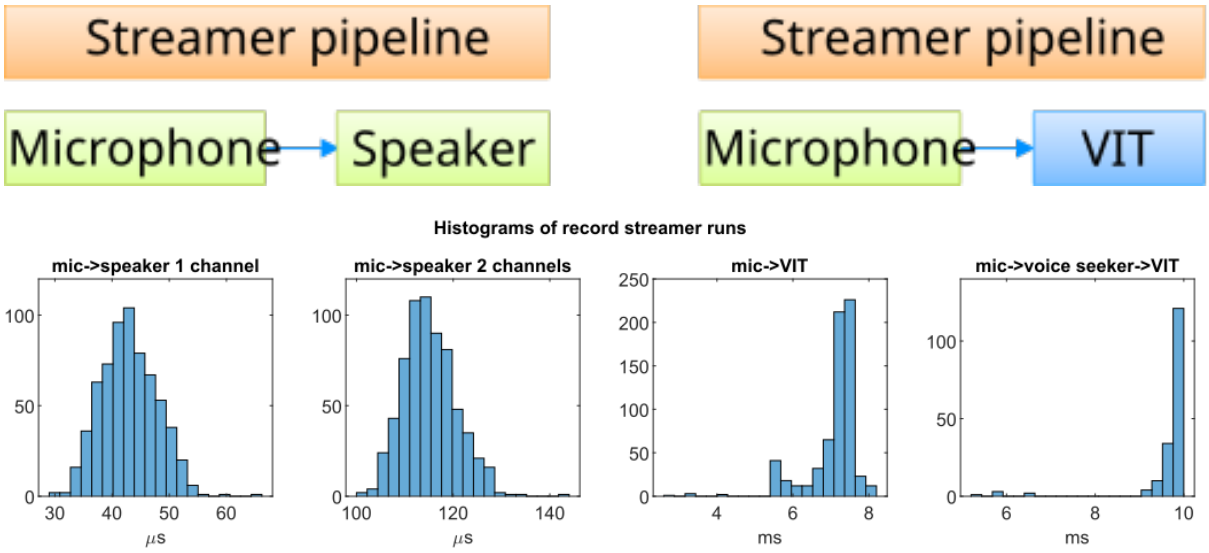


MP3 44 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	3.217 ms	436 µs	367 µs	2.300 ms	66 µs	7 µs	403 µs	3 µs	51 µs	12.188 ms
min	2.329 ms	383 µs	73 µs	1.411 ms	26 µs	2 µs	318 µs	<1 µs	35 µs	9.119 ms
max	3.726 ms	547 µs	464 µs	2.801 ms	441 µs	27 µs	454 µs	12 µs	65 µs	12.529 ms

MP3 44 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.891 ms	437 µs	x	x	x	9 µs	388 µs	3 µs	38 µs	11.934 ms
min	0.738 ms	268 µs	x	x	x	<1 µs	290 µs	<1 µs	22 µs	8.964 ms
max	1.115 ms	620 µs	x	x	x	45 µs	438 µs	17 µs	92 µs	12.624 ms



Maestro record example Typical execution times of the streamer pipeline and its individual elements for the EVKC-MIMXRT1060 development board are detailed in the following tables. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines are depicted in the figure below. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone and VIT blocks. The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines():streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.



The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers and without time reading from the microphone
- **audio_src_start** - time on audio src before reading from the microphone
- **audio_src_end** - time on audio src after reading from the microphone
- **pcm_read** - reading from the microphone
- **vit** - time on VIT element
- **audio_sink** - time on audio sink without ouput buffers

- **pcm_write** - time on output buffers
- **link** - time on element links

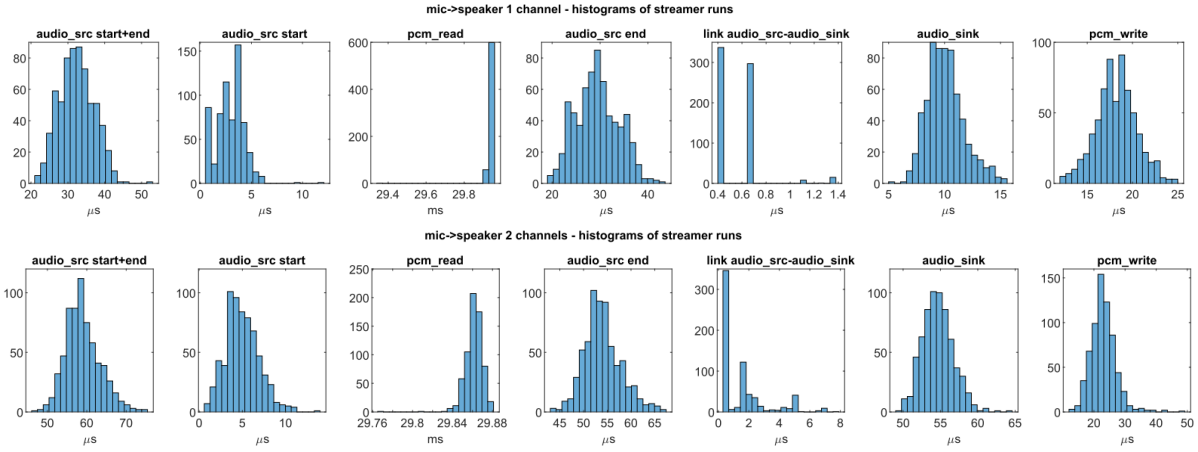
The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **audio_src** - audiosrc_src_process():audio_src.c
- **pcm_read** - streamer_pcm_read():streamer_pcm.c
- **vit** - vitsink_sink_pad_chain_handler():vit_sink.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

Pipeline Microphone -> Speaker

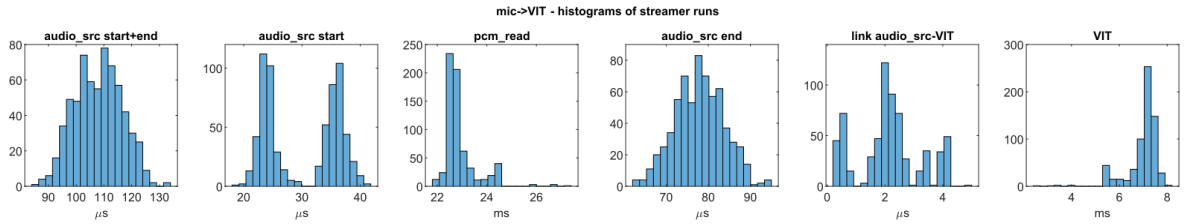
microphone -> speaker mono	stream	audio_src_sta	pcm_re	audio_src_er	link audio_src-audio_sink	audio_sink	pcm_write
mean	43 µs	3 µs	29.938 ms	29 µs	<1 µs	10 µs	18 µs
min	26 µs	<1 µs	29.350 ms	19 µs	<1 µs	5 µs	12 µs
max	72 µs	12 µs	29.957 ms	44 µs	1 µs	15 µs	25 µs

microphone -> speaker stereo	stream	audio_src_sta	pcm_re	audio_src_er	link audio_src-audio_sink	audio_sink	pcm_write
mean	115 µs	5 µs	29.861 ms	54 µs	2 µs	55 µs	23 µs
min	94 µs	<1 µs	29.768 ms	43 µs	<1 µs	50 µs	12 µs
max	154 µs	14 µs	29.880 ms	67 µs	8 µs	65 µs	49 µs



Pipeline Microphone -> VIT

microphone -> VIT	streamer	audio_src_start	pcm_read	audio_src_end	link audio_src-vit	vit
mean	7.380 ms	30 µs	22.624 ms	78 µs	2 µs	7.261 ms
min	2.641 ms	10 µs	2.2265 ms	58 µs	<1 µs	2.559 ms
max	7.780 ms	42 µs	2.7341 ms	94 µs	5 µs	7.624 ms



Maestro on Zephyr

- Based on and tested with Zephyr version, given by tag v4.0.0
- Tested with Zephyr SDK version 16.4
- To see the pre-built documentation, see: README.html. Also see the *documentation section*.

Maestro sample for recording data from microphone to RAM

Description This sample records data from microphone (alias `dmic0` in devicetree) and stores them to a buffer in RAM.

Currently one PDM channel with fixed 16 kHz sample rate and 16 bit sample width is supported.

For configuration options, see Kconfig and prj.conf.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - `Demo result: OK` if everything went OK
  - `Demo result: FAIL` otherwise

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro voice detection sample using VIT

Description Records data from microphone (alias dmic0 in devicetree) and detects voice commands from selected language model. Detected commands are printed via UART.

Language model may be changed via Kconfig using CONFIG_MAESTRO_EXAMPLE_VIT_LANGUAGE selection. For other configuration options, see example's Kconfig and prj.conf.

This project requires an NXP board supported by the VIT library.

The example has to be modified if a new board needs to be added. Please create an issue in that case.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - List of voice commands the model can detect (printed immediately after start)
  - <Specific voice command> if voice command was detected
  - Demo result: FAIL otherwise

Dependencies

- VIT library: https://www.nxp.com/design/design-center/software/embedded-software/voice-intelligent-technology-wake-word-and-voice-command-engines:VOICE-INTELLIGENT-TECHNOLOGY

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro decoder sample

Description Tests and demonstrates decoder functionality in Maestro pipeline.

Supported decoders:

- MP3
- WAV
- AAC
- FLAC
- OPUS with OGG envelop
- (RAW OPUS - TBD)

Data Input:

- Prepared encoded audio data (part of Maestro repository, folder zephyr/audioTracks)
- Prepared decoded audio data (RAW PCM format, part of Maestro repository, folder zephyr/audioTracks)

Function:

1. Loads encoded data into source buffer stored in RAM

2. Decodes audio data using selected decoder and stores data in RAM
3. Compares prepared data with decoded data to check if its the same
4. Prints Demo result: OK or Demo result: FAIL via UART

User Input/Output

- Input:

  None

- Output:

  UART Output

  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Decoder libraries

Configuration

- See prj.conf for user input sections
  - Selecting decoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_DECODER_SELECTED in prj.conf file. When no decoder is selected, default one (WAV) is used instead.
  - System settings should be modified (stack size, heap size) based on selected decoder and system capabilities/requirements in prj.conf.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms Currently tested for:

- RD_RW612_BGA - Working decoders: FLAC, WAV, OPUS OGG

Maestro encoder sample

Description Tests and demonstrates encoder functionality in Maestro pipeline.

Supported encoders:

- OPUS with OGG envelop - TBD
- RAW OPUS - TBD

Input:

- Prepared decoded audio data (RAW PCM format, part of Maestro repository)
- Prepared encoded audio data (part of Maestro repository)

Function:

1. Loads RAW data into source buffer stored in RAM
2. Encodes audio data using selected encoder and stores data in RAM
3. Compares prepared data with decoded data if same
4. Prints Demo result: OK or Demo result: FAIL via UART

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Encoder libraries

User Input/Output

Input:

- None

Output:

- UART Output
  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Configuration

- See prj.conf for user input sections
  - Selecting encoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_ENCODER_SELECTED in prj.conf file. When no encoder is selected, default one (OPUS) is used instead.
  - System settings should be modified (stack size, heap size) based on selected encoder and system capabilities/requirements in prj.conf file.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms

Currently tested for:

- RD_RW612_BGA - Working encoders: None.

Maestro mem2mem sample

Description

Tests basic memory to memory pipeline.

Function:

1. Moves generated data with fixed size of 256B from memory source to memory sink.
2. Compares copied data to check if they're the same.
3. Returns Demo result: OK or Demo result: FAIL via UART.

- *Maestro environment setup*
- *Build and run Maestro example*
  - *Using command line*
  - *Using MCUXpresso for VS Code*
- *Folder structure*
- *Supported elements and libraries*
- *Examples support*
- *Creating your own example*
- *Documentation*
- *FAQ*

Maestro environment setup Follow these steps to set up a Maestro development environment on your machine.

1. If you haven't already, please follow this guide to set up a Zephyr development environment and its dependencies first:
   - Cmake
   - Python
   - Devicetree compiler
   - West
   - Zephyr SDK bundle
2. Get Maestro. You can pick either of the options listed below. If you need help deciding which option is the best fit for your needs, please see the *FAQ*.
   - Freestanding Maestro - This option pulls in only Maestro's necessary dependencies.

     Run:

     ```
     1. west init -m <maestro repository url> --mr <revision> --mf west-freestanding.yml
     ↪<foldername>
     2. cd <foldername>
     3. west update
     ```

   - Maestro as a Zephyr module

     To include Maestro into Zephyr, update Zephyr's west.yml file:

     ```
     projects:
     name: maestro
     url: <maestro repository url>
     revision: <revision with Zephyr support>
     path: modules/audio/maestro
     import: west.yml
     ```

     Then run west update maestro command.

Build and run Maestro example These steps will guide you through building and running Maestro samples. You can use either the command line utilizing Zephyr's powerful west tool or you can use VS Code's GUI. Detailed steps for both options are listed below.

Using command line See Zephyr's Building, Flashing and Debugging guide if you aren't familiar with it yet.

1. To **build** a project, run:

   ```
   west build -b <board> -d <output build directory> <path to example> -p
   ```

   For example, this compiles VIT example for rd_rw612_bga board:

   ```
   1. cd maestro/zephyr
   2. west build -b rd_rw612_bga -d build samples/vit -p
   ```

2. To **run** a project, run:

   ```
   west flash -d <directory>
   ```

   e.g.:

```
west flash -d build
```

3. To **debug** a project, run:

```
west debug -d <directory>
```

e.g.:

```
west debug -d build
```

Using MCUXpresso for VS Code For this you have to have NXP's MCUXpresso for VS Code extension installed.

1. Import your topdir as a repository to MCUXPresso for VS Code:
   - Open the MCUXpresso Extension. In the *Quickstart Panel* click *Import Repository*.
   - In the displayed menu click *LOCAL* tab and select the folder location of your *topdir*.
   - Click *Import*.
   - The repository is successfully added to the *Installed Repositories* view once the import is successful.
2. To import any project from the imported repository:
   - In the *Quickstart Panel* click *Import Example from Repository*.
   - For **Repository** select *your imported* repository.
   - For **Zephyr SDK** the installed Zephyr SDK is selected automatically. If not, select one.
   - For **Board** select your board (*make sure* you've selected the *correct* revision).
   - For **Template** select the folder path to your project.
   - Click the *Create* button.
3. Build the project by clicking the *Build Selected* icon (displayed on hover) in the extension's *Projects* view. After the build, the debug console window displays the memory usage (or compiler errors if any).
4. Debug the project by clicking the *Debug* (play) icon (displayed on hover) in the extension's *Projects* view.
5. The execution will pause. To continue execution click *Continue* on the debug options.
6. In the *SERIAL MONITOR* tab of your console panel, the application prints the Zephyr boot banner during startup and then prints the test results.

Folder structure

```
maestro/
    ...
    zephyr/              All Zephyr related files
      samples/           Sample examples
      tests/             Tests
      audioTracks/       Audio tracks for testing
      doc/               Documentation configuration for Sphinx
      wrappers/          NXP SDK Wrappers
      scripts/           Helper scripts, mostly for testing
      module.yml         Defines module name, Cmake and Kconfig locations
      CMakeList.txt      Defines module's build process
      Kconfig            Defines module's configuration
      osa/               Deprecated. OSA port for Zephyr
      ...
```

Supported elements and libraries Here is the list of all features currently supported in Maestro on Zephyr. Our goal is to support all features in Maestro on Zephyr that are already supported in Maestro on NXP's SDK and to extend them further.

Supported elements:

- Memory source
- Memory sink
- Audio source
- Audio sink
- Process sink
- Decoder
- Encoder

Supported decoders:

- WAV
- MP3
- FLAC
- OPUS OGG
- AAC

Supported encoders:

- OPUS RAW

Supported libraries:

- VIT

Examples support All included examples use UART as output. Examples are located in `zephyr/tests` and `zephyr/samples` directories.

List of included examples:

- *Maestro sample for recording data from microphone to RAM*
- *Maestro voice detection sample using VIT*
- *Maestro encoder sample*
- *Maestro decoder sample*
- *Maestro mem2mem sample*

Examples support for specific boards:

Example	RDRW612BGA	LPCx-presso55s69	MIMXRT1060EVKE	MIMXRT1170EVKB
Record	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
VIT	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
Encoder	In progress: OPUS RAW	TO BE TESTED	TO BE TESTED	TO BE TESTED
Decoder	YES - WAV, FLAC, OPUS OGG	TO BE TESTED	TO BE TESTED	TO BE TESTED
Mem2mem	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED

Creating your own example There are two ways to create your own example - you can either one of the included examples as a reference or you can create your own example from scratch by hand.

When creating your own example from scratch, set CONFIG_MAESTRO_AUDIO_FRAMEWORK=y in your prj.conf file. Then you can start enabling specific elements by setting CONFIG_MAESTRO_ELEMENT_<NAME>_ENABLE=y.

However, the recommended way to edit config options is to open gui-config (or menuconfig) by calling west build -t guiconfig. Then you can use the graphical interface to interactively turn on/off the features you need.

Documentation Please note, Maestro documentation is under reconstruction. It is currently mixing several tools and formats.

To see the pre-generated Maestro Zephyr documentation, see zephyr/doc/doc/README.html

To generate the Zephyr documentation, go under zephyr/doc folder and execute make html. Sphinx version sphinx-build 8.1.3 must be installed. Open doc/doc/html/README.hml afterwards.

To see Maestro core documentation, go to the Maestro top directory and see README.md.

FAQ

1. Should I choose the freestanding version of Maestro or should integrate it into my west instead?
   - Freestanding version of Maestro pulls in all the dependencies it needs including Zephyr itself.
   - Integrating it as a module is easier if you already have your Zephyr environment set up.

Maestro Audio Framework changelog

2.0.2

- Removed VoiceSeeker support

2.0.1

- Fixed filesrc buffer alignment

2.0.0 (newest)

- Added Zephyr port, see *Zephyr README*.
  - Possible to use standalone version, pulling its own Zephyr and dependencies
  - Possible to import it as a module in your Zephyr project
- Changed build system - newly uses Kconfig and Cmake
- Supports NXP MCUXSDK (previously 2.x)
- Changed folder structure and names to improve readability (description may be found in *README*)
- Removed audio libraries and placed into audio-voice-components repository
- Added libraries are pulled into the build via Kconfig and Cmake

- Changed Maestro library core - minor changes

1.8.0

- New platforms support: MCX-N5XX-EVK, FRDMMCXN236 and RD-RW612-BGA
- Fixed compilation warnings
- Documentation improvements and updates
  - Added section with processing time information
  - Added application state diagrams
- Various updates and fixes

1.7.0

- Removed EAP support for future SDK releases
- Created new API for audio_sink and audio_src to support USB source, sink
- ASRC library integrated
- License changed to BSD 3-Clause
- Improved pipeline creation API
- Fixed compilation warnings in Opus
- Various other improvements and bug fixes

1.6.0

- Up to 2 parallel pipelines supported
- Synchronous Sample Rate Converter support Added
- Various improvements and bug fixes

1.5.0

- Enabled switching from 2 to 4 channel output during processing
- PadReturn type has been replaced by FlowReturn
- Support of AAC, WAV, FLAC decoders
- Renamed eap element to audio_proc element
- Added audio_proc to VIT pipeline to support VoiceSeeker
- Minor bug fixes

1.4.0

- Use Opusfile lib for Ogg Opus decoder
- Refactor code, fix issues found in unit tests
- Various bug fixes

1.3.0

- Make Maestro framework open source (except mp3 and wav decoder)
- Refactor code, remove unused parts, add comments

1.2.0

- Unified buffering in audio source, audio sink
- Various improvements and bug fixes

1.0_rev0

- Initial version of framework with support for Cortex-M7 platforms

3.5 Wireless

3.5.1 NXP Wireless Framework and Stacks

3.5.2 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.

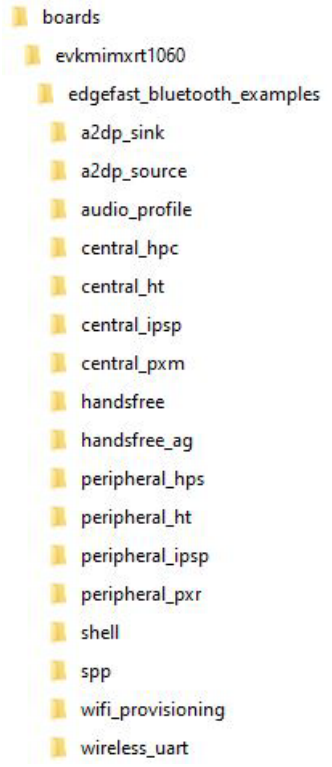
The APIs of the EdgeFast Bluetooth Protocol Abstraction Layer host stack are described in the EdgeFast Bluetooth Protocol Abstraction Layer RM document.

Note: The online document of the Zephyr Bluetooth Host stack is available here: https://docs.zephyrproject.org/latest/reference/bluetooth/index.html.

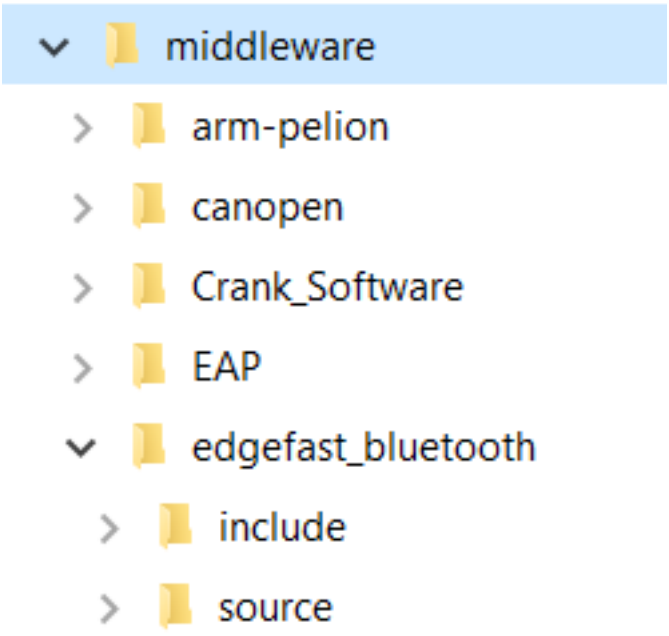
Parent topic:*Introduction*

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.

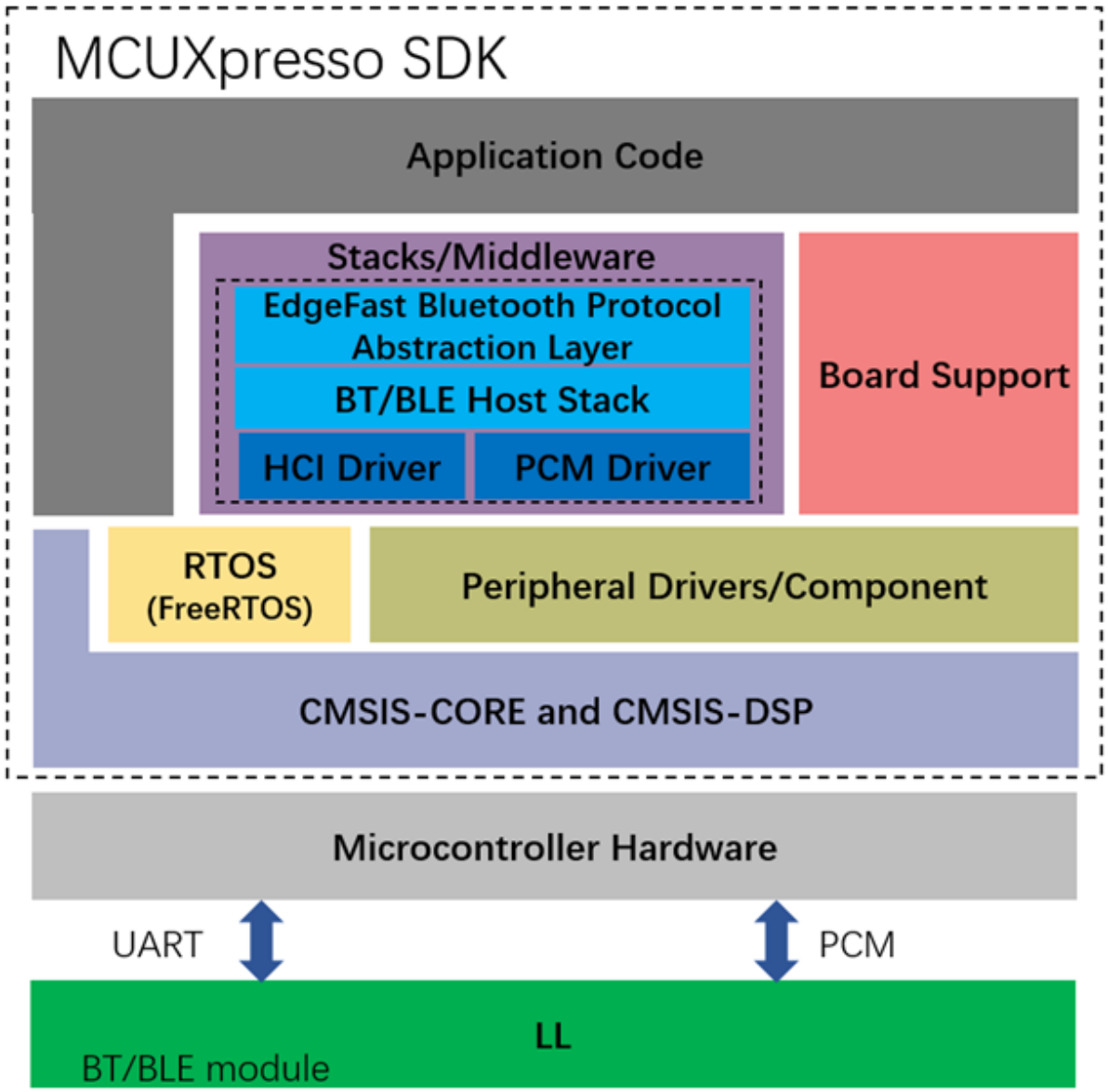
Parent topic:*Overview*

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.

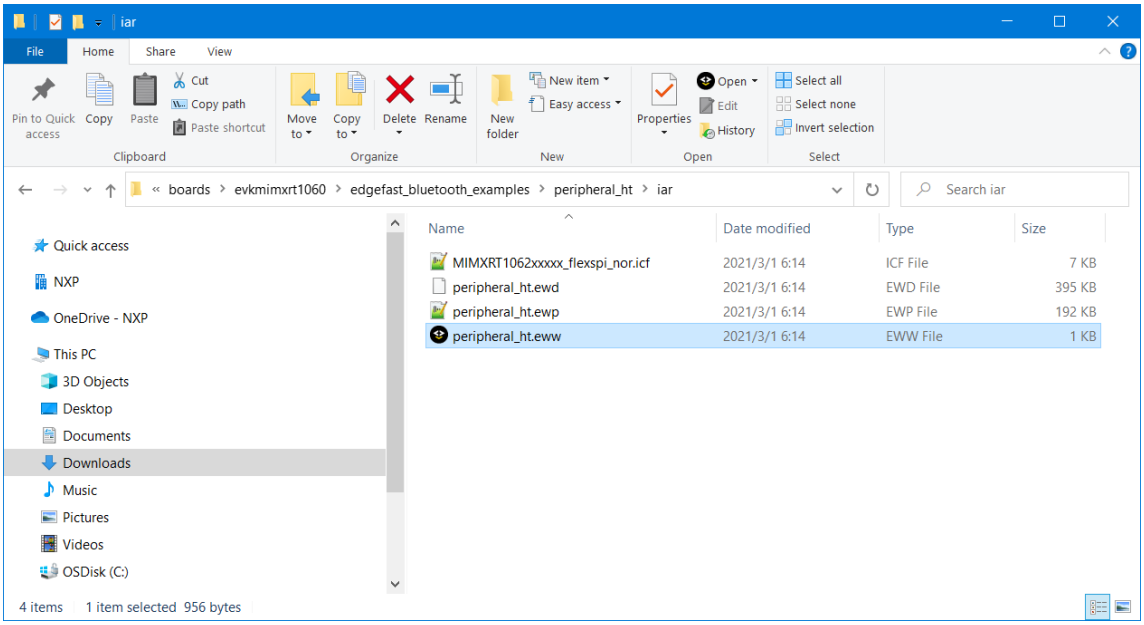
Run a demo application using IAR This document uses EVKRT1060 EdgeFast Bluetooth Protocol Abstraction Layer API example to describe the steps to open a project, build an example, and run a project. For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide*(document MCUXSDKGSUG) 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

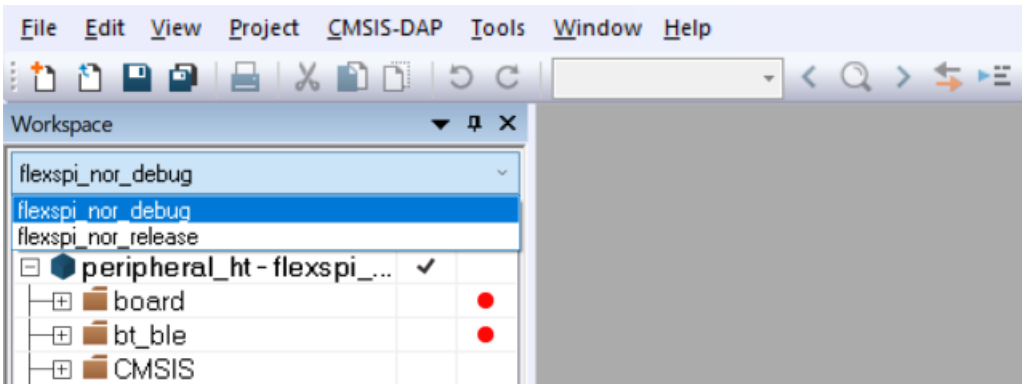
2. Open the IAR workspace file. For example, the highlighted *.eww format file



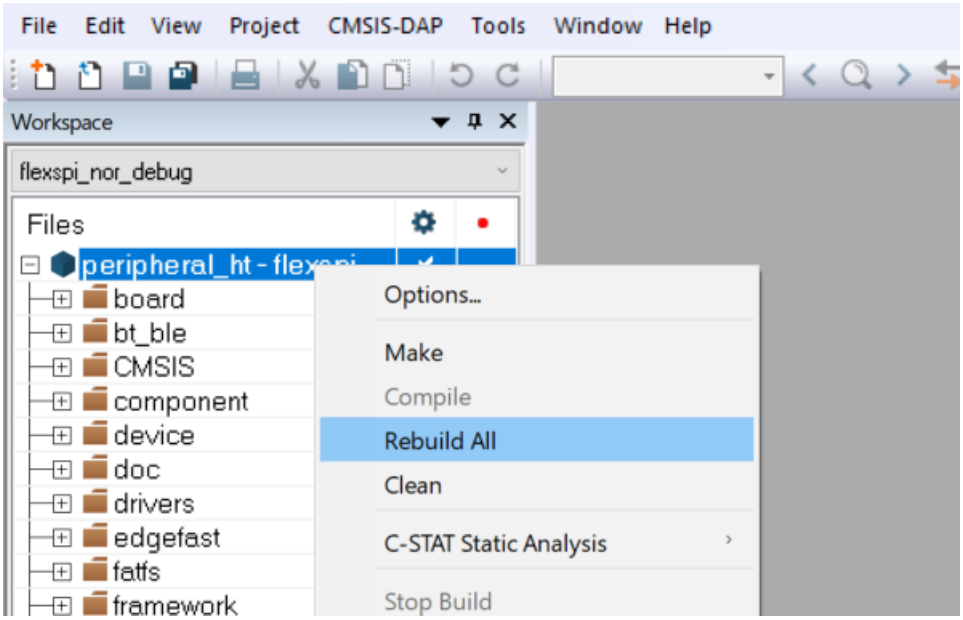
Parent topic: Run a demo application using IAR

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.

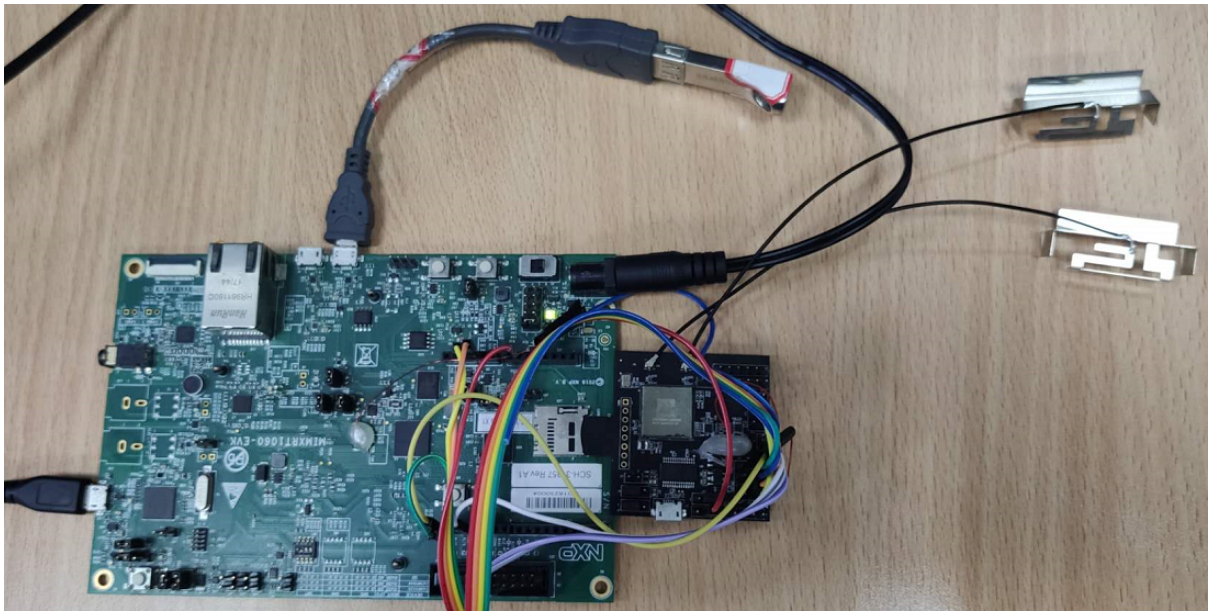


Note: Wireless module does not have flash hardware and requires 512 KB image loaded from board (such as RT1060) on system startup. The 512 KB image is kept on RT1060 side and only flexspi_nor target is supported for Bluetooth examples. Other targets are not supported because memory size limit.

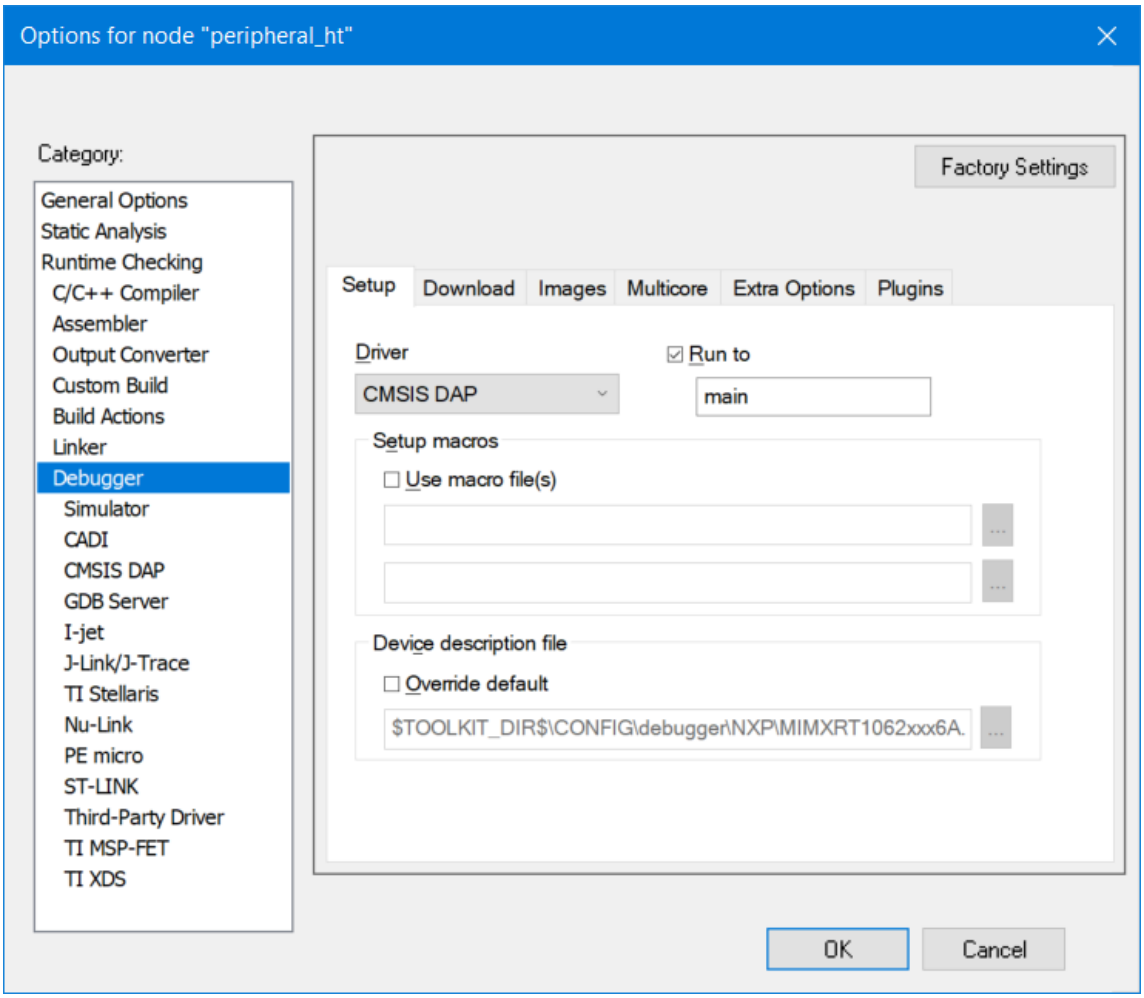
Parent topic: Run a demo application using IAR

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.

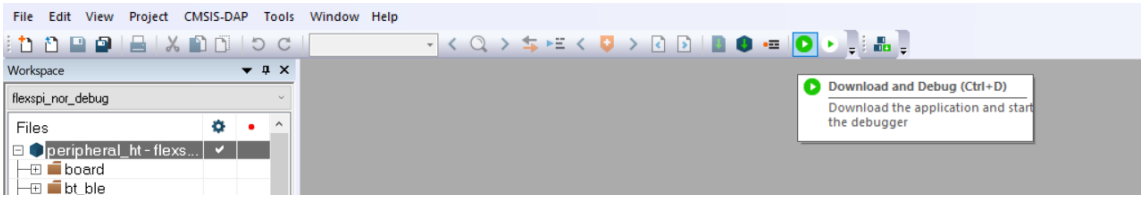
The following figure shows the connection of RT1060 and the uSD wireless module.



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*

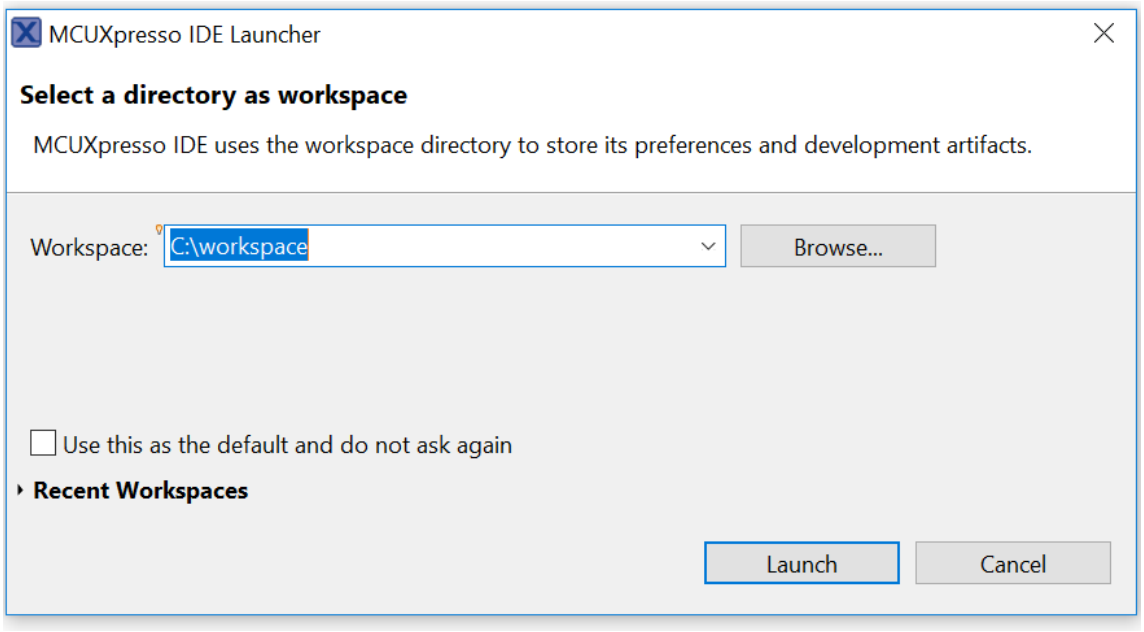
Run a demo application using MCUXpresso IDE

This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MCUXpresso IDE.

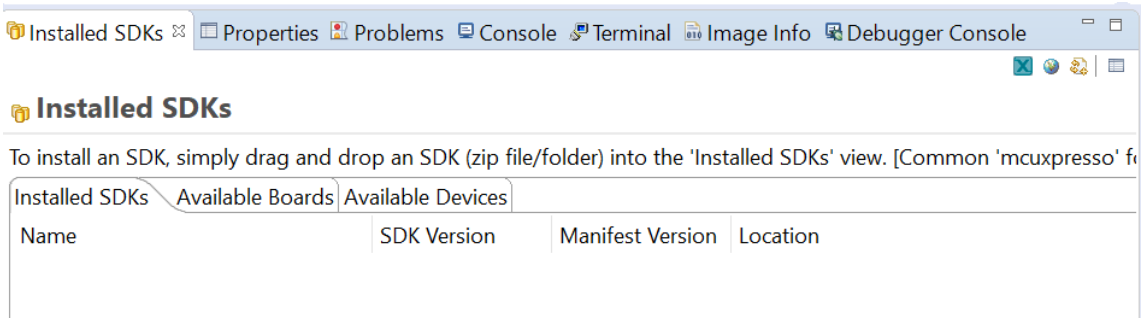
For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf* and refer to the readme file in the corresponding demo's directory.

Open an MCUXpresso IDE example

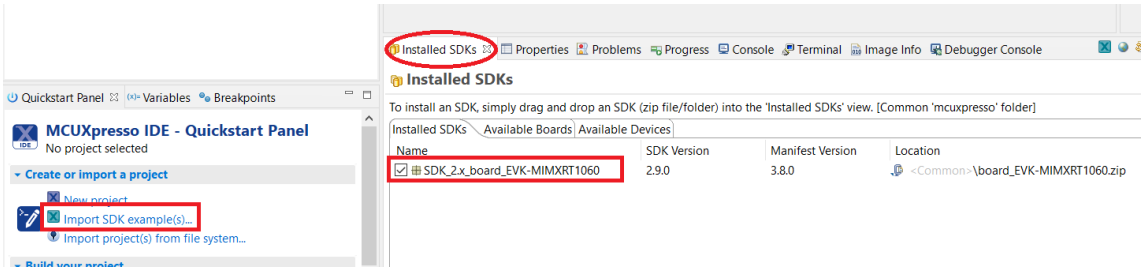
1. Open MCUXpresso IDE and open an existing or a new workspace location.



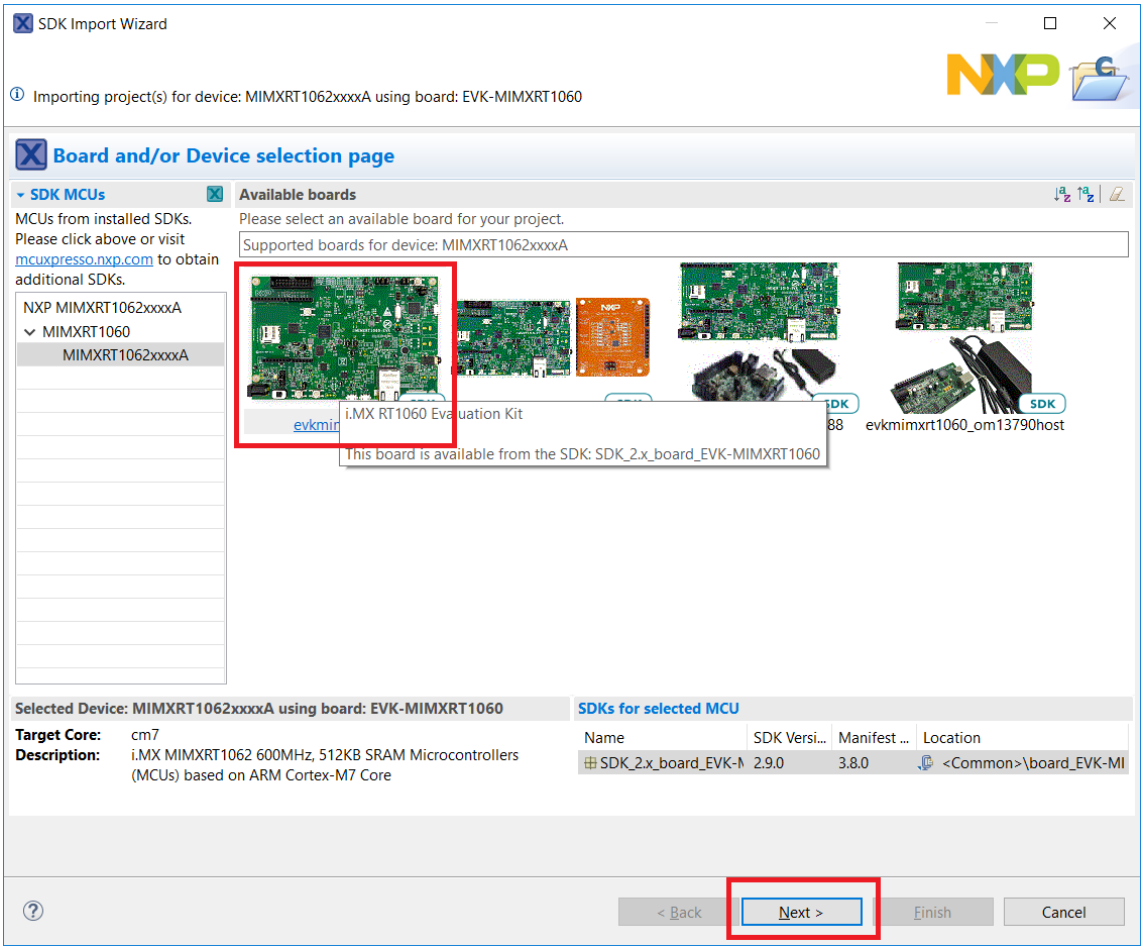
2. Drag and drop the package archive into the MCUXpresso Installed SDKs area in the lower right of the main window.



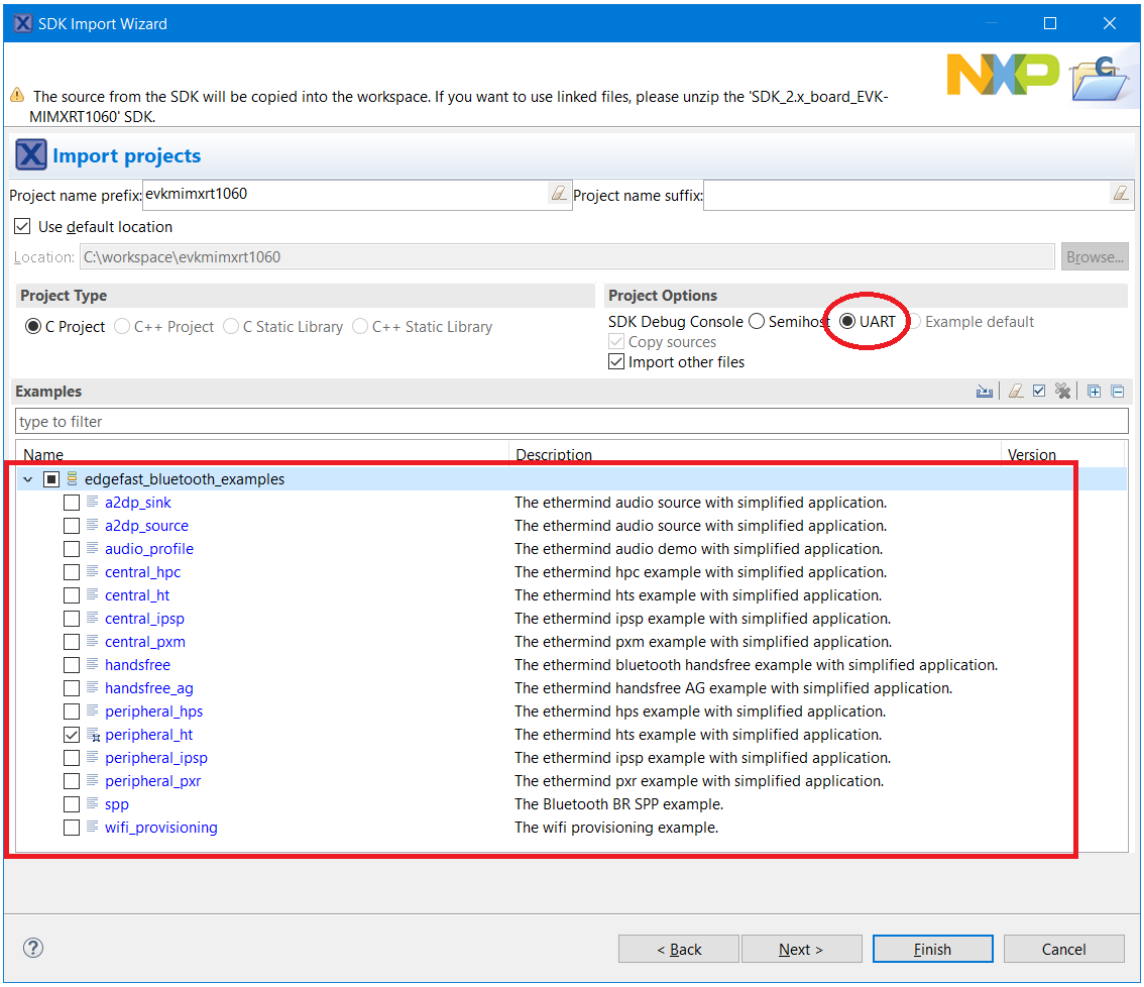
3. After the SDK is loaded successfully, select the **Import the SDK examples(s)...** to add examples to your workspace.



4. Select the evkmimxrt1060 board and click the **Next** button to select the desired example(s).



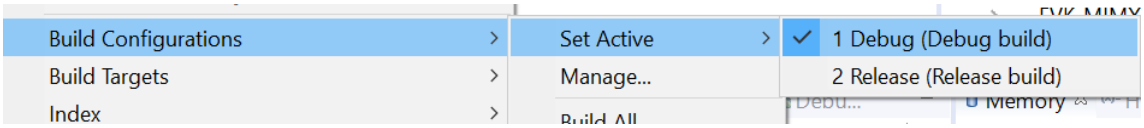
5. Select the evkmimxrt1060 board EdgeFast Bluetooth example. For example, peripheral_ht.
6. Ensure to change SDK debug console from **Semihost** to **UART**.
7. Click **Finish**.



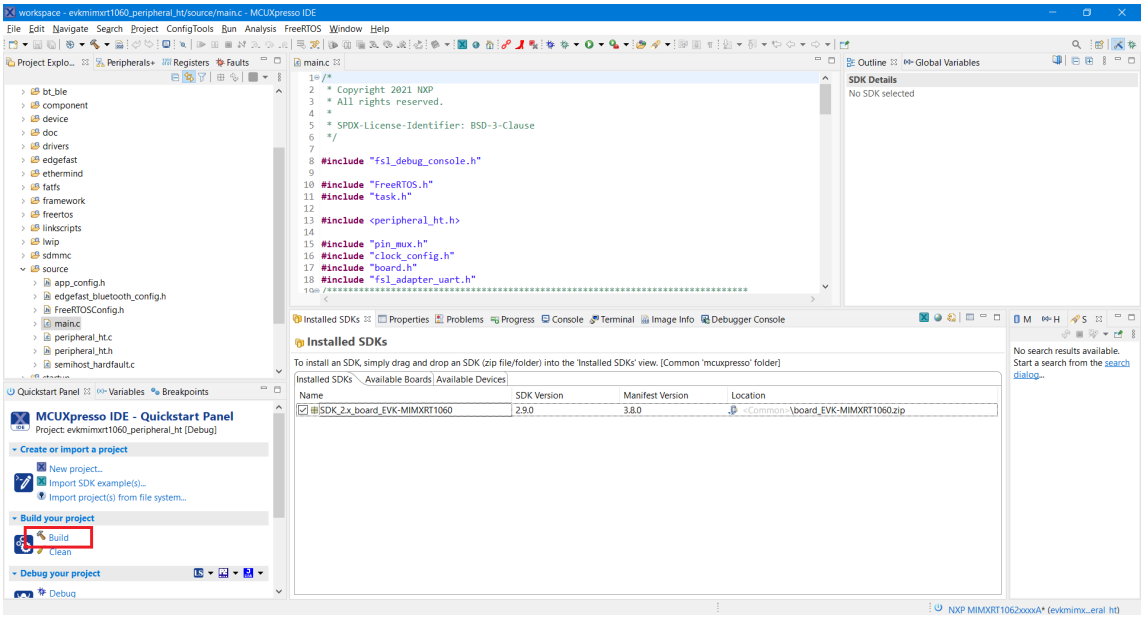
Parent topic: Run a demo application using MCUXpresso IDE

Build an MCUXpresso IDE example

1. Select desired target for your project.



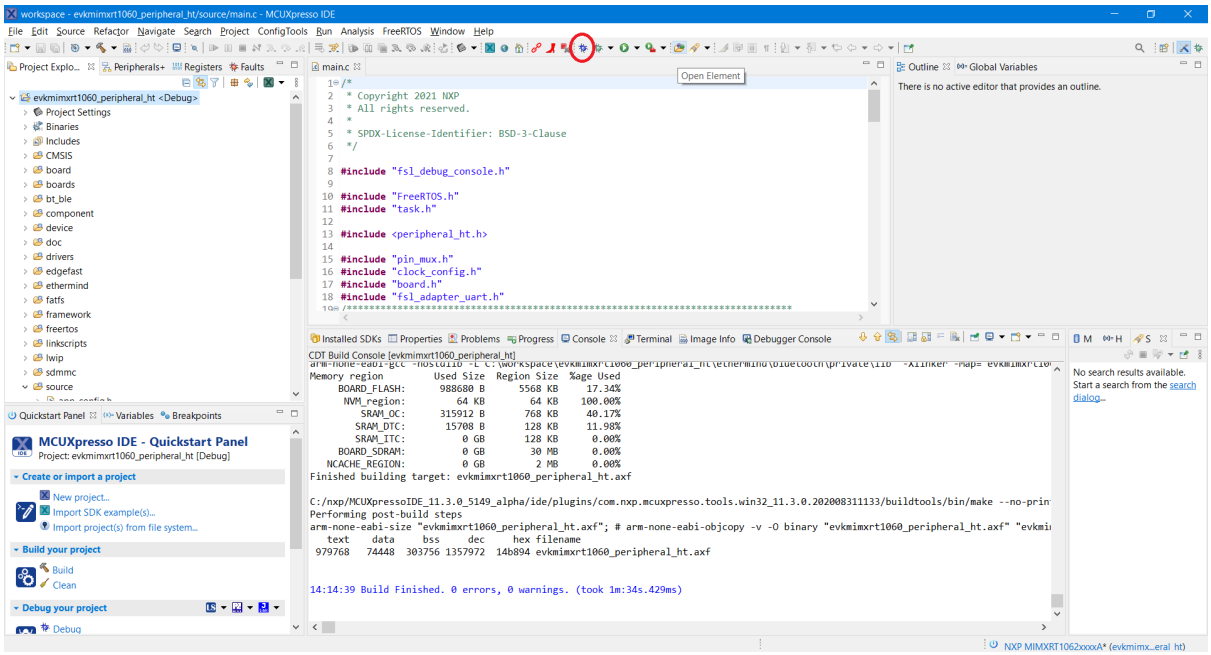
2. Build MCUXpresso IDE EdgeFast Bluetooth Protocol Abstraction Layer project.



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

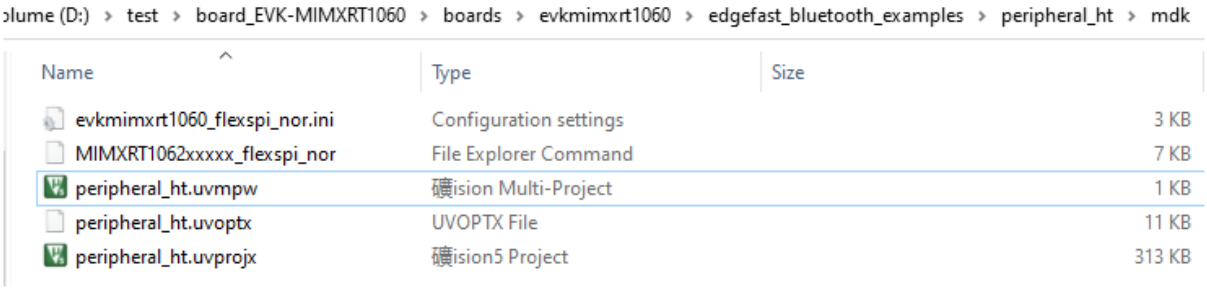
Parent topic:*Demo*

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.

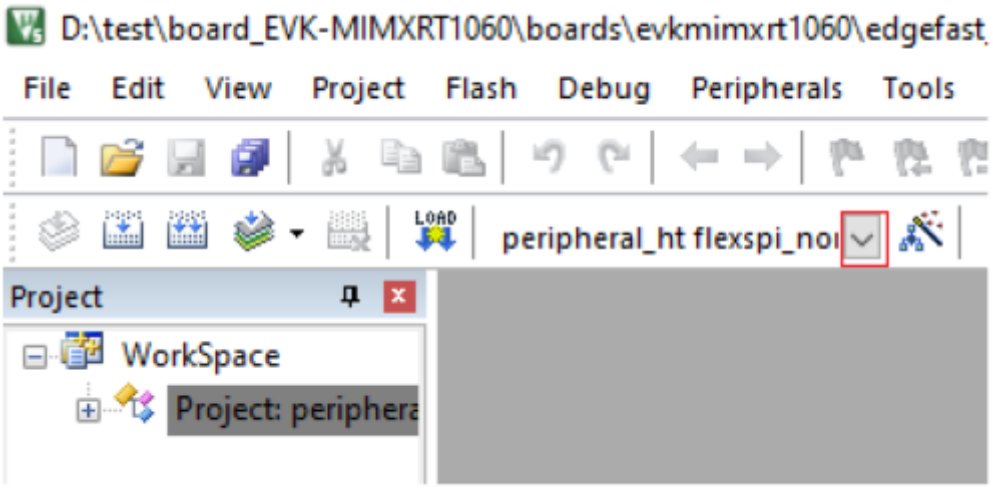
1. The example projects are available at: <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/mdk.
2. Open the mdk workspace file. For example, the highlighted *.uvmpw format file.



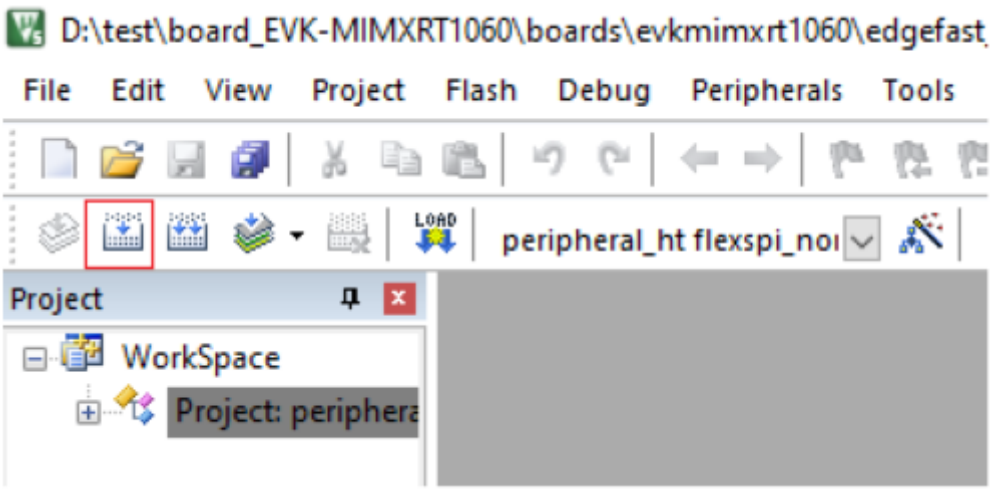
Parent topic: Run a demo application using MDK

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.



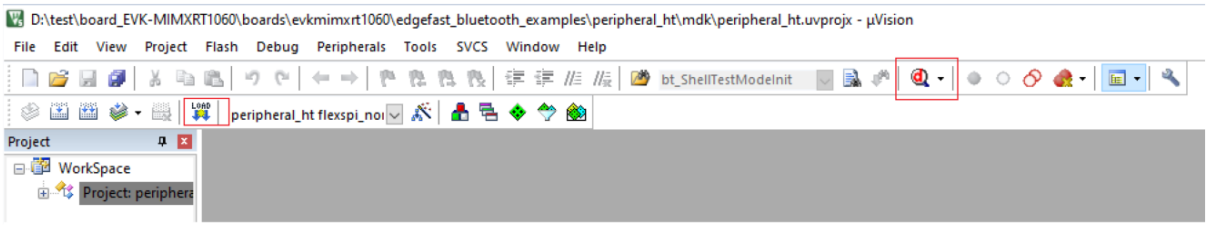
2. Click the highlighted icon to build the EdgeFast Bluetooth Protocol Abstraction Layer project.



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*.

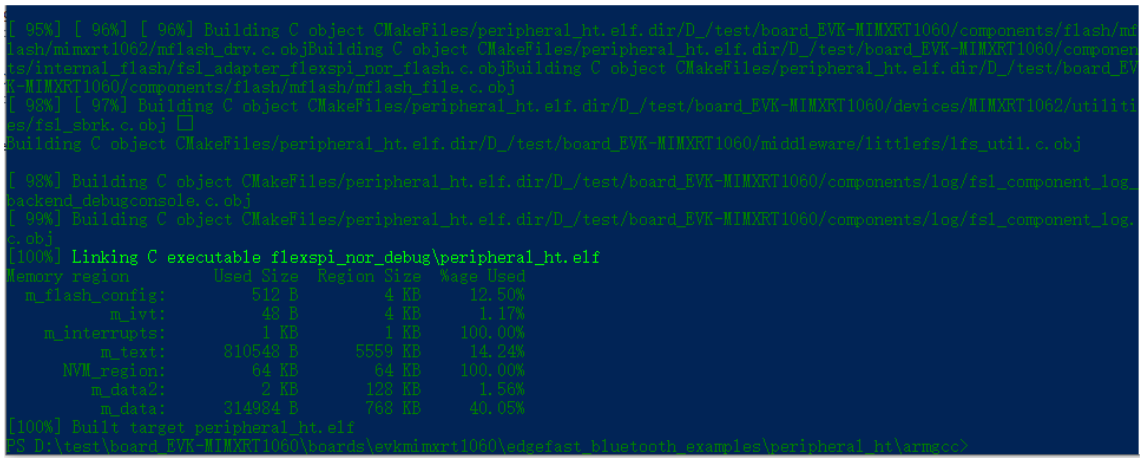
Parent topic: Run a demo application using Arm GCC

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

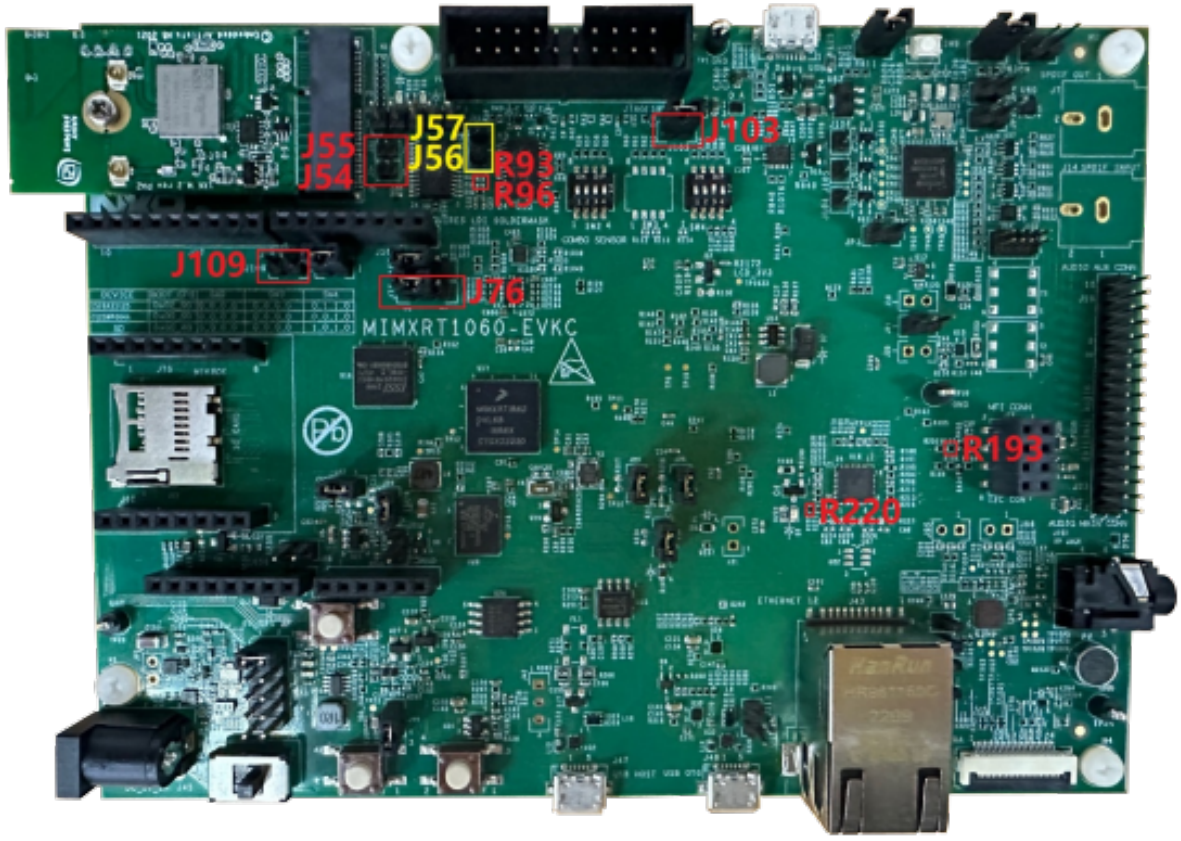
- **HCI UART rework**

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

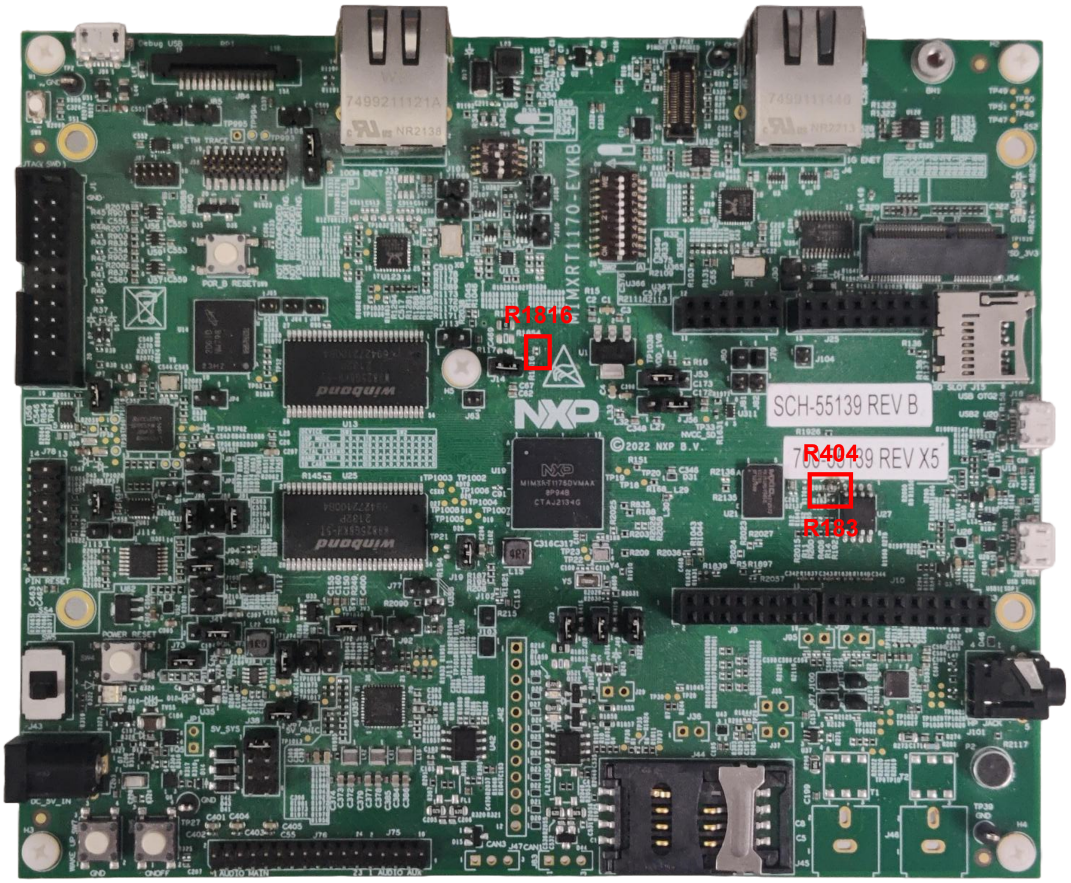
This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

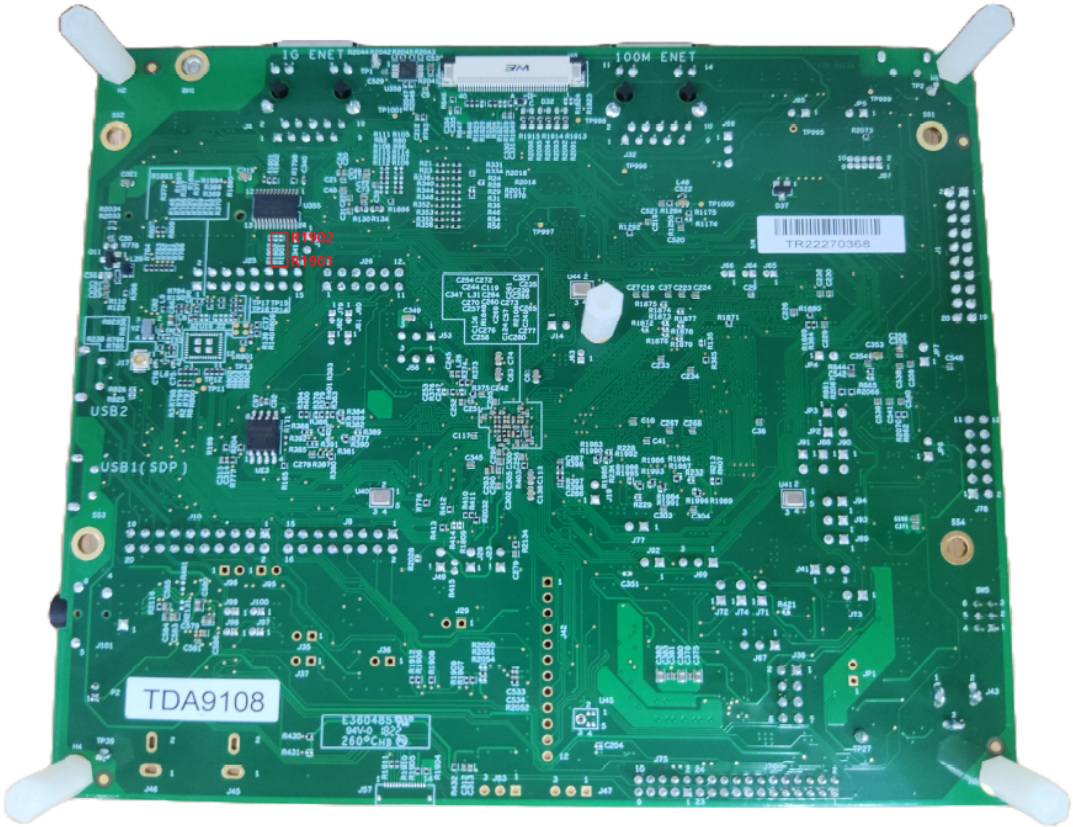
The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

Hardware rework

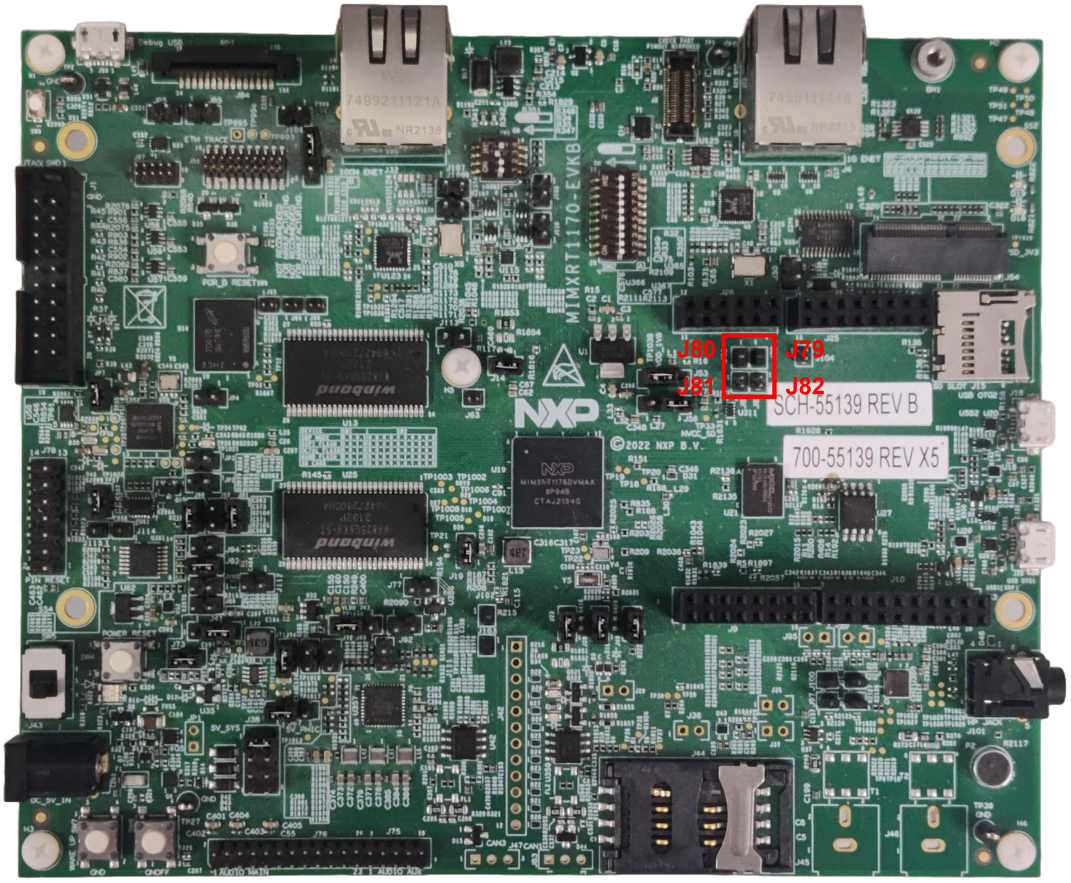
- **HCI UART rework**
  1. Remove resistors R183 and R1816.
  2. Solder 0 ohm resistor to R404, R1901, and R1902.

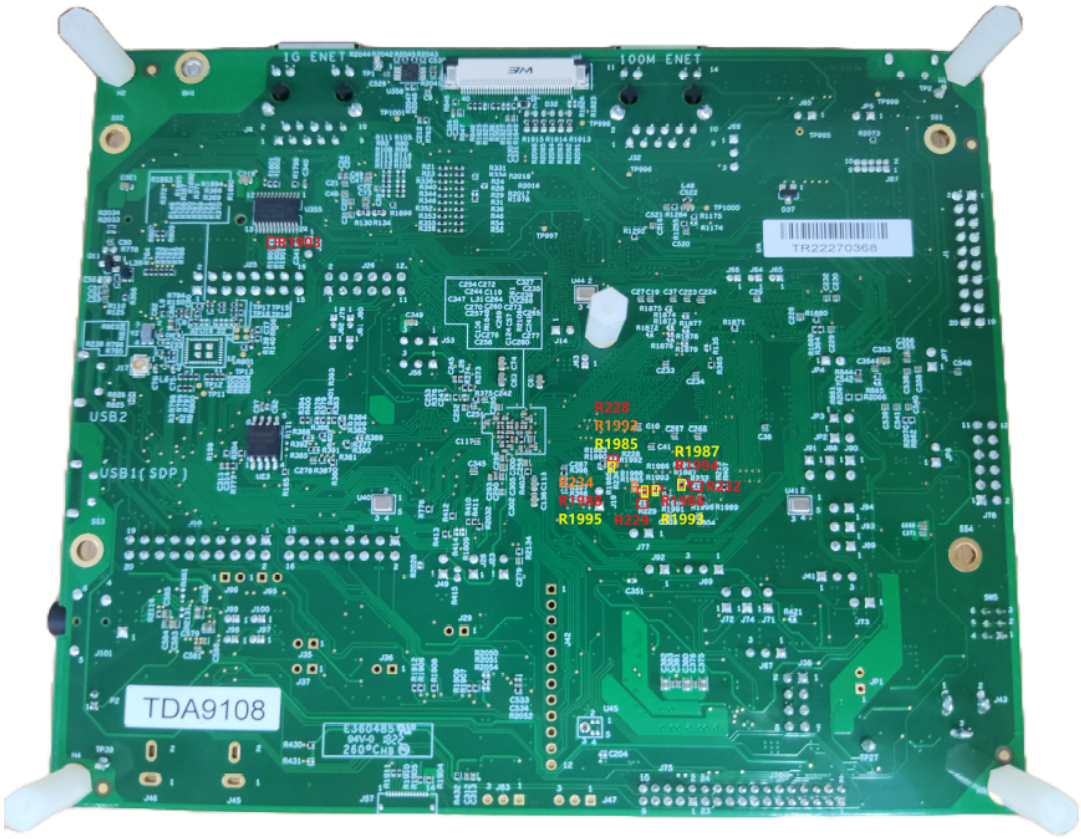




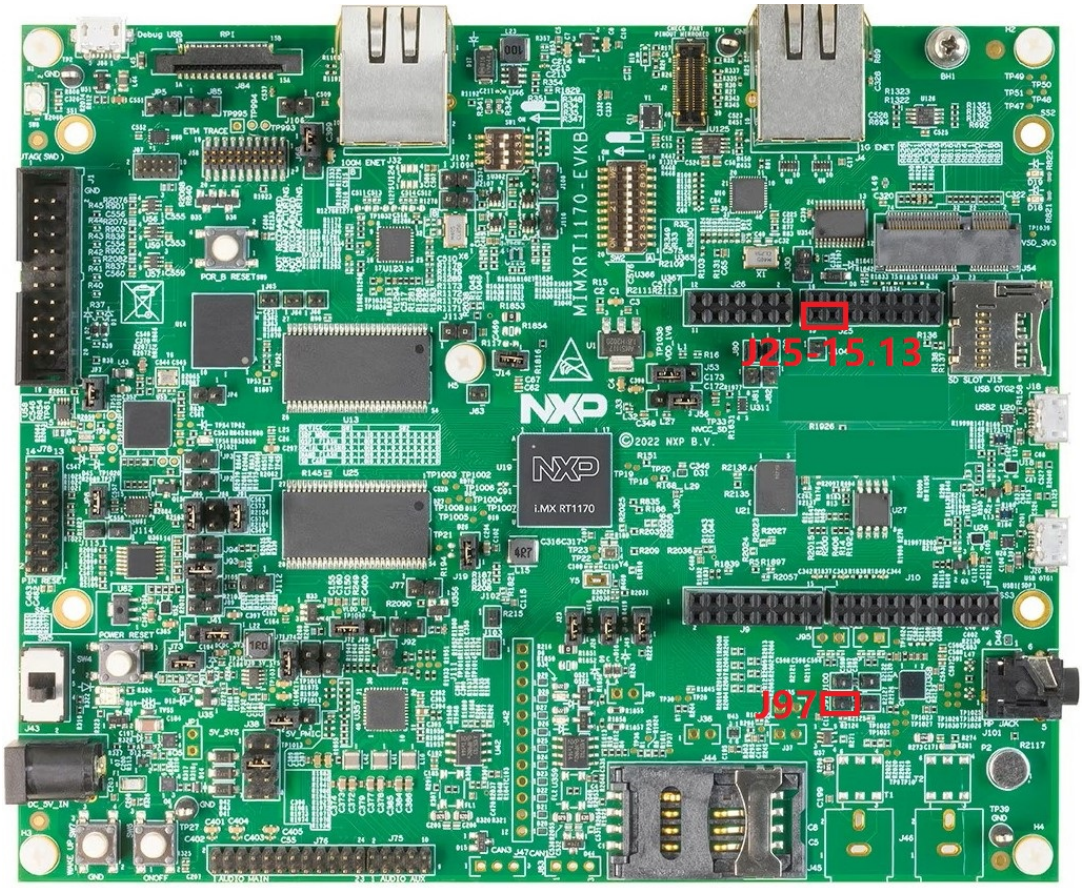
- **PCM interface rework**
  1. Disconnect header J79 and J80.

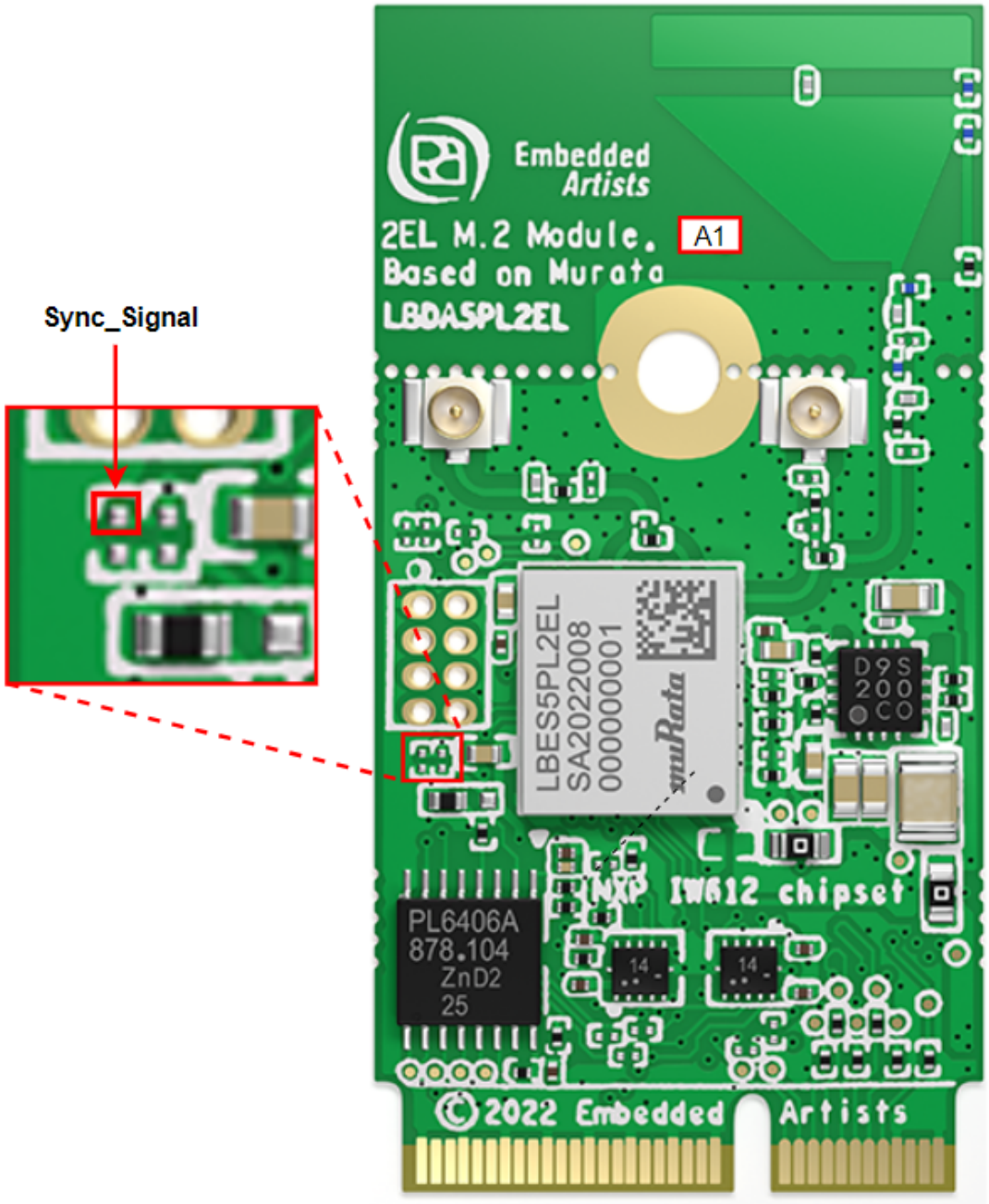
2. Connect header J81 and J82.
3. Remove resistors R1985, R1986, R1987, R1988, R1992, R1993, R1994, and R1995.
4. Solder 0 ohm resistor to R228, R229, R232, R234, and R1903.





- 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:

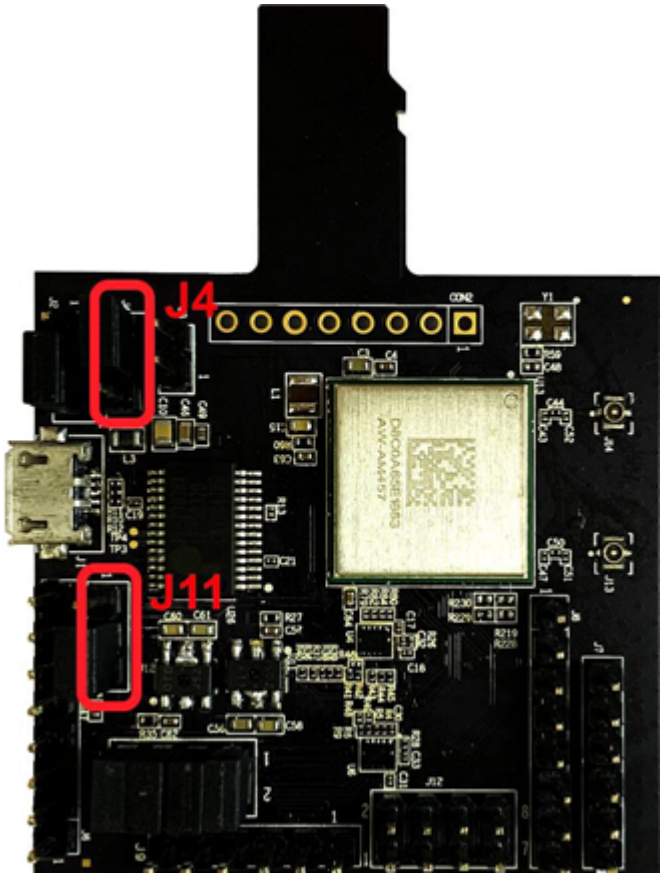
- HCI UART rework

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

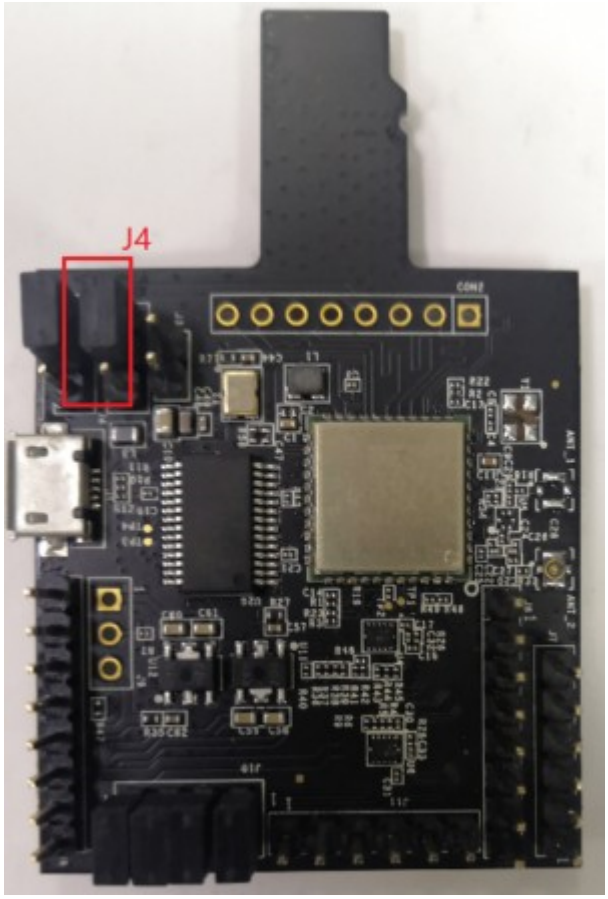
Hardware rework **HCI UART rework**

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:

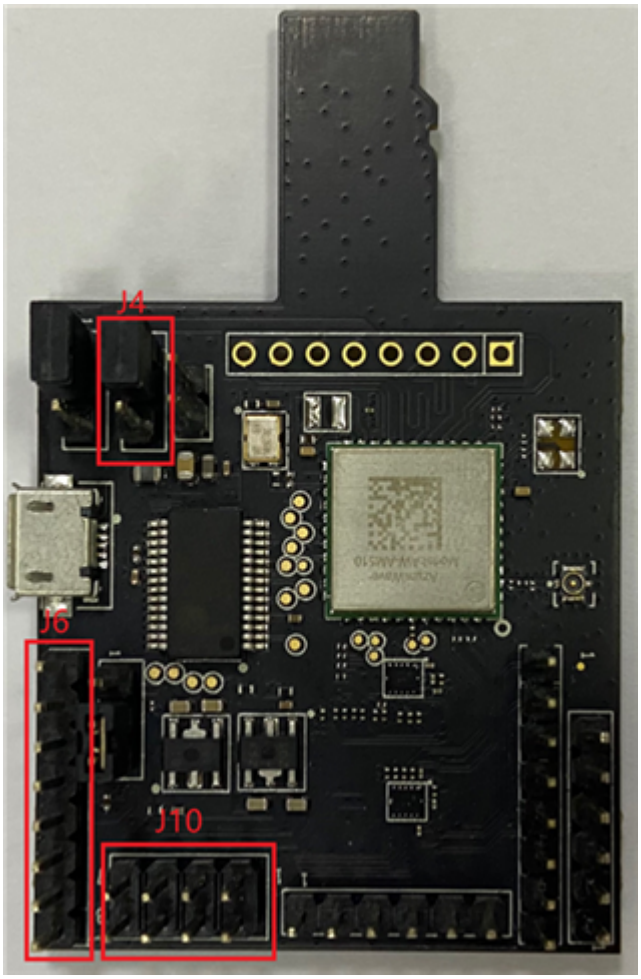
- HCI UART rework

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



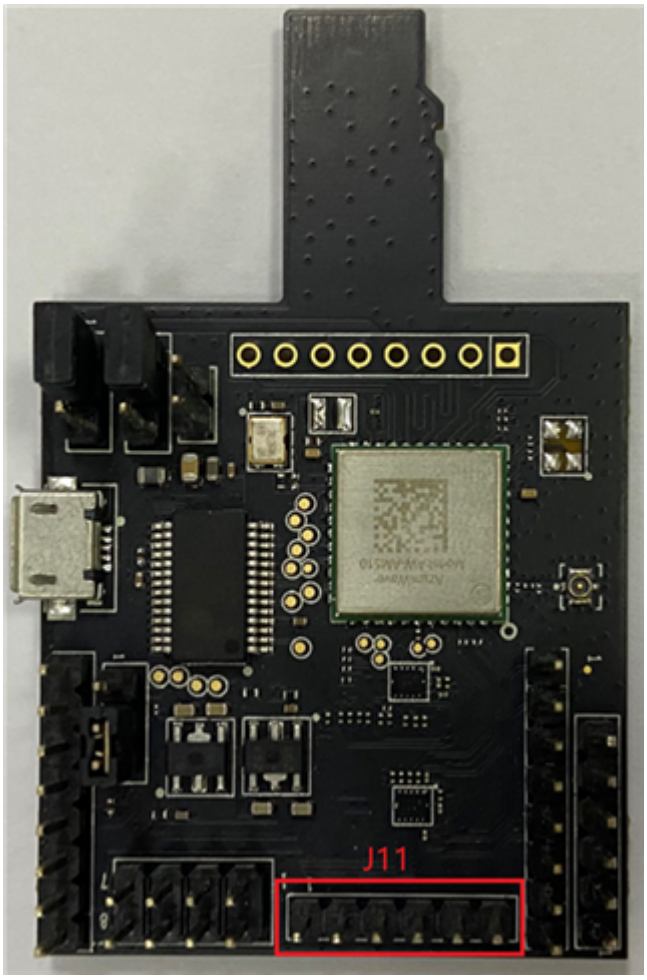
Jumper Setting:

– 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.

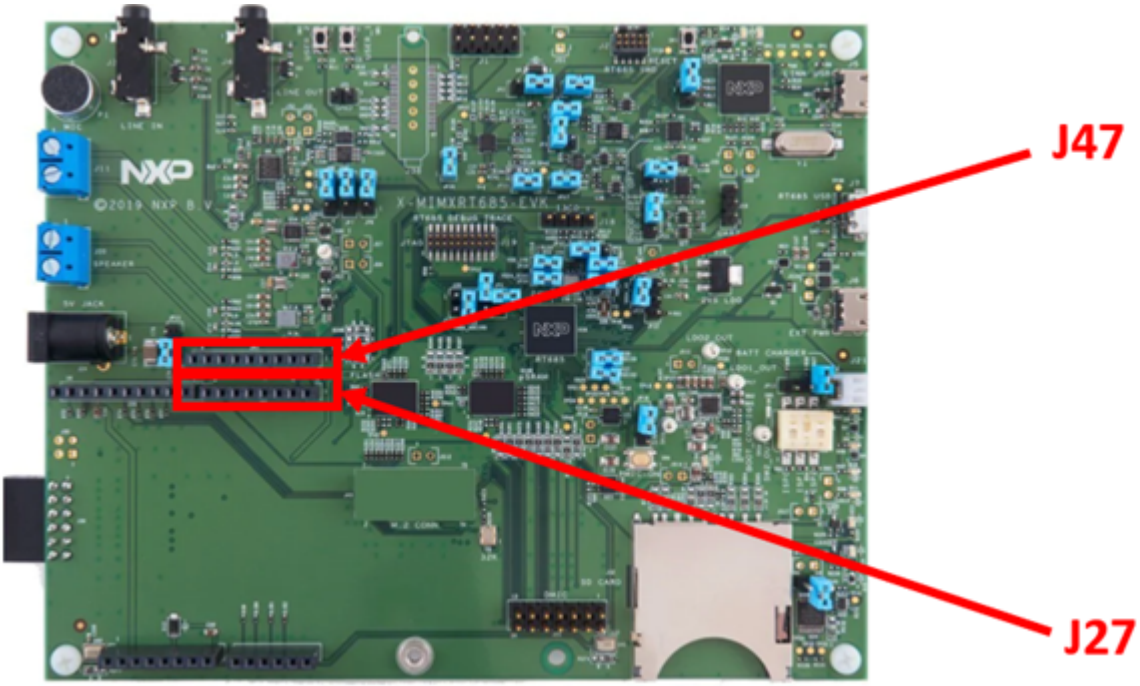
The hardware rework has one part:

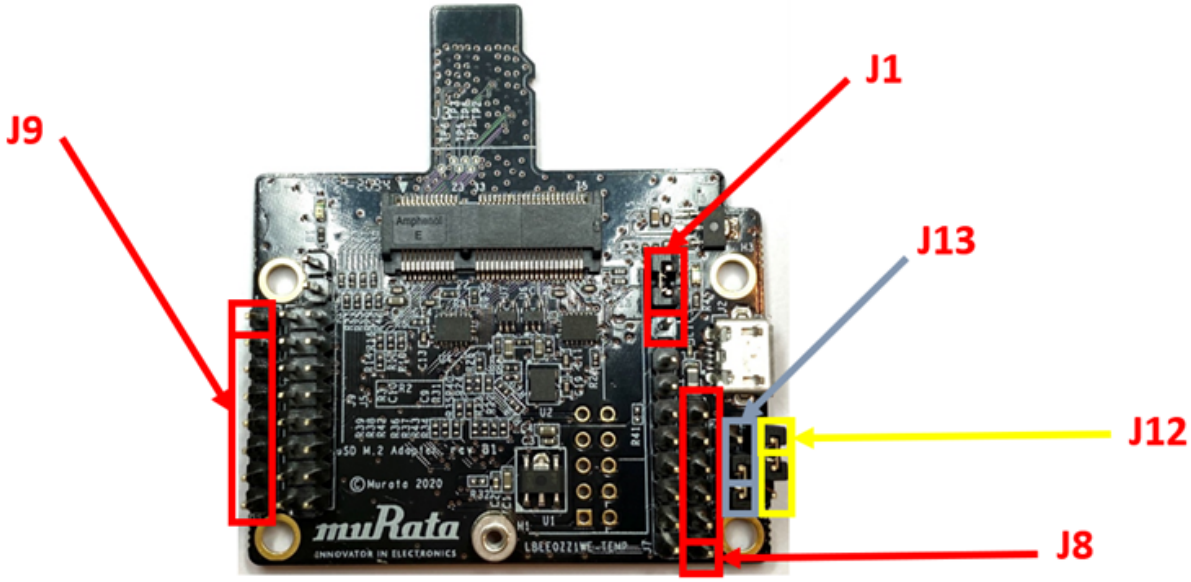
- HCI UART rework

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:

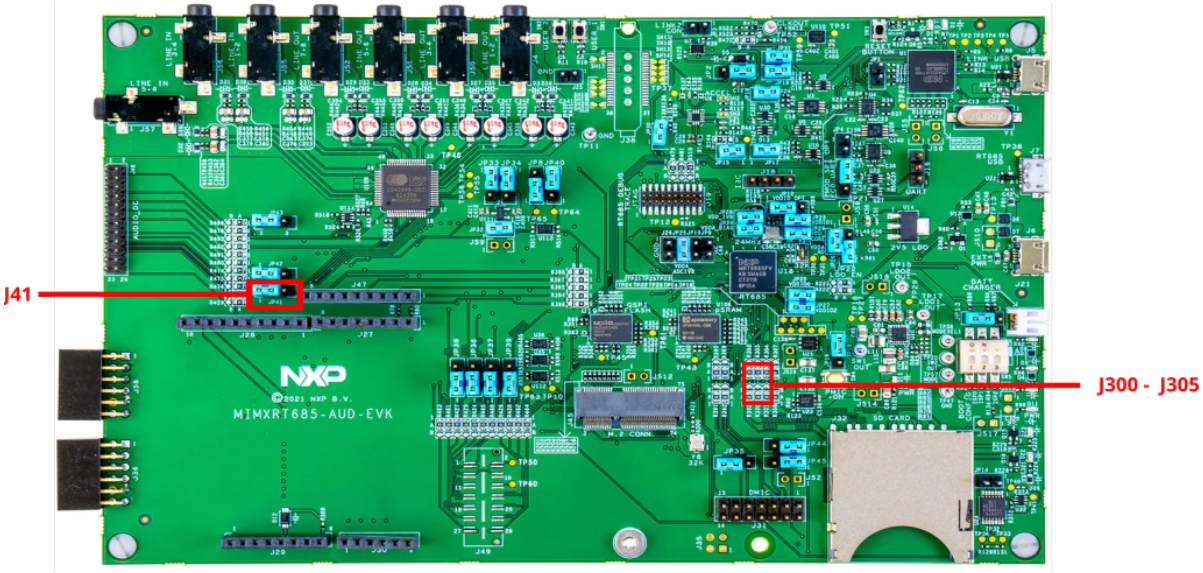
- HCI UART rework

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**

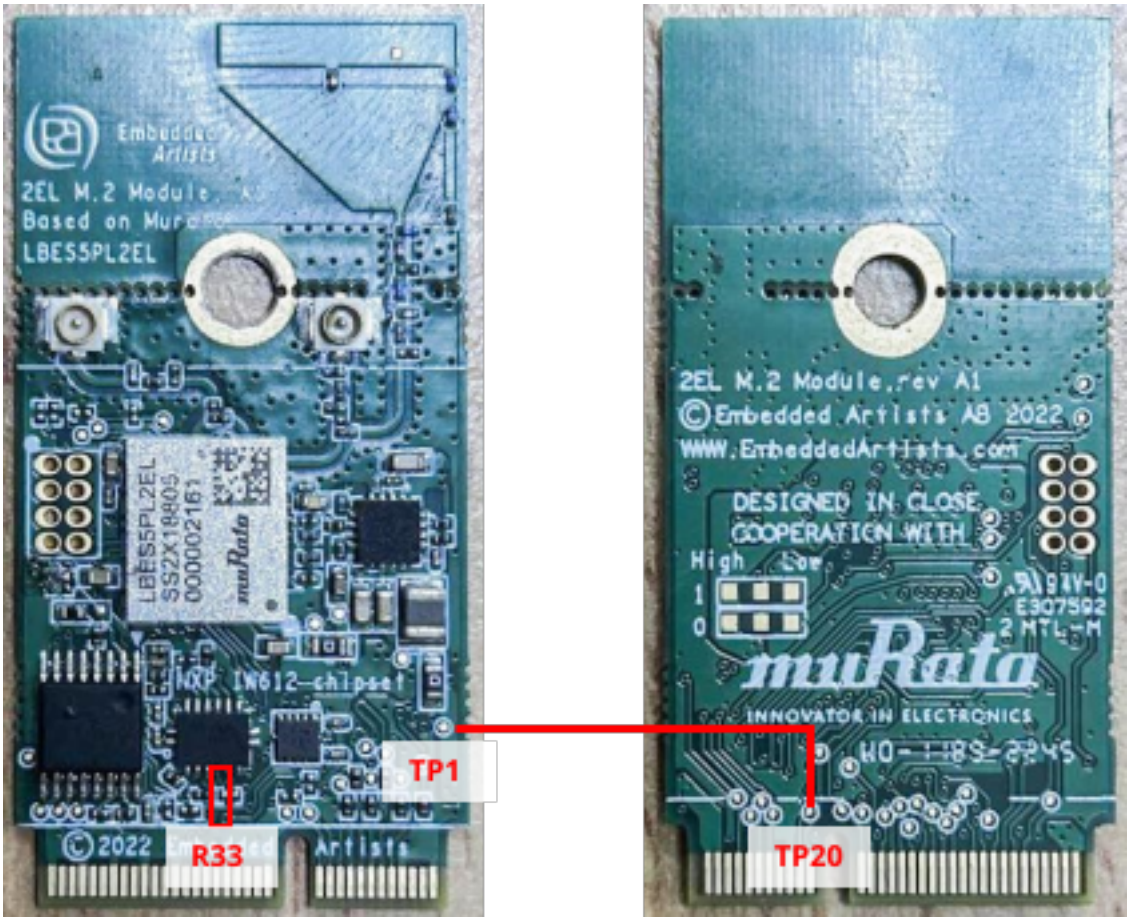
  For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

- **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.

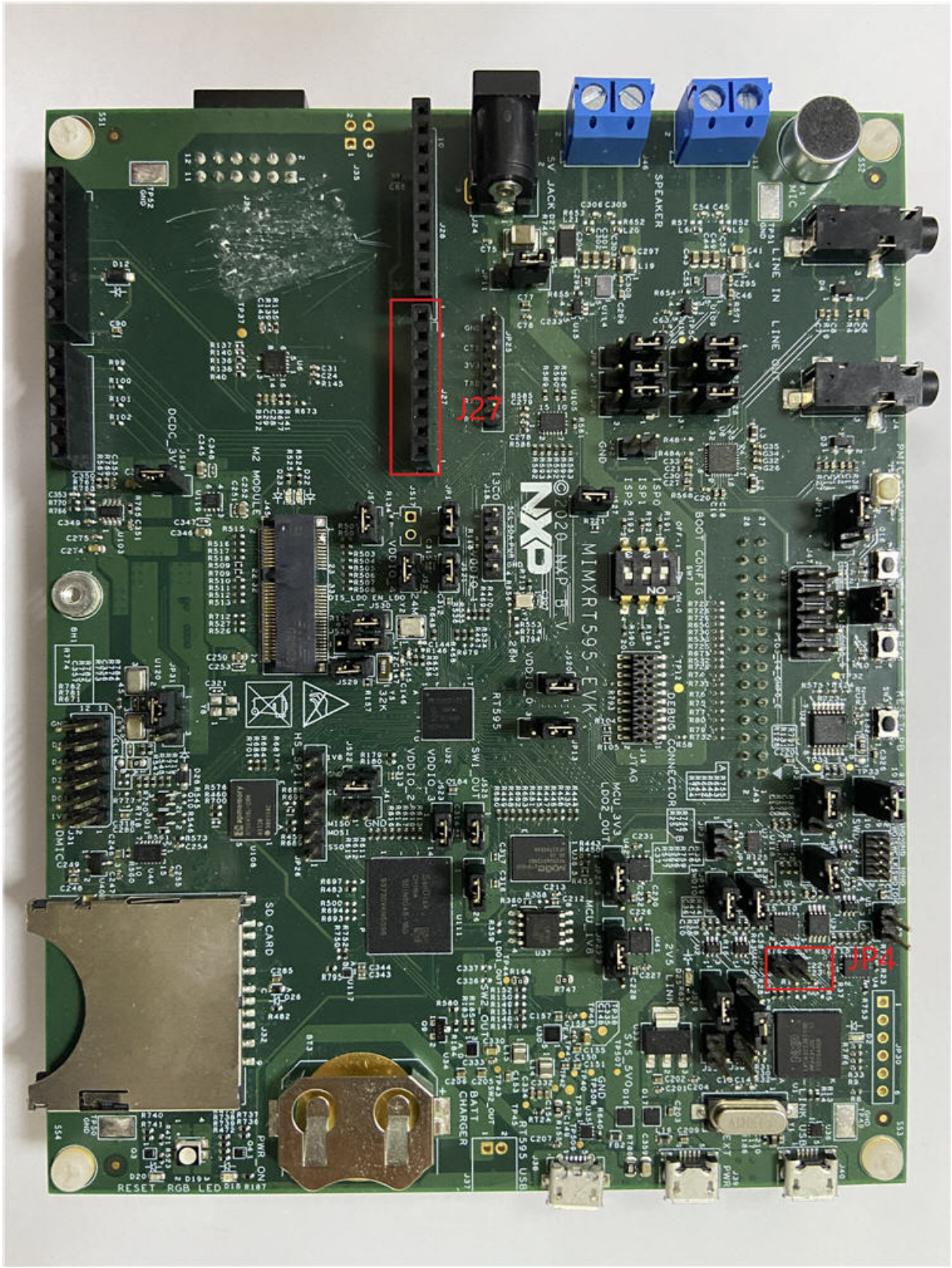
The hardware rework has one part:

- 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:

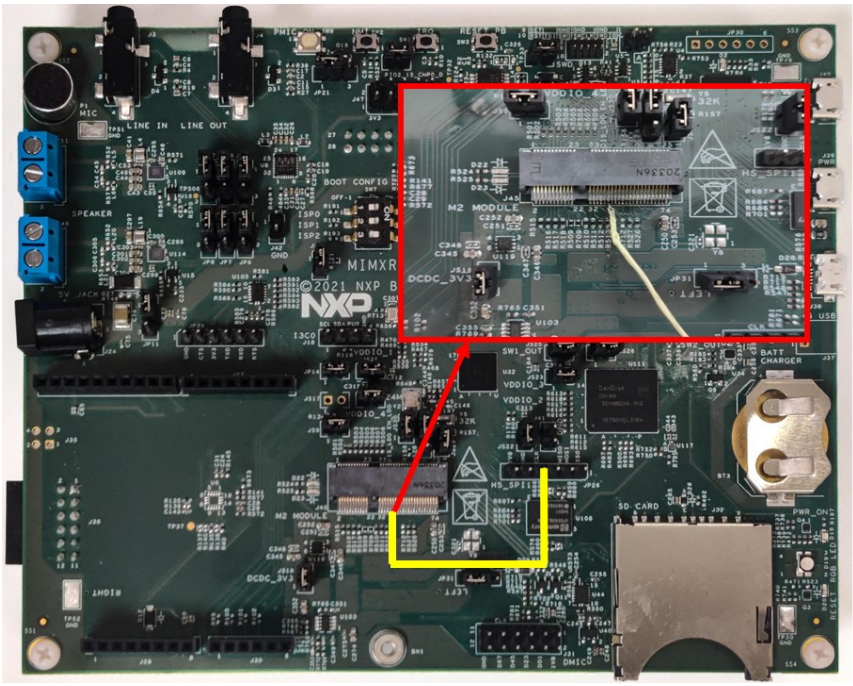
- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework **Debug console serial rework:**

For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

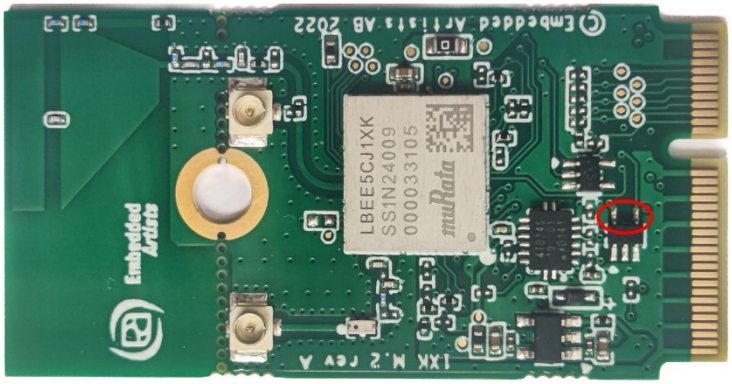
Host wake-up controller pin rework:

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*

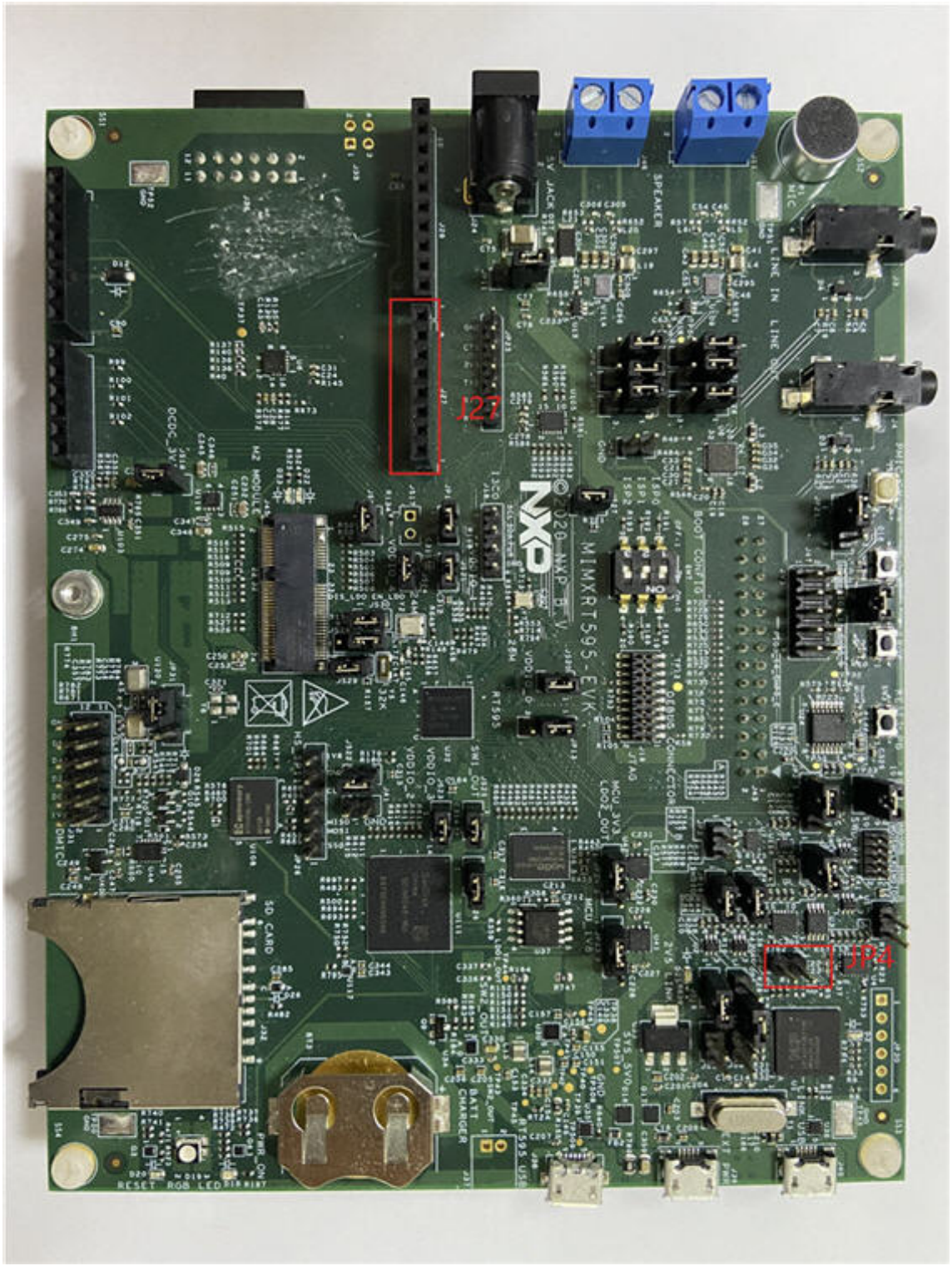
Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT595-EVK board and AW-AM510MA. The AW-AM510MA user guide is available here. The hardware rework has one part:

- 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.



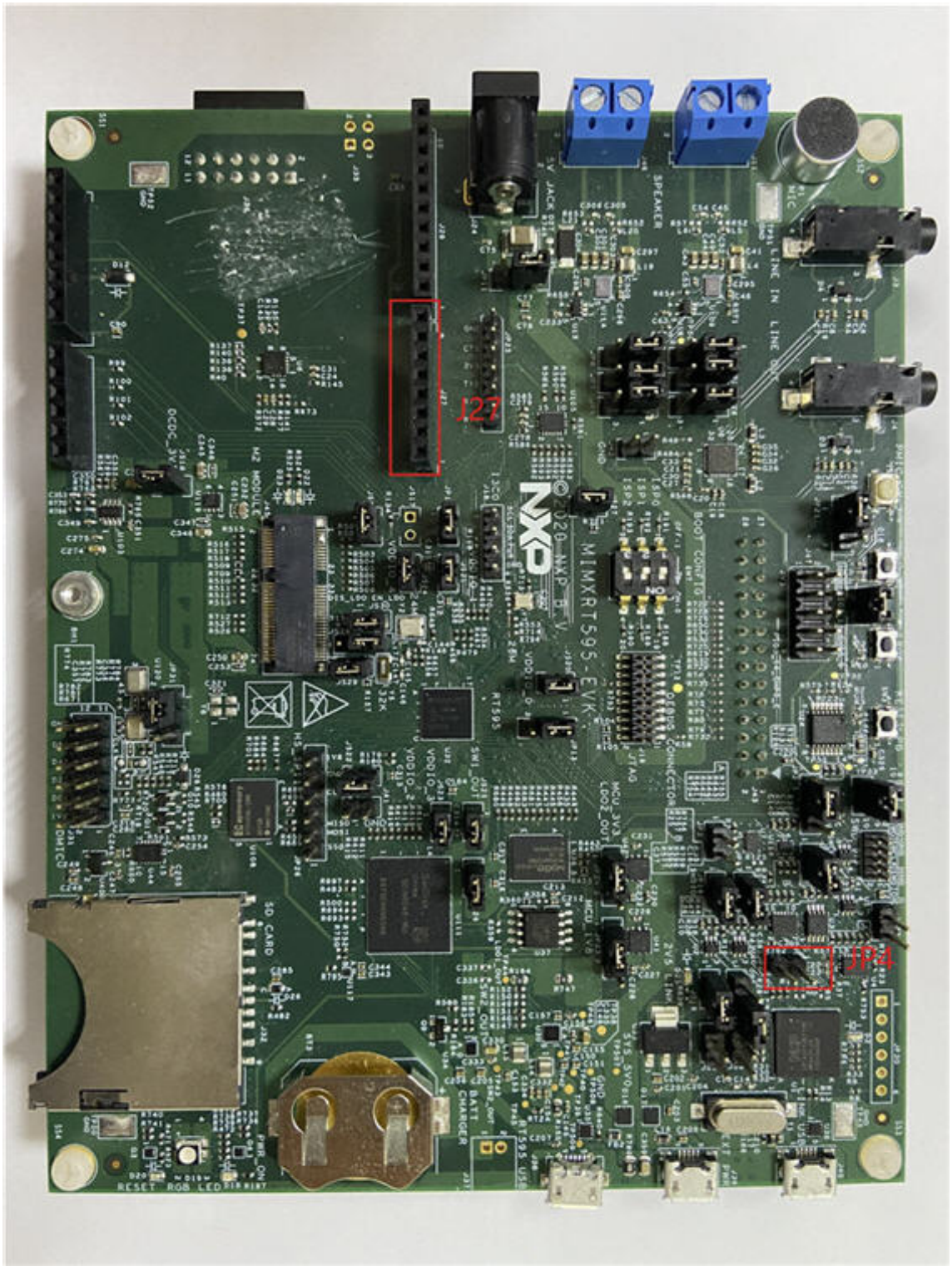
Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA*

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

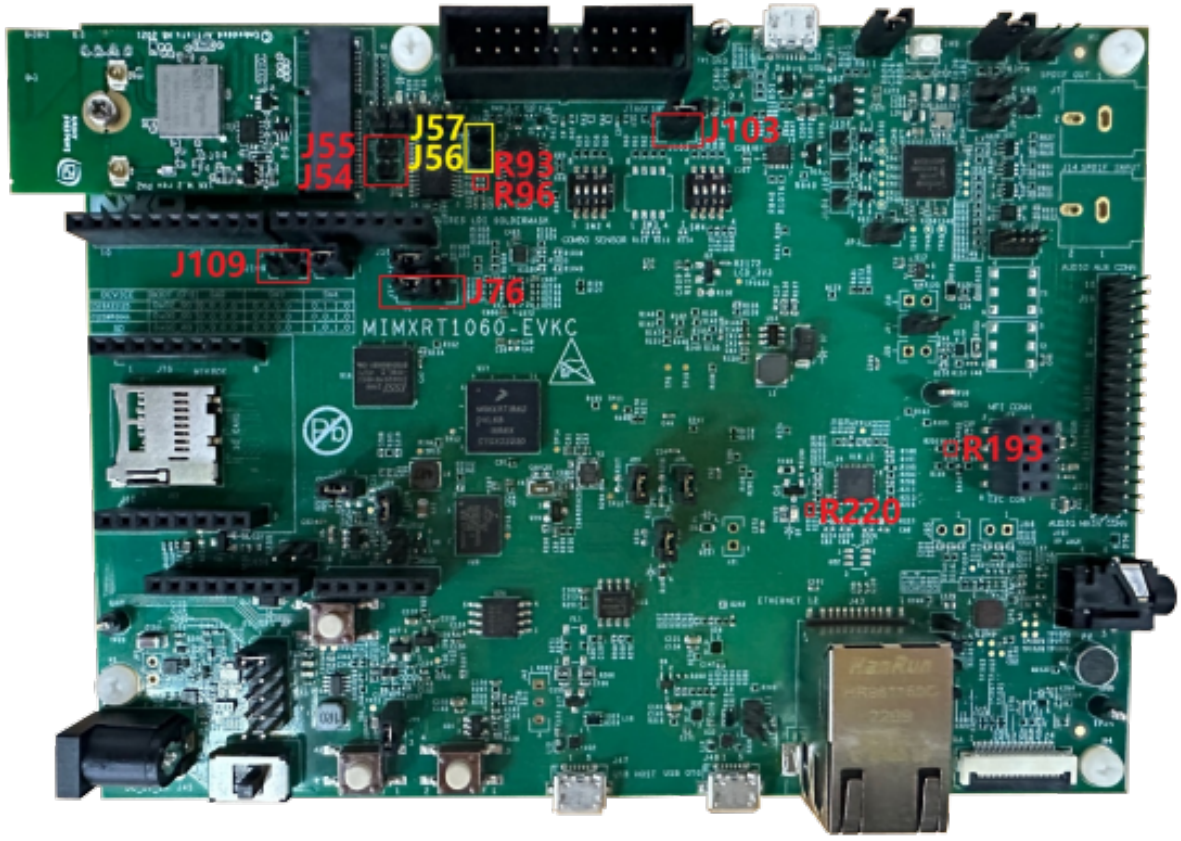
- **HCI UART rework**

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

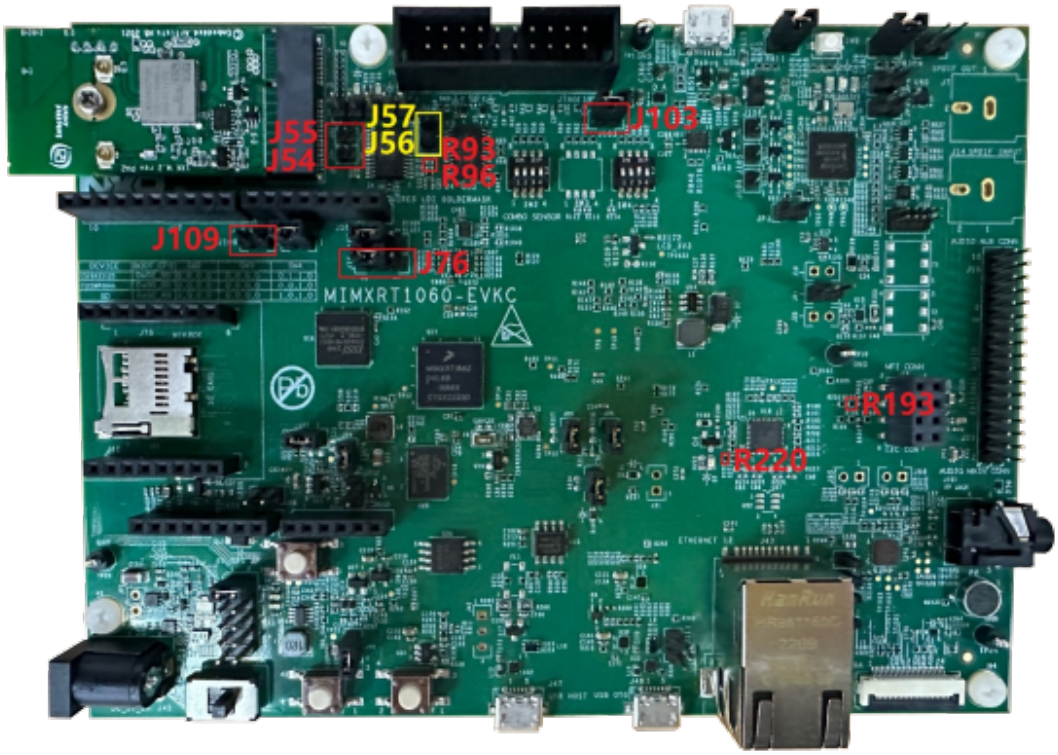
- HCI UART rework

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

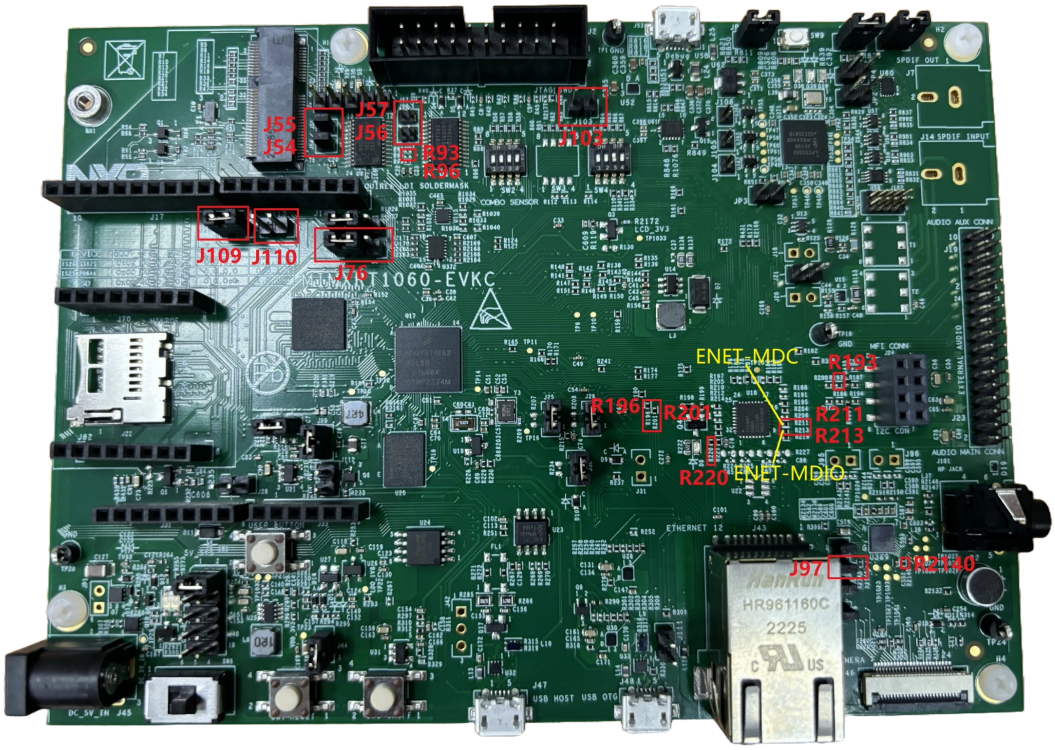
- PCM interface rework

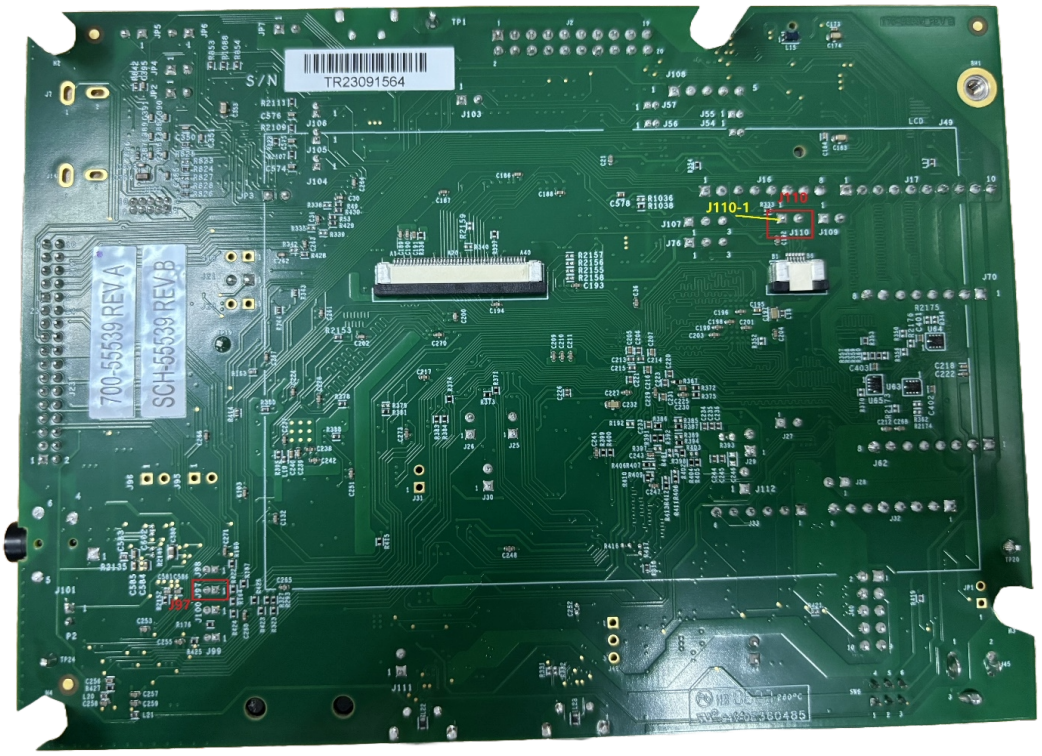
1. Remove J54 and J55, connect J56, and J57.
2. Remove R220.
3. Connect J103.

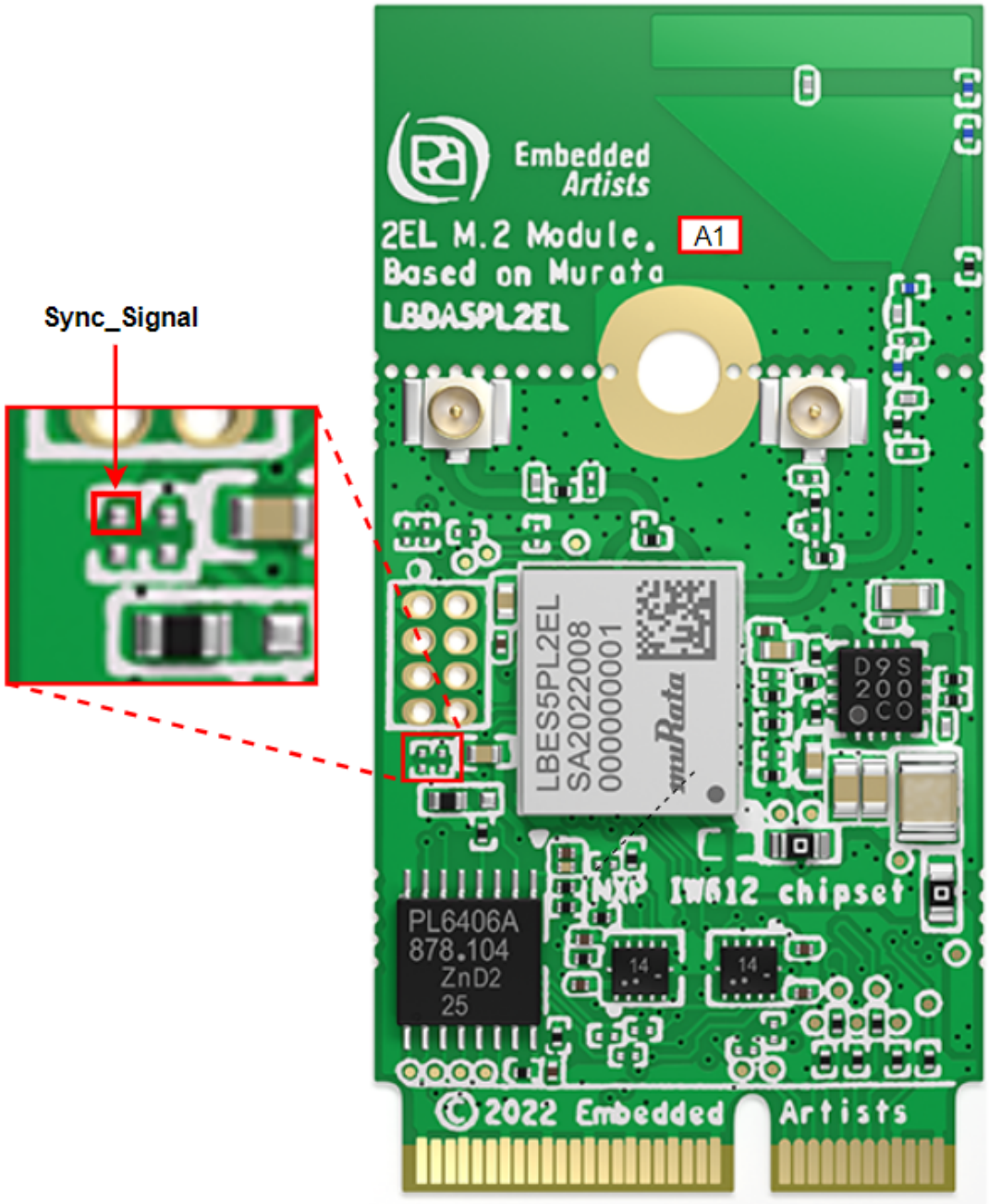
   Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



- 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.

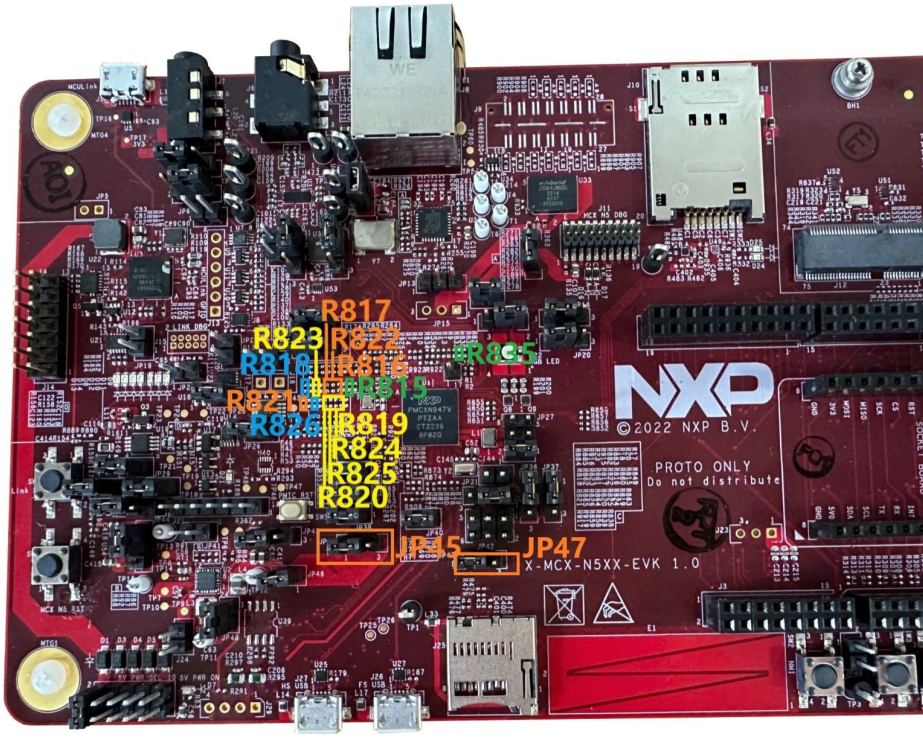
The hardware rework consists of two parts:

- 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.

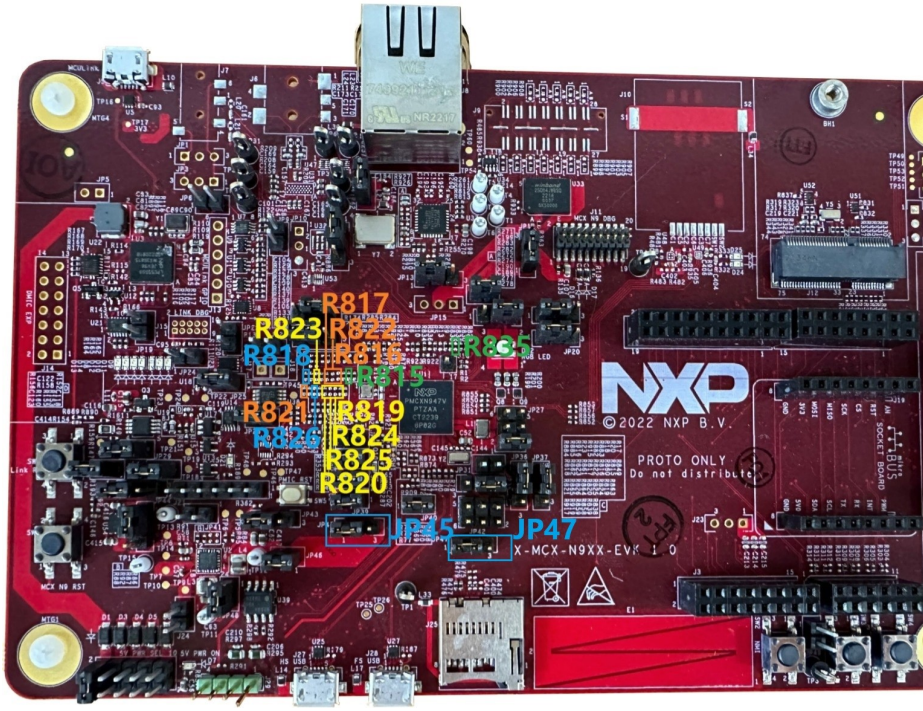
The hardware rework consists of two parts:

- 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*

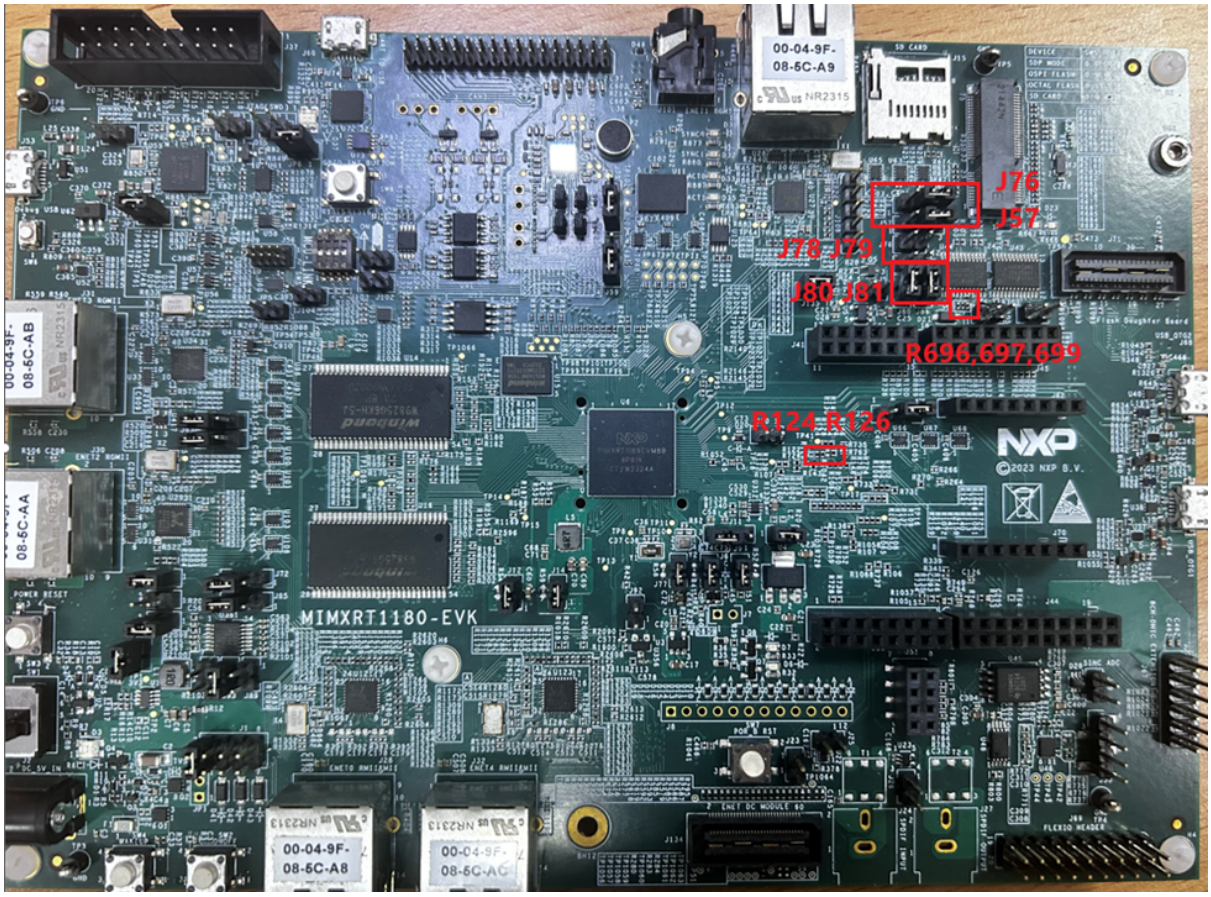
Hardware Rework Guide for MIMXRT1180 and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MIMXRT1180 board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) 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



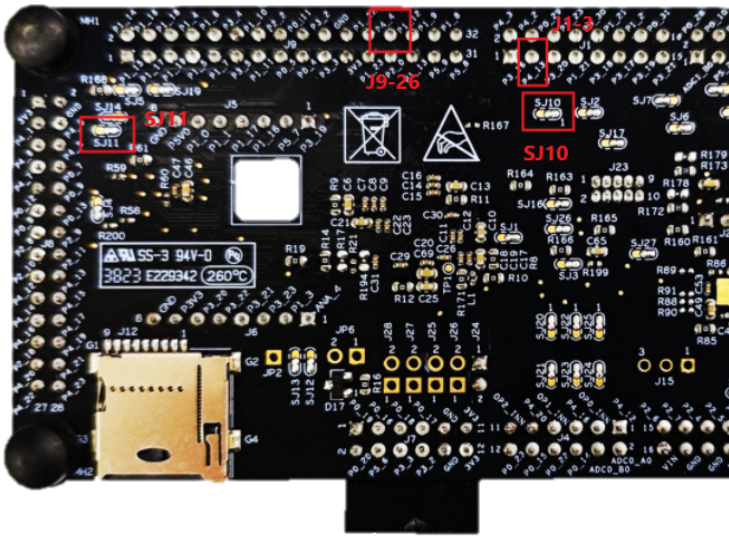
Hardware Rework Guide for FRDM-MCXN947 and X-FRDM-WIFI-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and X-FRDM-WIFI-M.2 or the Murata's 2LL EAR00500 (2LL) M.2 modules solution.

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



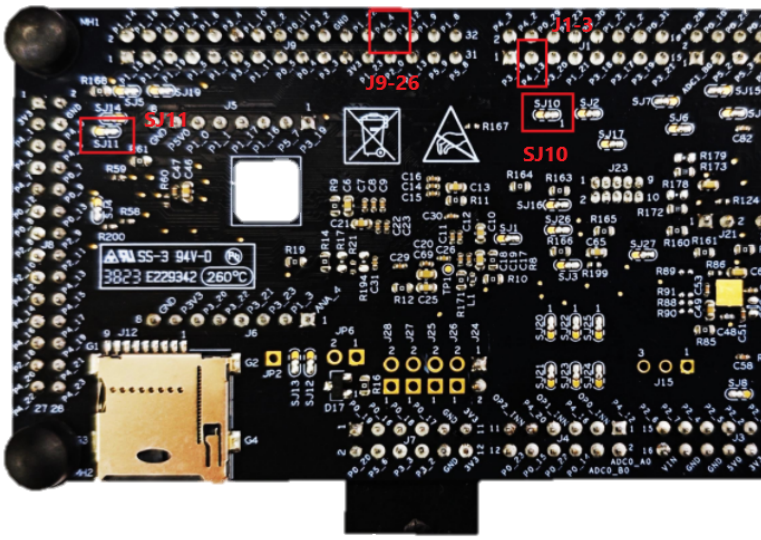
– Connect J18(On X-FRDM-WIFI-M.2) for 3.3V

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:

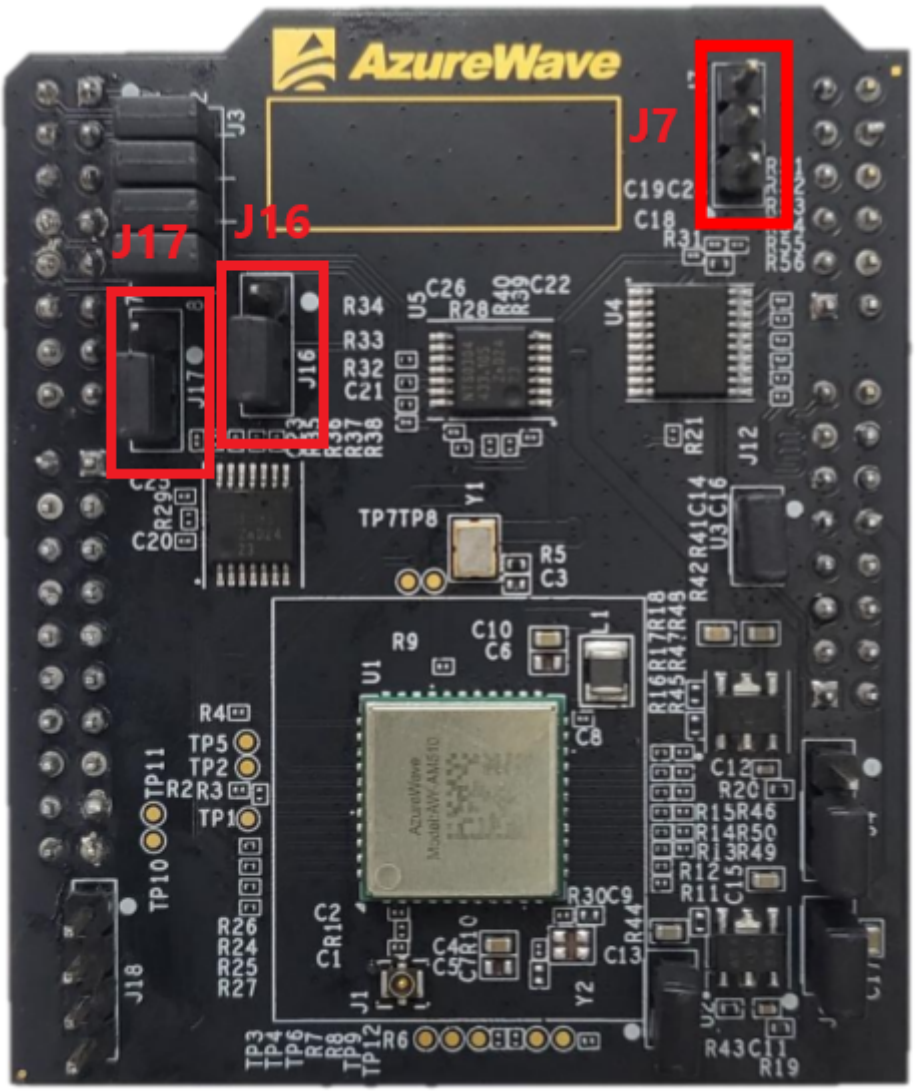
- UART interface rework
- FRDM-IW416-AW-AM510

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.

```
73 #define BOARD_FLASH_SIZE (0x800000U)
```

```
73 #define BOARD_FLASH_SIZE (0x400000U)
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes.

File name	Previous item	New item
peripheral_ht.ewp	1060 1062	1064 1064

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.

```
47 define symbol m_interrupts_start       = 0x60002000;
48 define symbol m_interrupts_end         = 0x600023FF;
49
50 define symbol m_text_start             = 0x60002400;
51 define symbol _ROM_END_                = 0x6057FFFF;
66 define exported symbol m_boot_hdr_conf_start = 0x60000000;
67 define symbol m_boot_hdr_ivt_start           = 0x60001000;
68 define symbol m_boot_hdr_boot_data_start     = 0x60001020;
69 define symbol m_boot_hdr_dcd_data_start      = 0x60001030;
95 BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash secto
96 BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x60800000 - EDGEFAST_BT_
97 ***/
```

```
47 define symbol m_interrupts_start       = 0x70002000;
48 define symbol m_interrupts_end         = 0x700023FF;
49
50 define symbol m_text_start             = 0x70002400;
51 define symbol _ROM_END_                = 0x7017FFFF;
66 define exported symbol m_boot_hdr_conf_start = 0x70000000;
67 define symbol m_boot_hdr_ivt_start           = 0x70001000;
68 define symbol m_boot_hdr_boot_data_start     = 0x70001020;
69 define symbol m_boot_hdr_dcd_data_start      = 0x70001030;
95 BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash sect
96 BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x70400000 - EDGEFAST_BT
97 ***/
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.
2. Rename the following files.

Path	Previous name	**New name**
<rt1064_install_dir>/middleware/wireless/ethermind/	middleware_edgefast_bluetooth_cmake	middleware_edgefast_bluetooth_k32w061_contro cmake

3. Make following changes.

File name	Previous item	New item
config.cmake	1060	1064
	1062	1064
flags.cmake	1062	1064
CMakeLists.txt	1060	1064
	1062	1064

4. *mflash* is used in RT1064 instead of flash_adapter,therefore, comment include(component_flexspi_nor_flash_adapter_rt1064_MIMXRT1064) in *CMakeLists.txt*.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.

```
_ROM_START_ = 0x60002400;
_ROM_END_ = 0x6057FFFF;
EDGEFAST_BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x60800000 - EDG
/*** littleFS configuration End ***/


HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x10
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x04
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400

/* Specify the memory areas */
MEMORY
{
  m_flash_config        (RX)  : ORIGIN = 0x60000000, LENGTH
  m_ivt                 (RX)  : ORIGIN = 0x60001000, LENGTH
  m_interrupts          (RX)  : ORIGIN = 0x60002000, LENGTH
```

```
_ROM_START_ = 0x70002400;
_ROM_END_ = 0x7017FFFF;
EDGEFAST_BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x70400000 - EDG
/*** littleFS configuration End ***/


HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x10
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x04
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400

/* Specify the memory areas */
MEMORY
{
  m_flash_config        (RX)  : ORIGIN = 0x70000000, LENGTH
  m_ivt                 (RX)  : ORIGIN = 0x70001000, LENGTH
  m_interrupts          (RX)  : ORIGIN = 0x70002000, LENGTH
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

MDK

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

2. Make following changes.

File name	Previous item	New item
peripheral_ht.uvprojx	1060	1064
	1062	1064

3. Copy evkmimxrt1064_flexspi_nor.ini from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/mdk/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

4. Rename MIMXRT1062xxxxx_flexspi_nor as MIMXRT1064xxxxx_flexspi_nor and make the following changes.

```
#define m_flash_config_start            0x60000000
#define m_flash_config_size             0x00001000

#define m_ivt_start                     0x60001000
#define m_ivt_size                      0x00001000

#define m_interrupts_start              0x60002000
#define m_interrupts_size               0x00000400

#define m_text_start                    0x60002400
#define _ROM_END_                       0x6057FFFF
EDGEFAST_BT_LITTLEFS_STORAGE_START_ADDRESS  (_ROM_END_ + 1)
EDGEFAST_BT_LITTLEFS_STORAGE_END_ADDRESS  (0x60800000)
EDGEFAST_BT_LITTLEFS_STORAGE_SECTOR_SIZE    (0x1000)
```

```
#define m_flash_config_start            0x70000000
#define m_flash_config_size             0x00001000

#define m_ivt_start                     0x70001000
#define m_ivt_size                      0x00001000

#define m_interrupts_start              0x70002000
#define m_interrupts_size               0x00000400

#define m_text_start                    0x70002400
#define _ROM_END_                       0x7017FFFF
EDGEFAST_BT_LITTLEFS_STORAGE_START_ADDRESS  (_ROM_END_ + 1)
EDGEFAST_BT_LITTLEFS_STORAGE_END_ADDRESS  (0x70400000)
EDGEFAST_BT_LITTLEFS_STORAGE_SECTOR_SIZE    (0x1000)
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

Migrate examples from RT1060EVKC to RT1064 This topic describes the Common steps and the steps to migrate with the IAR, Arm GCC, and MDK toolchains.

Common steps

1. Download SDK_2.14.0_EVKC-MIMXRT1060 and SDK_2.14.0_EVK-MIMXRT1064.
2. Copy the following folders from the RT1060EVKC package to the RT1064 package: <install_dir>/middleware/edgefast_bluetooth/ <install_dir>/middleware/wireless/ethermind.
3. Create a new folder named edgefast_bluetooth_examples/ under <rt1064_install_dir>/boards/evkmimxrt1064/.
4. Copy the entire folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/.
5. Copy clock_config.[c/h] and board.c from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/ to <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/ to replace the previous files.

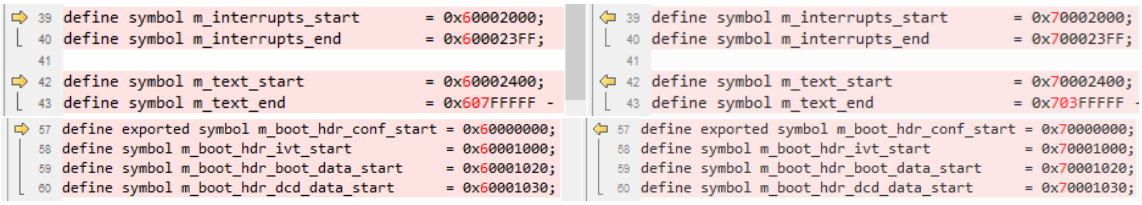
Parent topic:*Migrate examples from RT1060EVKC to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes in the listed order.

File name	Previous item	New item
peripheral_ht.ewp	1062	1064
mflash/evkcmimxrt1060	mflash/mimxrt1064	
evkcmimxrt1060	evkmimxrt1064	
6B	6A	

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.



Parent topic:*Migrate examples from RT1060EVKC to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.

2. Copy folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/template/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/ and rename the files.

Path	Previous name	New name
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/	middleware_edgefast_bluetooth_mcux_linker_template_evkcmimxrt1060.cmake	middleware_edgefast_bluetooth_mcux_linker_template_evkmimxrt1064.cmake
	middleware_edgefast_bluetooth_sdio_template_evkcmimxrt1060.cmake	middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064.cmake

3. Add the following content to<rt1064_install_dir>/devices/MIMXRT1064/all_lib_device.cmake at appropriate location.

```
...
  ${CMAKE_CURRENT_LIST_DIR}/../../boards
  ${CMAKE_CURRENT_LIST_DIR}/../../boards/evkmimxrt1064/edgefast_bluetooth_examples/
↪template
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/edgefast_bluetooth
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/wireless/ethermind
...
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci_uart)
include_if_use(middleware_edgefast_bluetooth_config_ethermind)
include_if_use(middleware_edgefast_bluetooth_config_template)
include_if_use(middleware_edgefast_bluetooth_extension_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_k32w061_controller)
include_if_use(middleware_edgefast_bluetooth_mcux_linker_template_ evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_pal)
include_if_use(middleware_edgefast_bluetooth_pal_db_gen_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_host_msd_fatfs_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_platform_ethermind)
include_if_use(middleware_edgefast_bluetooth_porting)
include_if_use(middleware_edgefast_bluetooth_porting_atomic)
include_if_use(middleware_edgefast_bluetooth_porting_list)
include_if_use(middleware_edgefast_bluetooth_porting_net)
include_if_use(middleware_edgefast_bluetooth_porting_toolchain)
include_if_use(middleware_edgefast_bluetooth_porting_work_queue)
include_if_use(middleware_edgefast_bluetooth_profile_bas)
include_if_use(middleware_edgefast_bluetooth_profile_dis)
include_if_use(middleware_edgefast_bluetooth_profile_fmp)
include_if_use(middleware_edgefast_bluetooth_profile_hps)
include_if_use(middleware_edgefast_bluetooth_profile_hrs)
include_if_use(middleware_edgefast_bluetooth_profile_hts)
include_if_use(middleware_edgefast_bluetooth_profile_ipsp)
include_if_use(middleware_edgefast_bluetooth_profile_pxr)
include_if_use(middleware_edgefast_bluetooth_profile_tip)
include_if_use(middleware_edgefast_bluetooth_profile_wu)
include_if_use(middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_shell)
include_if_use(middleware_edgefast_bluetooth_shell_ble)
include_if_use(middleware_edgefast_bluetooth_template)
include_if_use(middleware_edgefast_bluetooth_wifi_nxp_controller_base)...
```

4. Make the following changes in the listed order.

File name	Previous item	New item
config.cmake	MIMXRT1	MIMXRT1064xxxxA
mflash_evkcmimxrt1060	mflash_rt	
1062	1064	
evkcmimxrt1060	evk-mimxrt10	
flags.cmake	1062	1064
6B	6A	
CMakeLists.txt	1062	1064
<rt1064_install_dir>/middleware/edgefast_bluetooth/middleware_edgefast_bluetooth_template.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_common_ethermind_hci_uart.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_k32w061_controller.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_wifi_nxp_controller_base.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_mcux_cmake	1062	1064 mxrt1064.
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_sdio_t cmake	1062	1064 1.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.

```
39 m_text_start         = 0x60002400;
40 m_text_size          = 0x007FDC00 - LITTLEFS_REGION_SIZE;
41
42 HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x1000;
43 STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x0400;
44 VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400 : 0;
45
46 /* Specify the memory areas */
47 MEMORY
48 {
49   m_flash_config        (RX)  : ORIGIN = 0x60000000, LENGTH = 0x00001000
50   m_ivt                 (RX)  : ORIGIN = 0x60001000, LENGTH = 0x00001000
51   m_interrupts          (RX)  : ORIGIN = 0x60002000, LENGTH = 0x00000400
```

```
39 m_text_start         = 0x70002400;
40 m_text_size          = 0x003FDC00 - LITTLEFS_REGION_SIZE;
41
42 HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x1000;
43 STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x0400;
44 VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400 : 0;
45
46 /* Specify the memory areas */
47 MEMORY
48 {
49   m_flash_config        (RX)  : ORIGIN = 0x70000000, LENGTH = 0x00001000
50   m_ivt                 (RX)  : ORIGIN = 0x70001000, LENGTH = 0x00001000
51   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

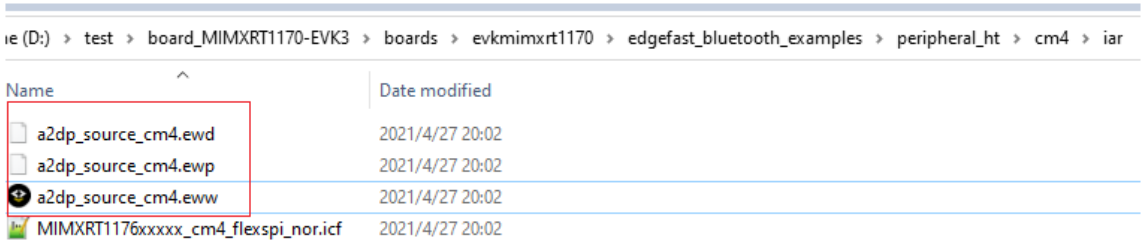
4. Adjust project settings
5. Delete function

In this document, the peripheral_ht example is used to demonstrate how to enable EdgeFast examples on M4 core with IAR and ARMGCC.

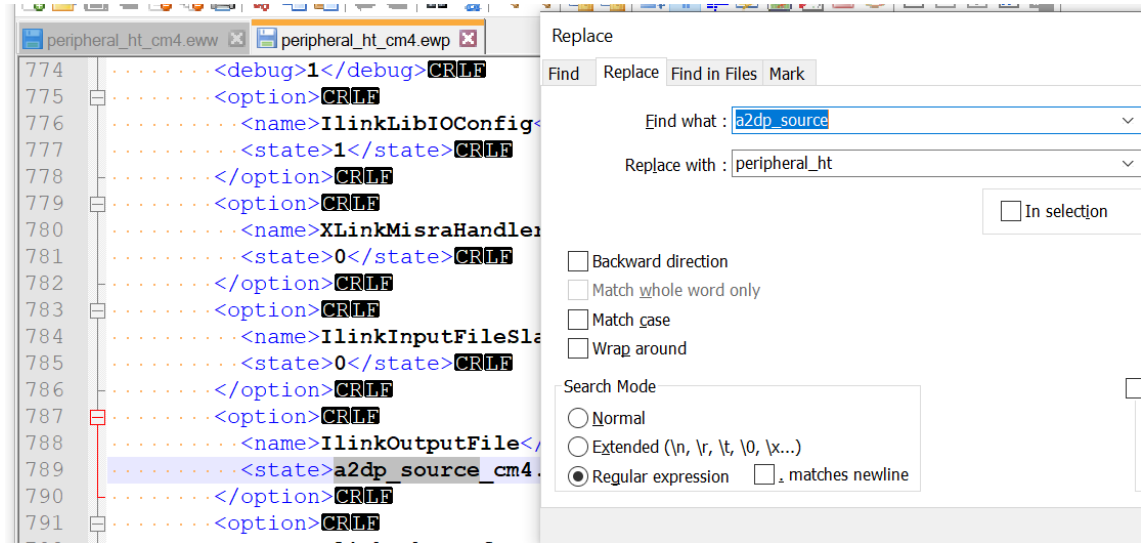
IAR This section describes the steps to create an M4 project with IAR, rearrange source and project files, adjust project settings, and delete function.

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.



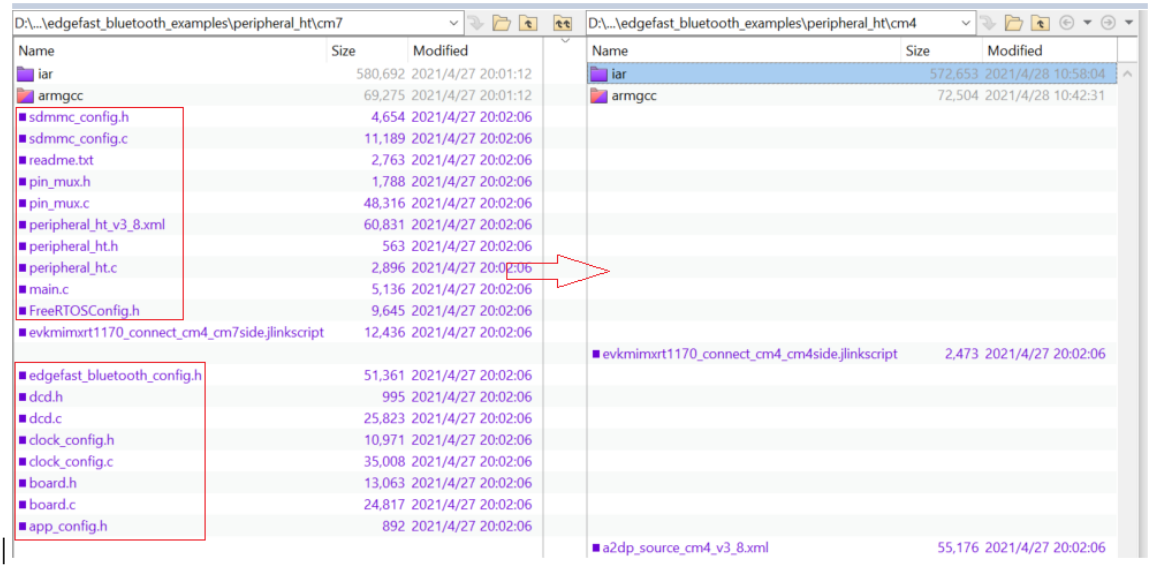
4. Open the files *peripheral_ht_cm4.eww* and *peripheral_ht_cm4.ewp* with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio Code.
5. Search and replace all *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.



Parent topic:*IAR*

Rearrange source files To rearrange source files, perform the following steps:

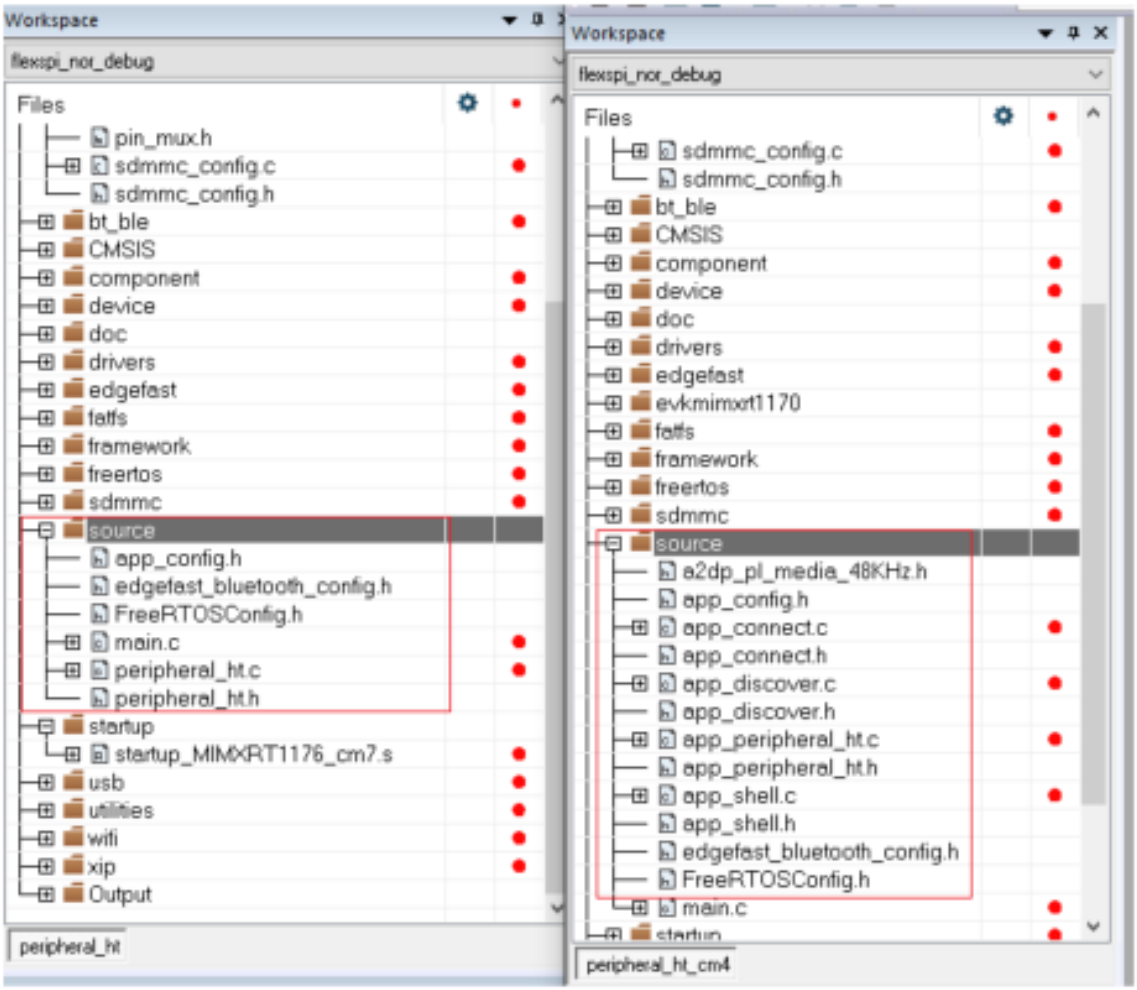
1. Open the folder *cm4* in the directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peri* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h from the folder *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm7*| to the folder <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*.



Parent topic:*IAR*

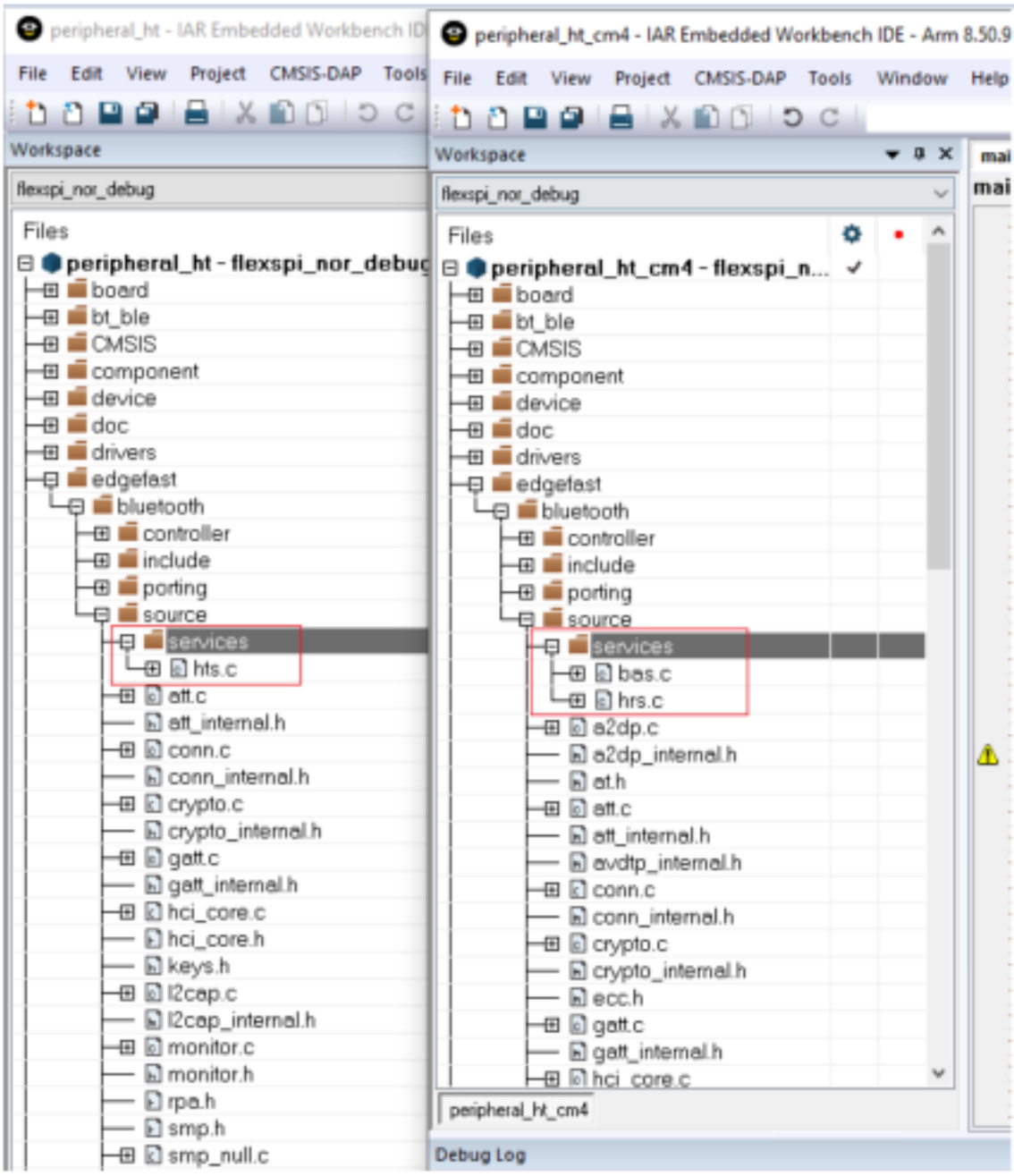
Rearrange project files To rearrange project files, perform the following steps:

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.

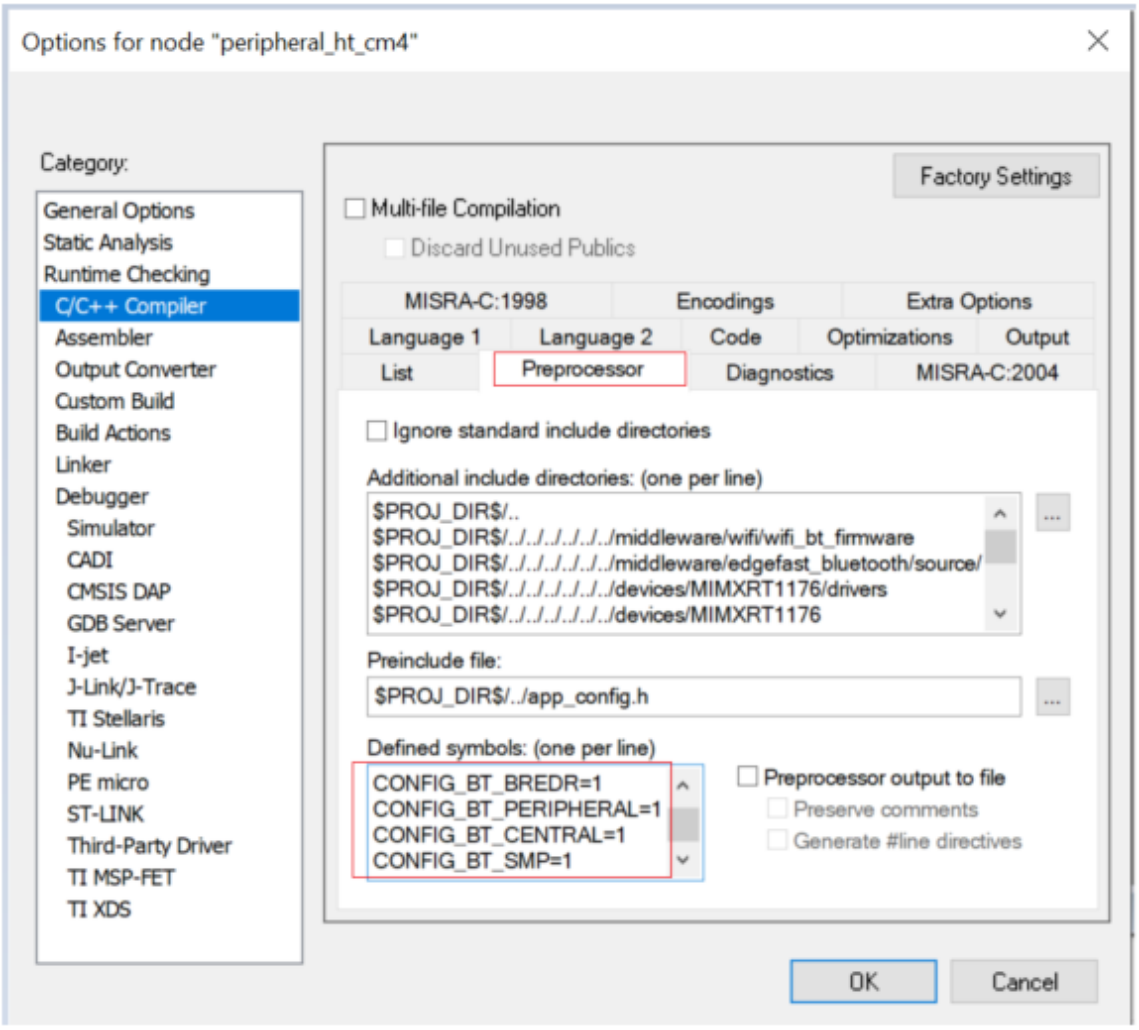
   The peripheral hts profile is in the *services* folder. Add the hts.c file to the *services* group of the *cm4* folder.



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.

|[](../images/image7.png "Compare 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*>

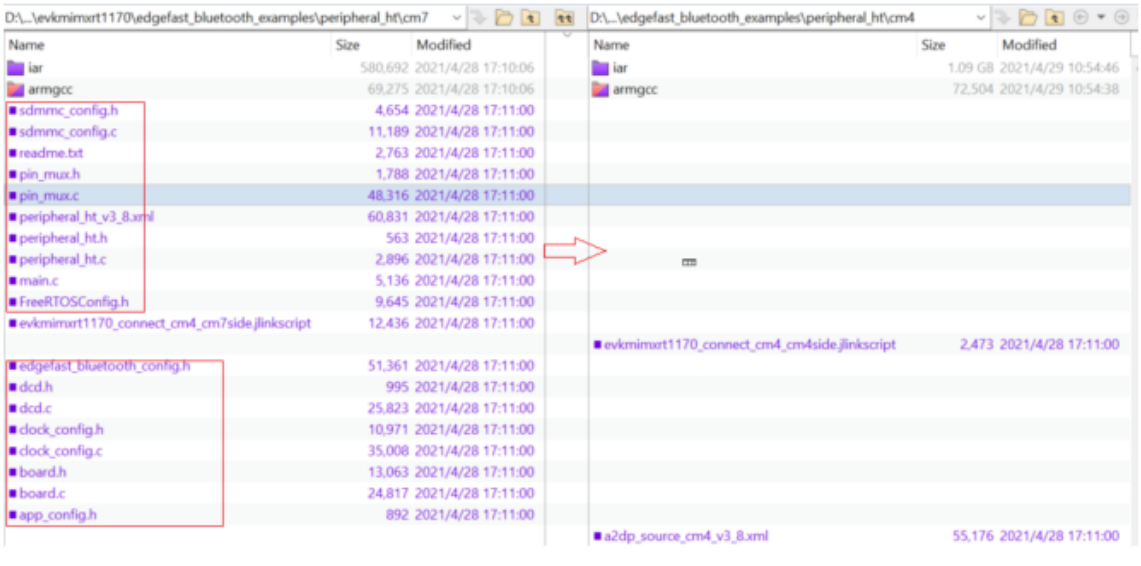
boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source \cm4 into*<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.

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:

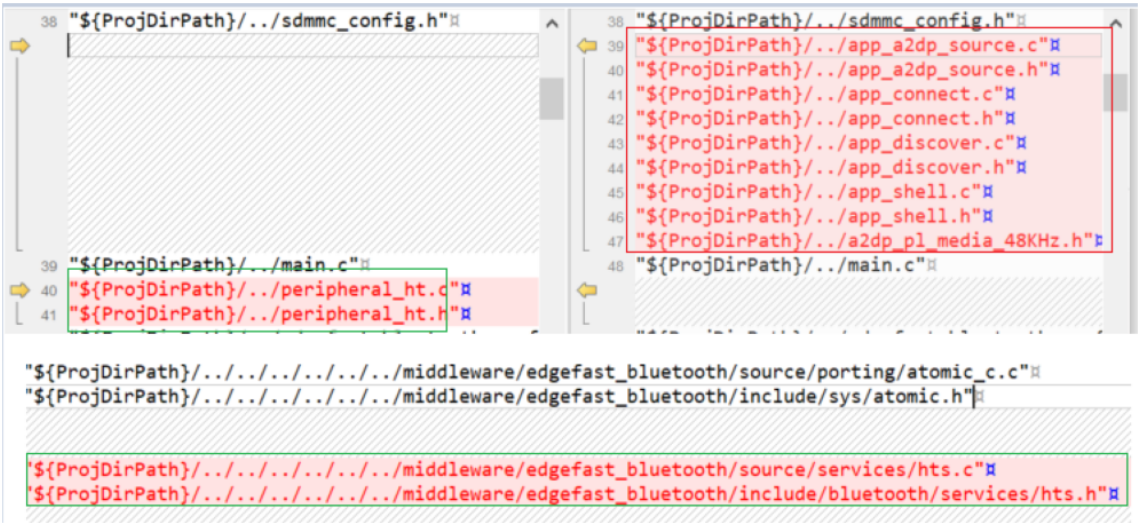
1. Open the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h in the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7* 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:

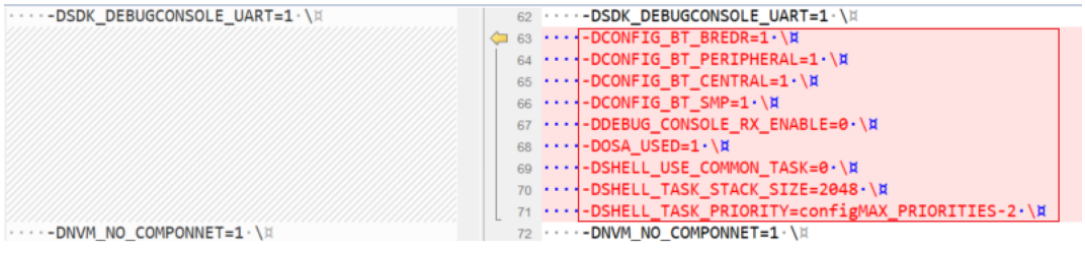
1. Open the *CMakeLists.txt* of the two examples respectively. The two files are in the *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7\armgcc* and *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\armgcc* folders respectively.
2. Search 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.

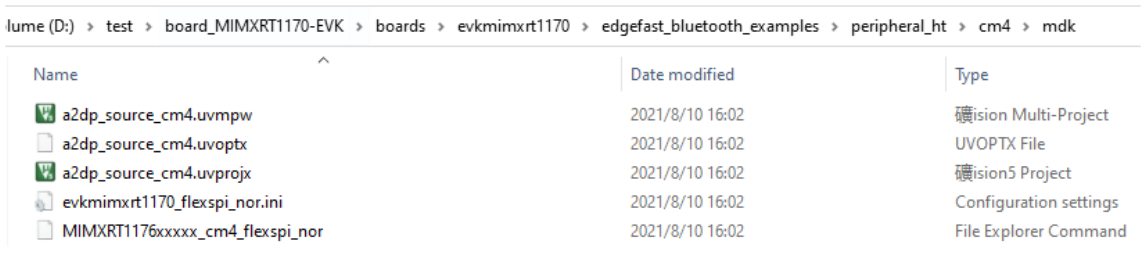
On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:*Arm GCC*

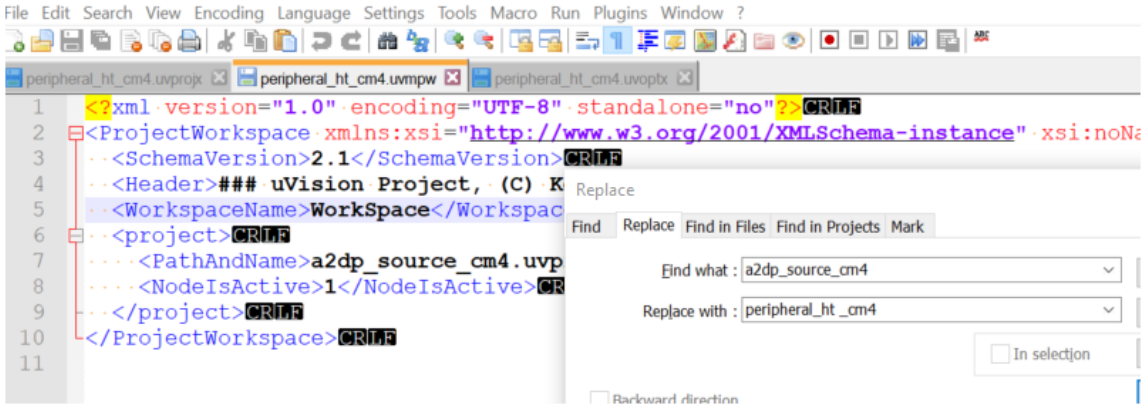
MDK This section describes the steps to create an M4 project with MDK, rearrange source and project files, adjust project settings, and delete function.

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*



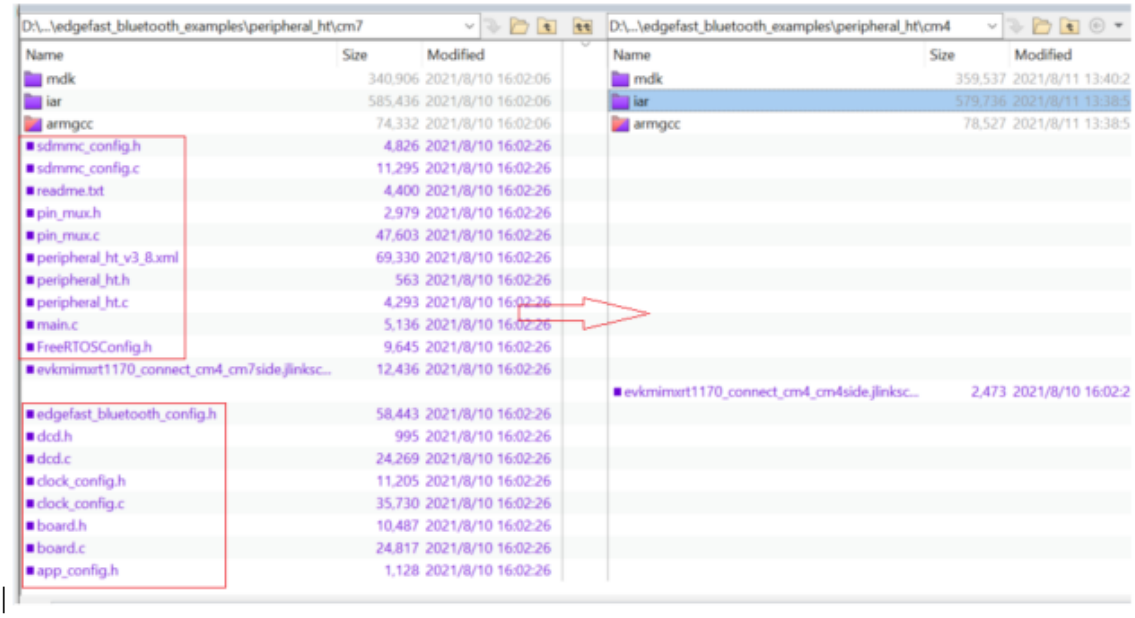
3. Change the filename *a2dp_source_cm4* to *peripheral_ht_cm4* respectively.
4. Open the files *peripheral_ht_cm4.**uvmpw* and *peripheral_ht_cm4.* *uvoptx, peripheral_ht_cm4.uvprojx*with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio code.
5. Search and replace *a2dp_source_cm4* 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*.

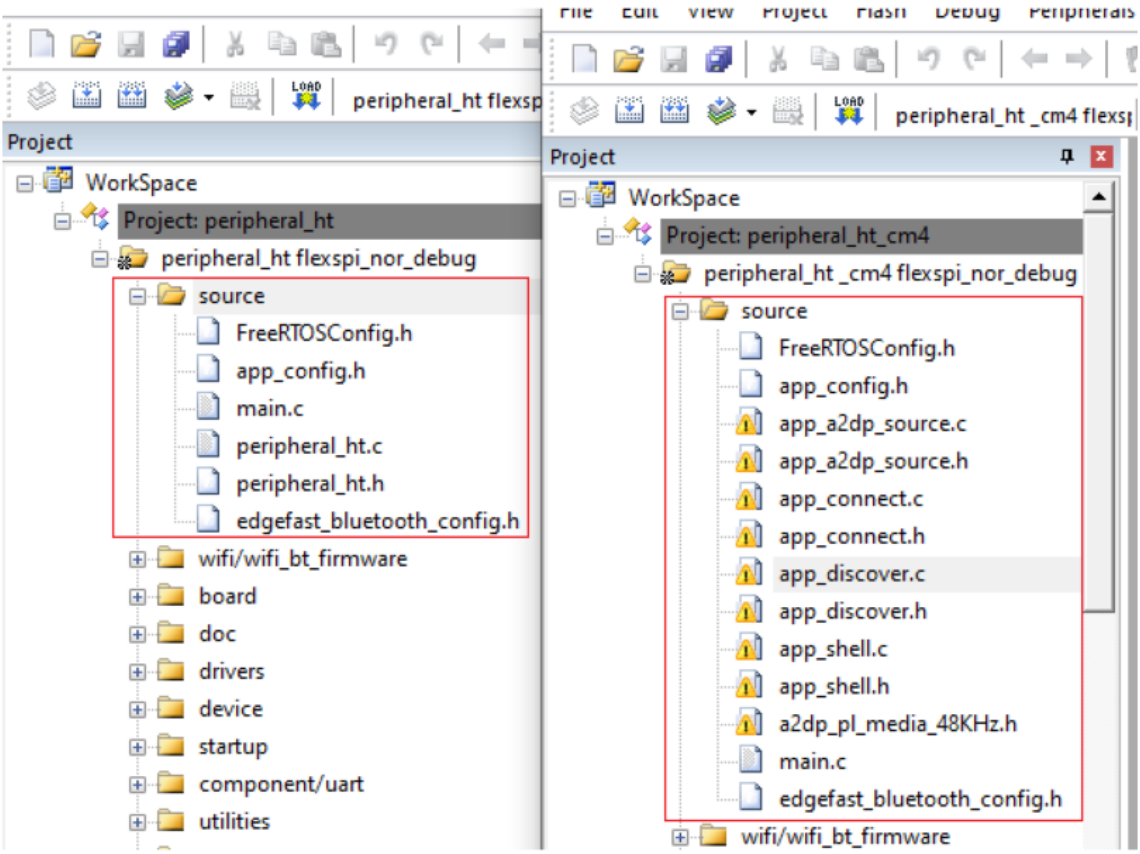


.

Parent topic:*MDK*

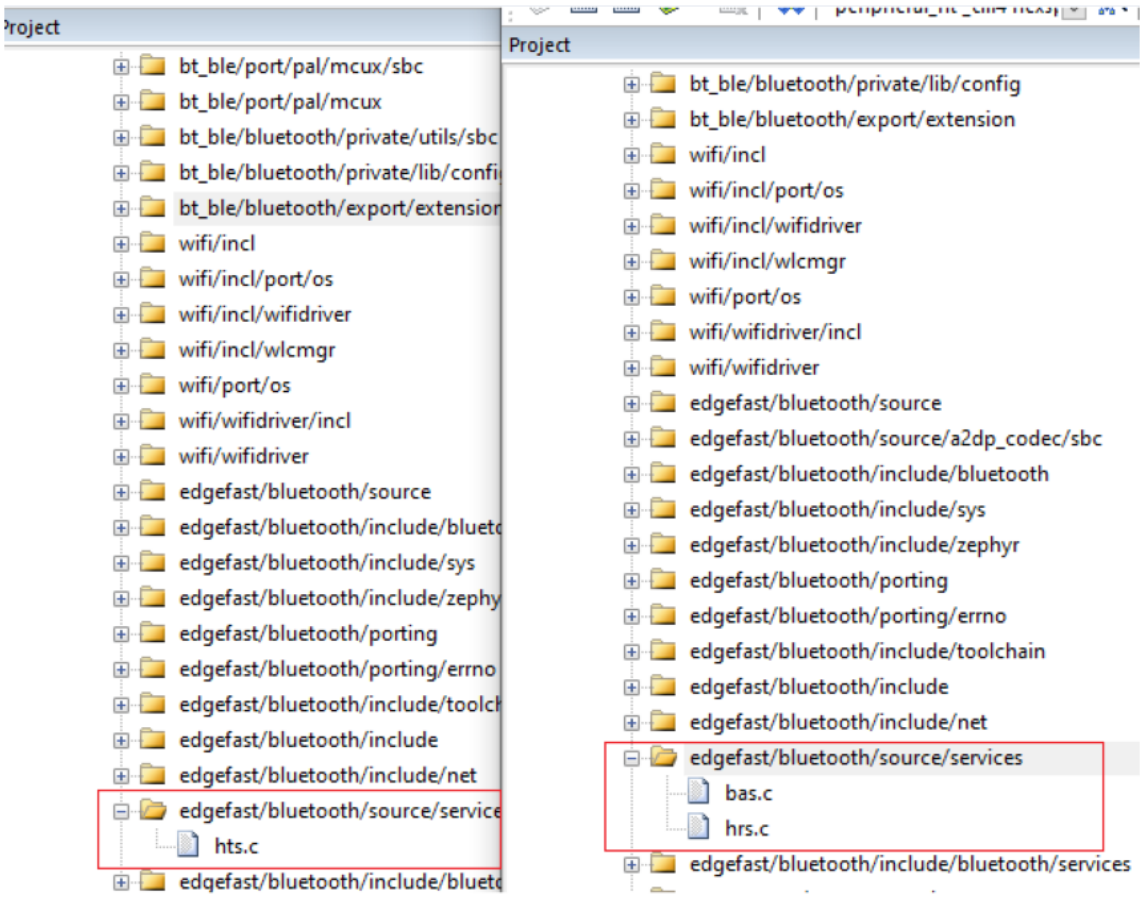
Rearrange project files

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.

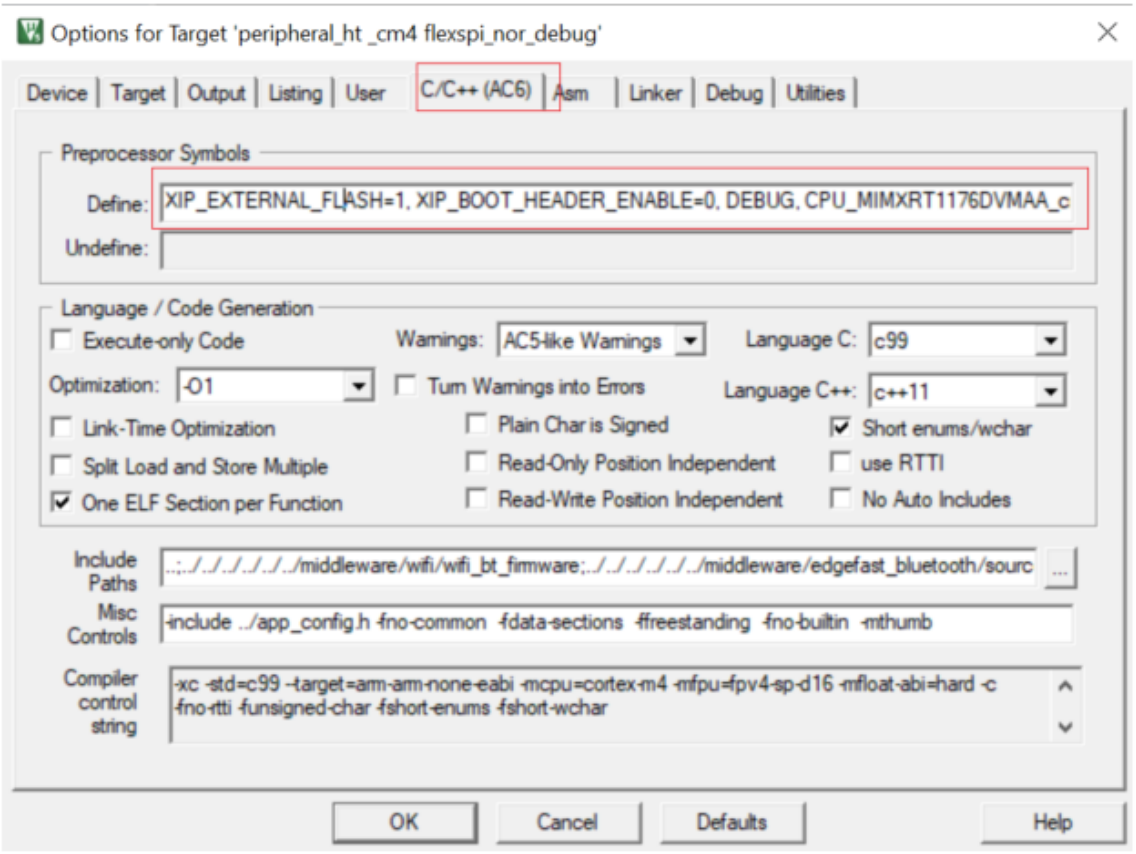


.

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