



MCUXpresso SDK Documentation

Release 25.09.00-pvw1



NXP
Jul 17, 2025



Table of contents

1	FRDM-MCXL255	3
1.1	Overview	3
1.2	Getting Started with MCUXpresso SDK Package	3
1.2.1	Getting Started with MCUXpresso SDK Package	3
1.3	Getting Started with MCUXpresso SDK GitHub	55
1.3.1	Getting Started with MCUXpresso SDK Repository	55
1.4	Release Notes	68
1.4.1	MCUXpresso SDK Release Notes	68
1.5	ChangeLog	71
1.5.1	MCUXpresso SDK Changelog	71
1.6	Currently supported drivers	129
1.7	Driver API Reference Manual	130
1.8	Middleware Documentation	130
1.8.1	Multicore	130
1.8.2	FreeRTOS	130
2	MCXL255	131
2.1	AON_LPADC: 12-bit single-ended SAR Analog-to-Digital Converter Driver	131
2.2	Clock Driver	144
2.3	CRC: Cyclic Redundancy Check Driver	153
2.4	eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	156
2.5	eDMA core Driver	186
2.6	eDMA soc Driver	193
2.7	FGPIO Driver	194
2.8	GLIKEY	194
2.9	GLIKEY	194
2.10	GPIO: General-Purpose Input/Output Driver	198
2.11	GPIO Driver	201
2.12	INPUTMUX: Input Multiplexing Driver	207
2.13	IPMQ Driver	236
2.14	KPP: KeyPad Port Driver	244
2.15	Common Driver	246
2.16	LPCMP: Low Power Analog Comparator Driver	258
2.17	LPI2C: Low Power Inter-Integrated Circuit Driver	267
2.18	LPI2C Master Driver	269
2.19	LPI2C Master DMA Driver	283
2.20	LPI2C Slave Driver	285
2.21	LPSPi: Low Power Serial Peripheral Interface	296
2.22	LPSPi Peripheral driver	296
2.23	LPSPi eDMA Driver	317
2.24	LPTMR: Low-Power Timer	324
2.25	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	330
2.26	LPUART Driver	330
2.27	LPUART eDMA Driver	348
2.28	MU: Messaging Unit	351
2.29	OSTIMER: OS Event Timer Driver	360

2.30	PORT: Port Control and Interrupts	363
2.31	POWER Driver	371
2.32	QTMR: Quad Timer Driver	393
2.33	Reset Driver	403
2.34	ROMAPI Driver	405
2.35	SLCD: Segment LCD Driver	408
2.36	Trdc_soc	420
2.37	UTICK: MictoTick Timer Driver	421
3	Middleware	423
3.1	MultiCore	423
3.1.1	Multicore SDK	423
4	RTOS	519
4.1	FreeRTOS	519
4.1.1	FreeRTOS kernel	519
4.1.2	FreeRTOS drivers	525
4.1.3	backoffalgorithm	525
4.1.4	corehttp	528
4.1.5	corejson	530
4.1.6	coremqtt	533
4.1.7	coremqtt-agent	536
4.1.8	corepkcs11	540
4.1.9	freertos-plus-tcp	543

This documentation contains information specific to the frdm-mcxl255 board.

Chapter 1

FRDM-MCXL255

1.1 Overview

The FRDM-MCXL255 board is a design and evaluation platform based on the NXP MCXL255 microcontroller (MCU). The MCXL255 MCU is a low-power microcontroller for industrial and consumer Internet of Things (IoT) applications. It has one Arm Cortex-M33 core running at speeds of up to 96 MHz and one Arm Cortex-M0+ core up to 10 MHz. This series features our Adaptive Dynamic Voltage Control (ADVC) for optimized power consumption at low frequency operation. Compared to traditional low-power MCUs, a dedicated ultra-low-power (ULP) Sense Domain allows operation of low-power peripherals while keeping the main core in Deep Power Mode. This avoids event triggering and keeps data acquisition to extremely low power levels.

The board is compatible with Arduino boards (Arduino UNO R3 and Arduino A4/A5), MikroE click boards, and Pmod boards. It can be used with IAR Embedded Workbench development tool.

For debugging the MCXA276 MCU, the FRDM-MCXL255 board uses an onboard (OB) debug probe, MCU-Link lite OB, which is based on another NXP MCU: LPC55S16

MCU device and part on board is shown below:

- Device: MCXL255
- PartNumber: MCXL255VDF

1.2 Getting Started with MCUXpresso SDK Package

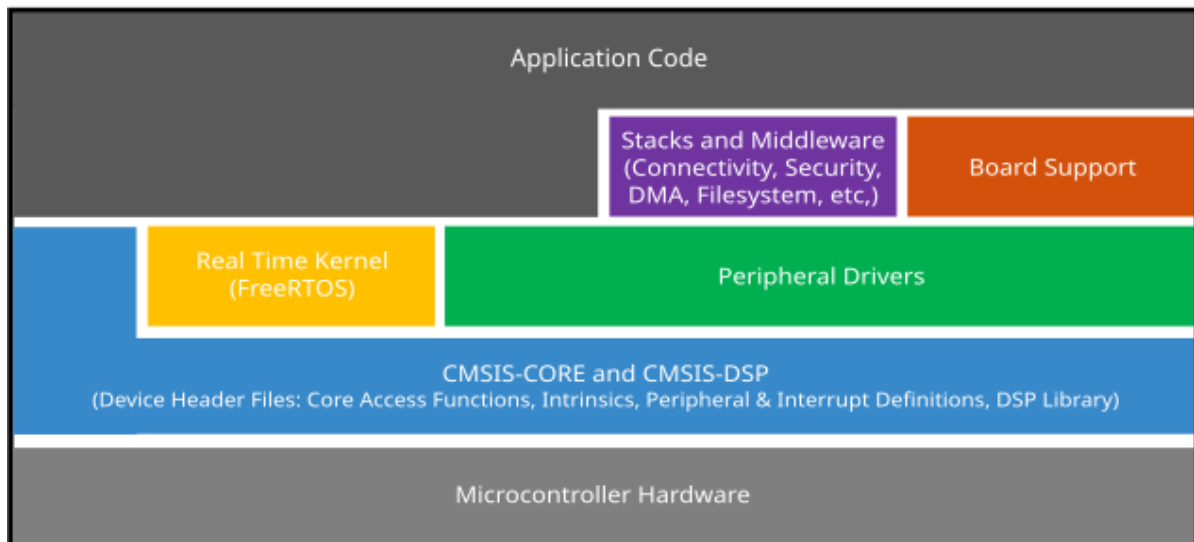
1.2.1 Getting Started with MCUXpresso SDK Package

Overview

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

For supported toolchain versions, see *MCUXpresso SDK Release Notes* (document MCUXSDKRN).

For more details about MCUXpresso SDK, see [MCUXpresso Software Development Kit \(SDK\)](#).



MCUXpresso SDK board support package folders

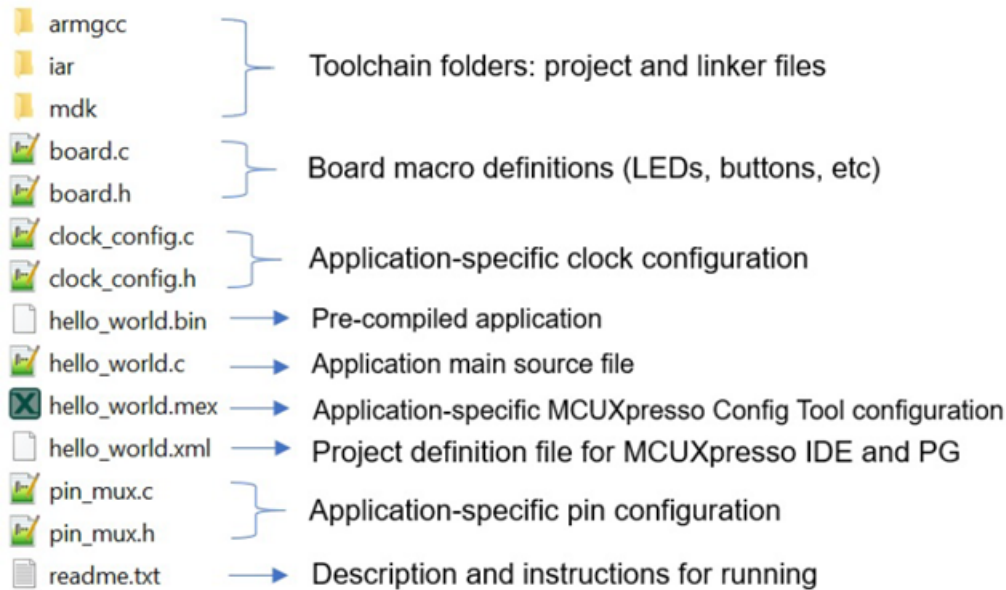
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has its own folder (an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

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

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

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

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



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

Locating example application source files When opening an example application in any of the supported IDEs, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- `devices/<device_name>`: The device's CMSIS header file, MCUXpresso SDK feature file, and a few other files
- `devices/<device_name>/cmsis_drivers`: All the CMSIS drivers for your specific MCU
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code, including vector table definitions
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications
- `devices/<device_name>/project`: Project template used in CMSIS PACK new project creation

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

Run a demo using MCUXpresso IDE

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

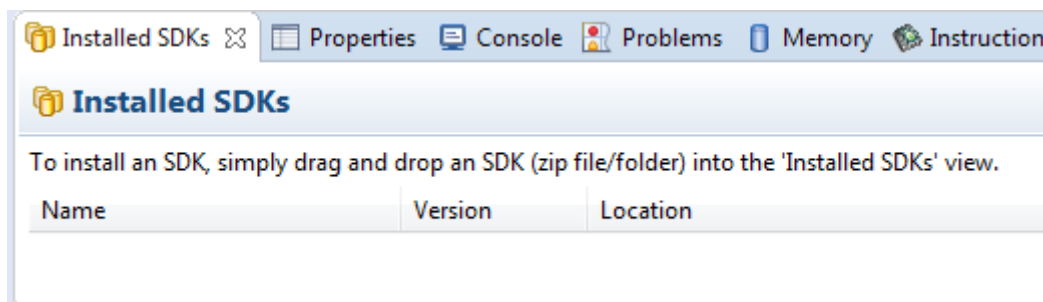
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The `hello_world` demo application targeted for the hardware platform is

used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

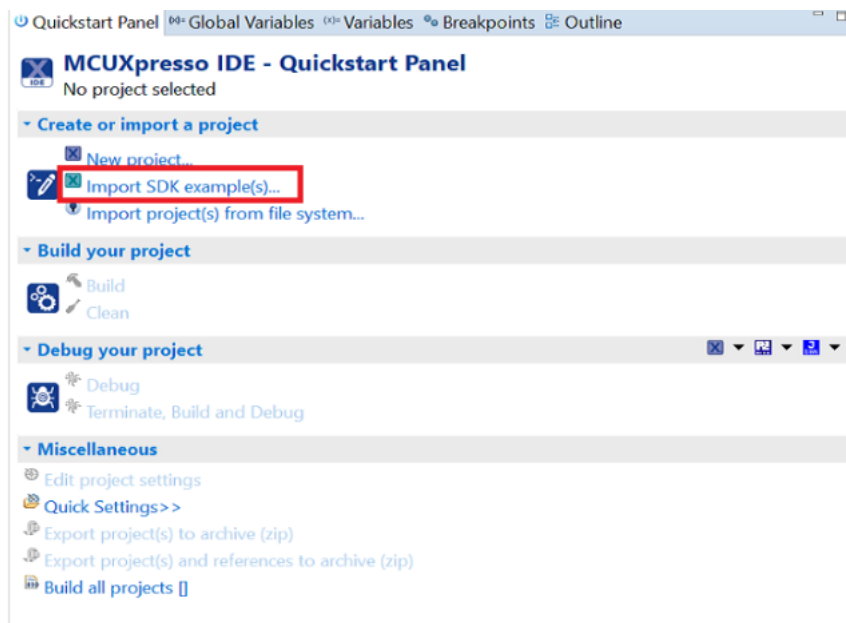
Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace.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

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

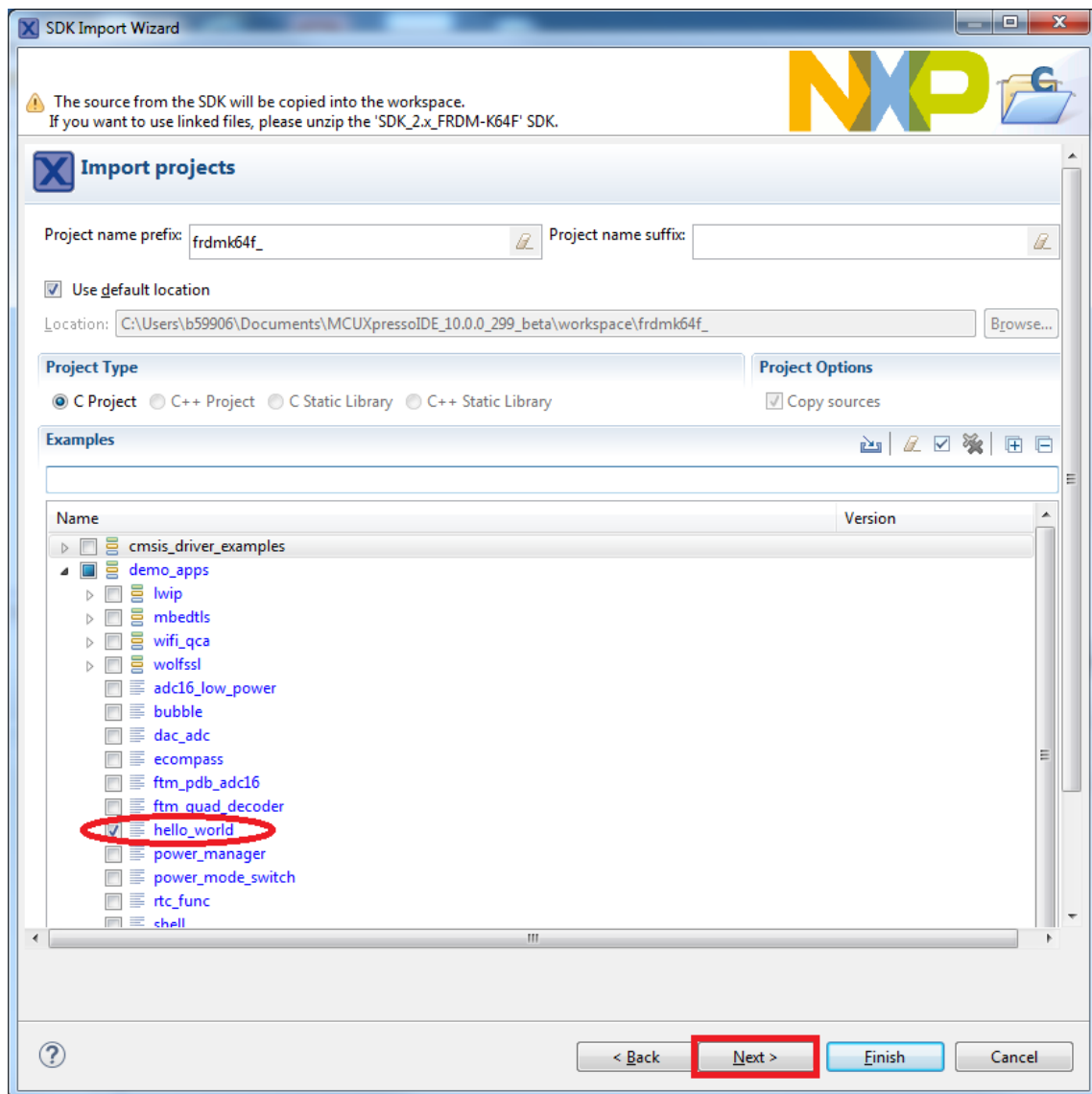
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



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



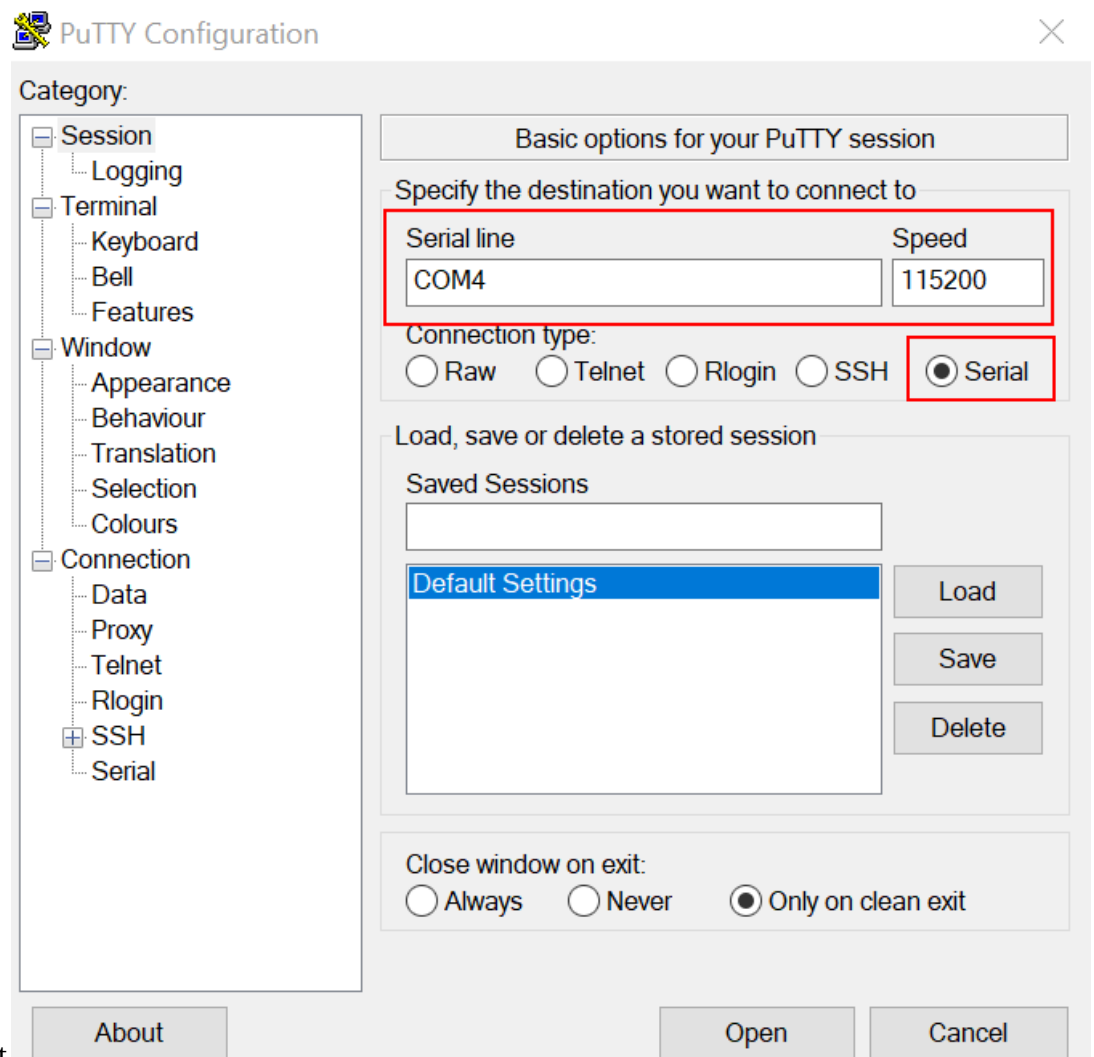
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

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

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

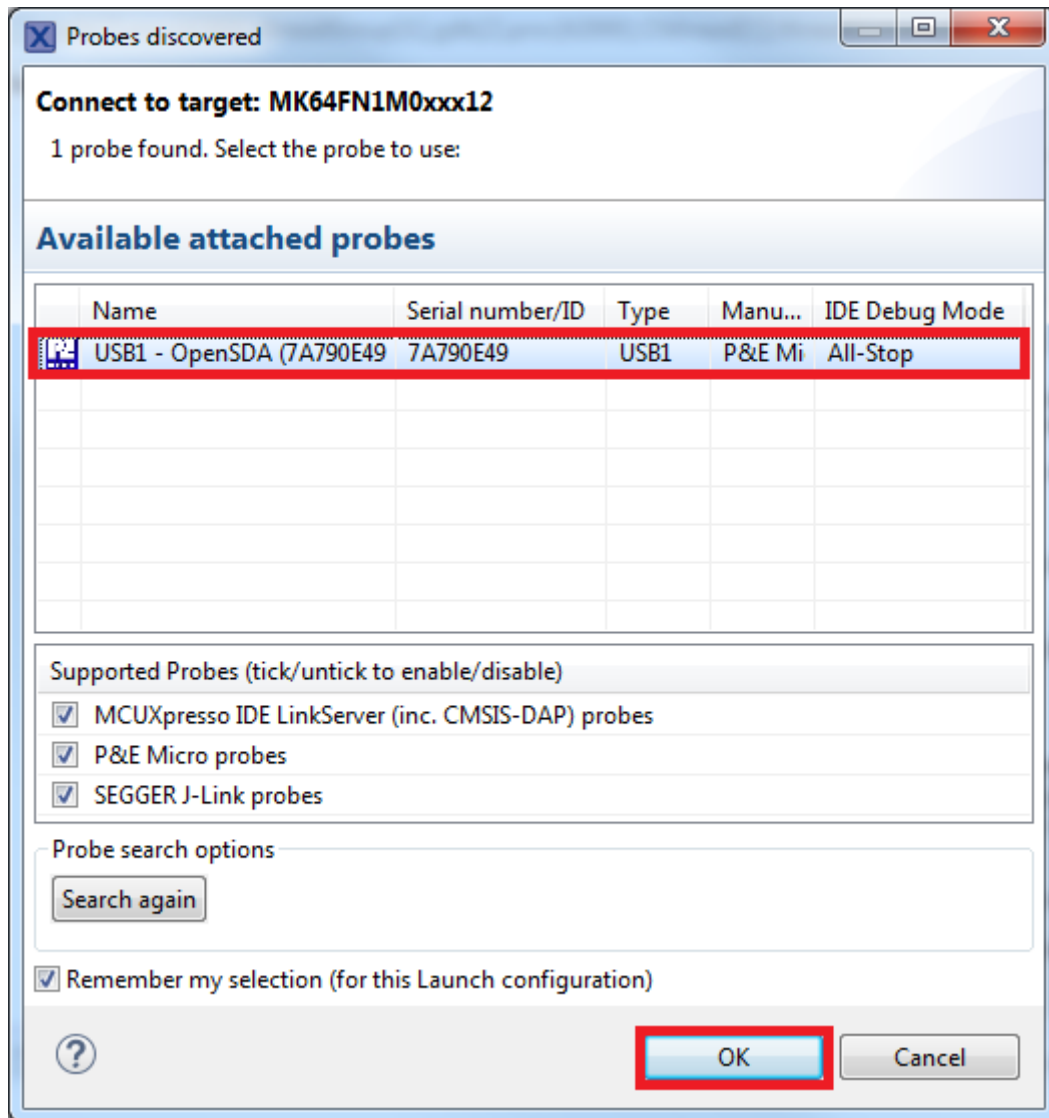
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM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

3. 8 data bits



4. 1 stop bit

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



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

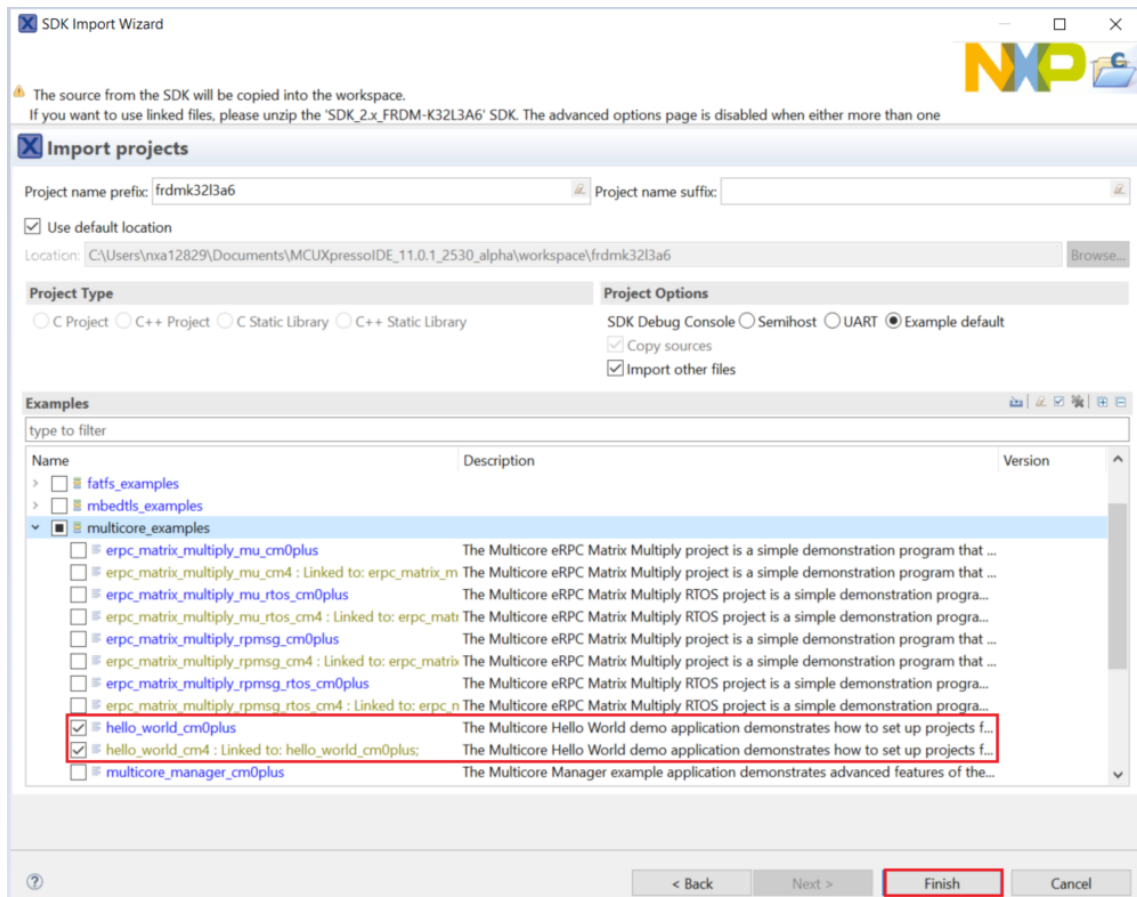


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

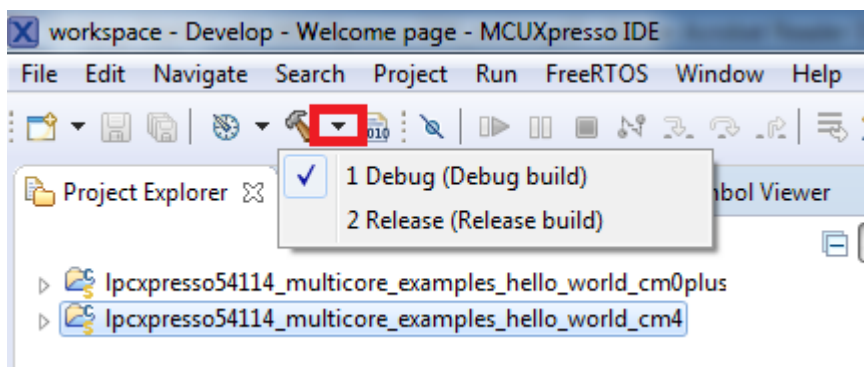


Build a multicore example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world example application targeted for the LPCXpresso54114 hardware platform is used as an example.

1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in **Build an example application**. When the SDK zip package for LPCXpresso54114 is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **LPCxx** folder and select **LPC54114J256**. Then, select **lpcxpresso54114** and click **Next**.
2. Expand the multicore_examples/hello_world folder and select **cm4**. The cm0plus counterpart project is automatically imported with the cm4 project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.

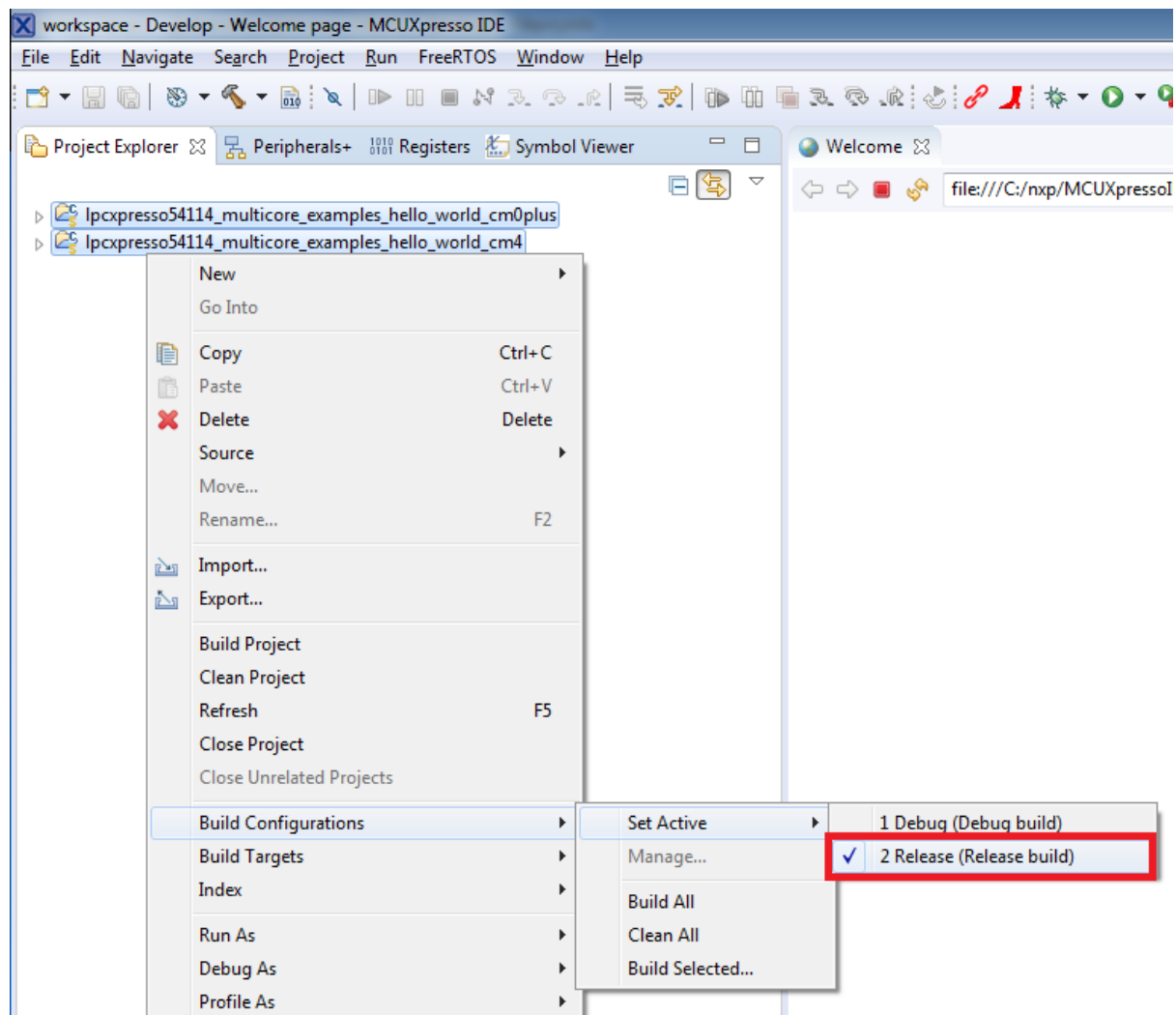


3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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

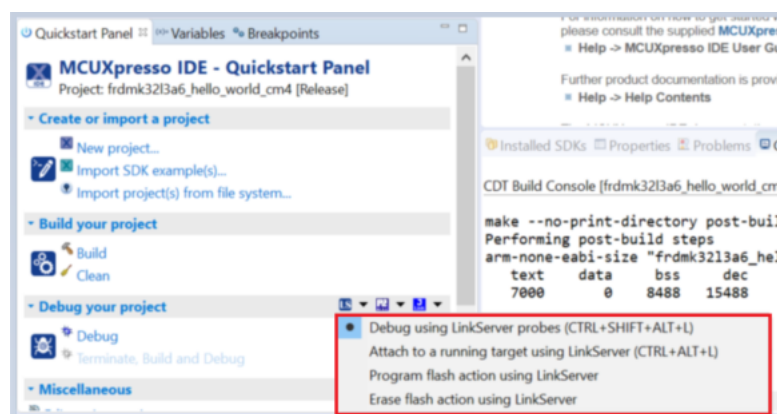


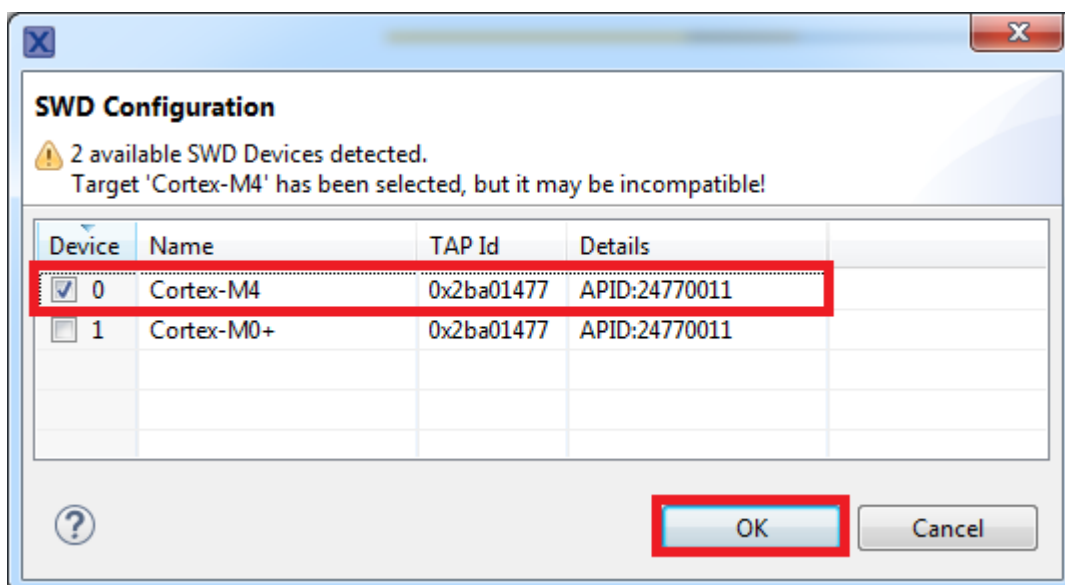
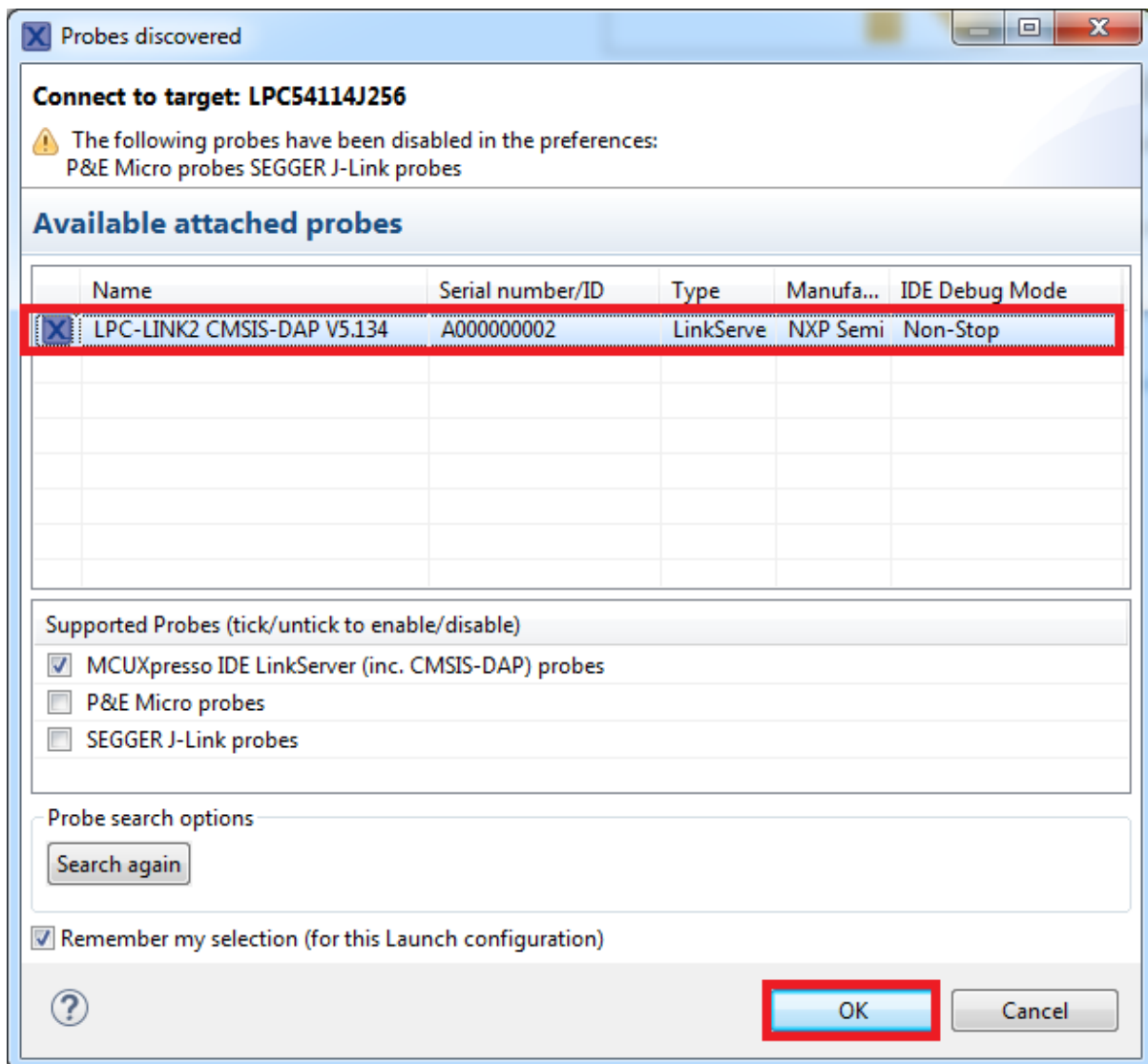
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

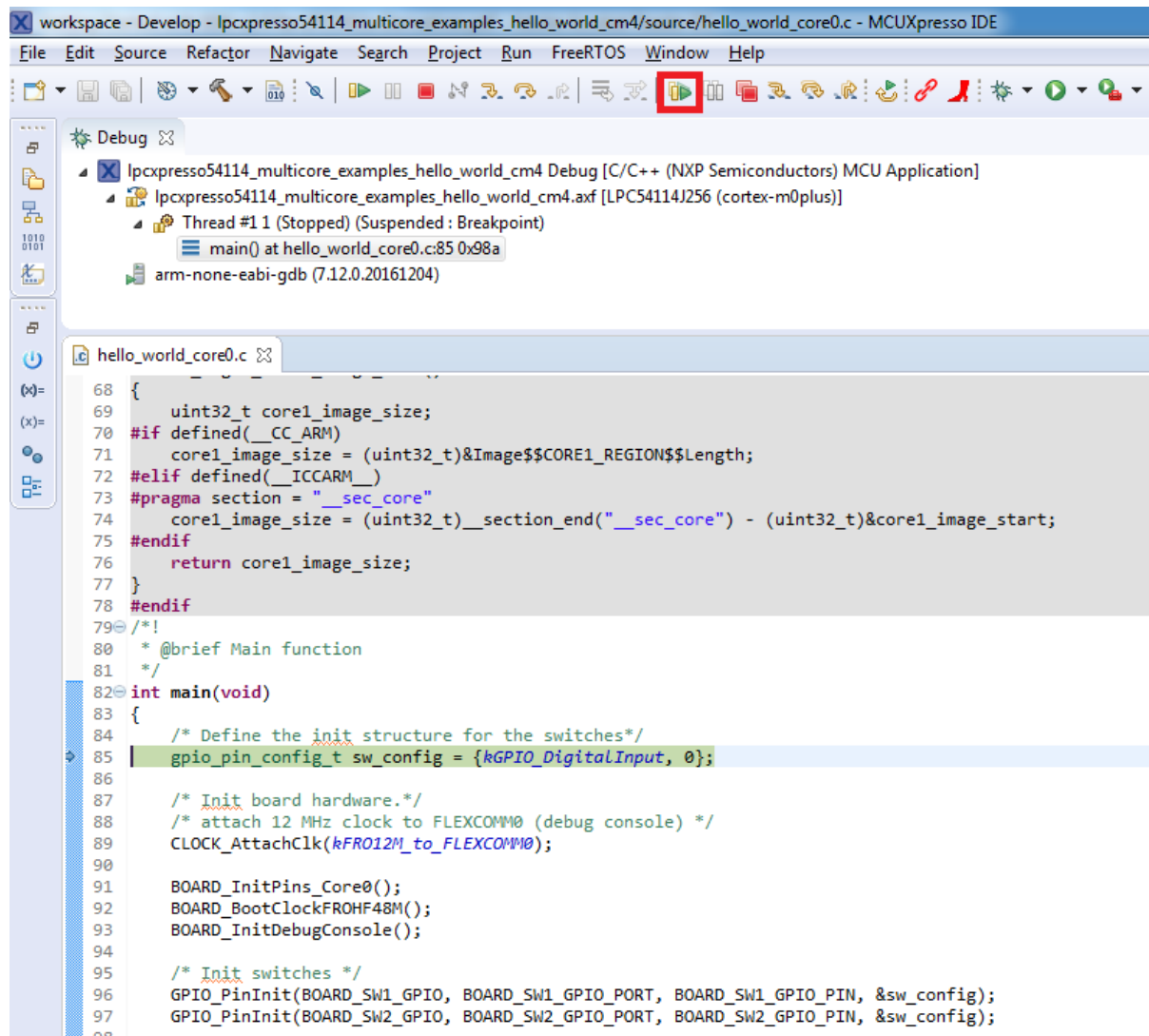
Note: When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build Configurations -> Set Active -> Release**. This alternate navigation using the menu item is **Project -> Build Configuration -> Set Active -> Release**. After switching to the **Release** build configuration, the build of the multicore example can be started by triggering the primary core application (cm4) build.



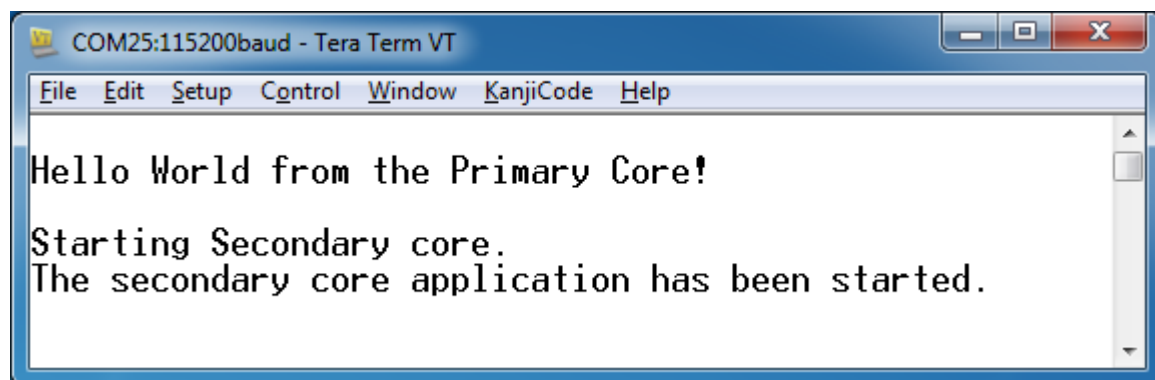
Run a multicore example application The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run an example application**. These steps are common for both single-core applications and the primary side of dual-core applications, ensuring both sides of the multicore application are properly loaded and started. However, there is one additional dialogue that is specific to multicore examples which requires selecting the target core. See the following figures as reference.





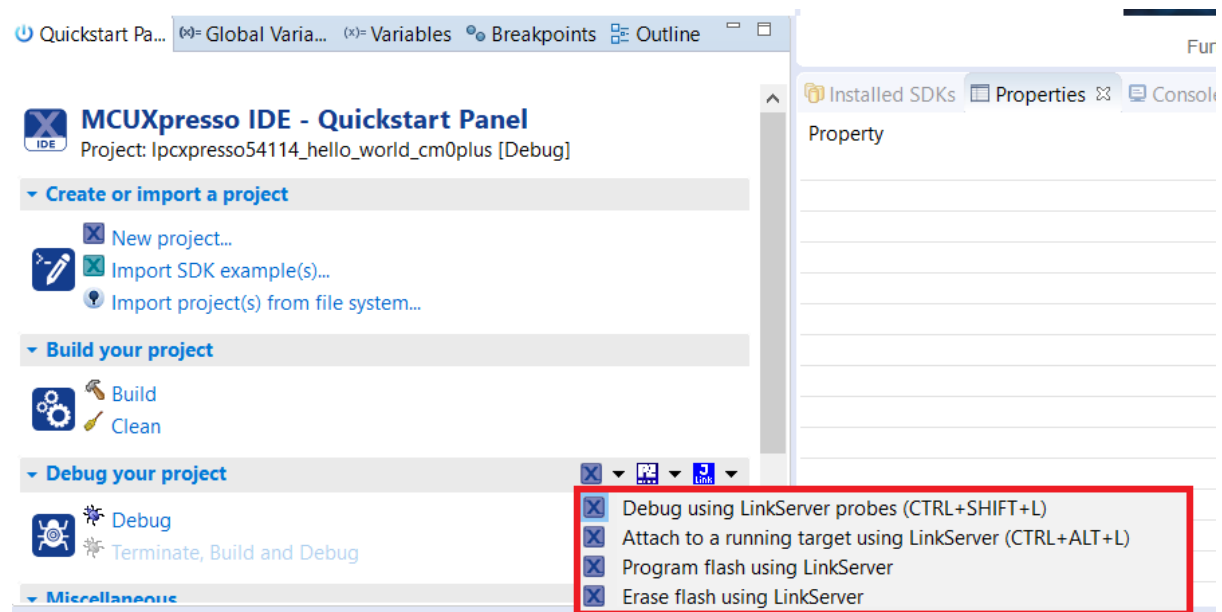


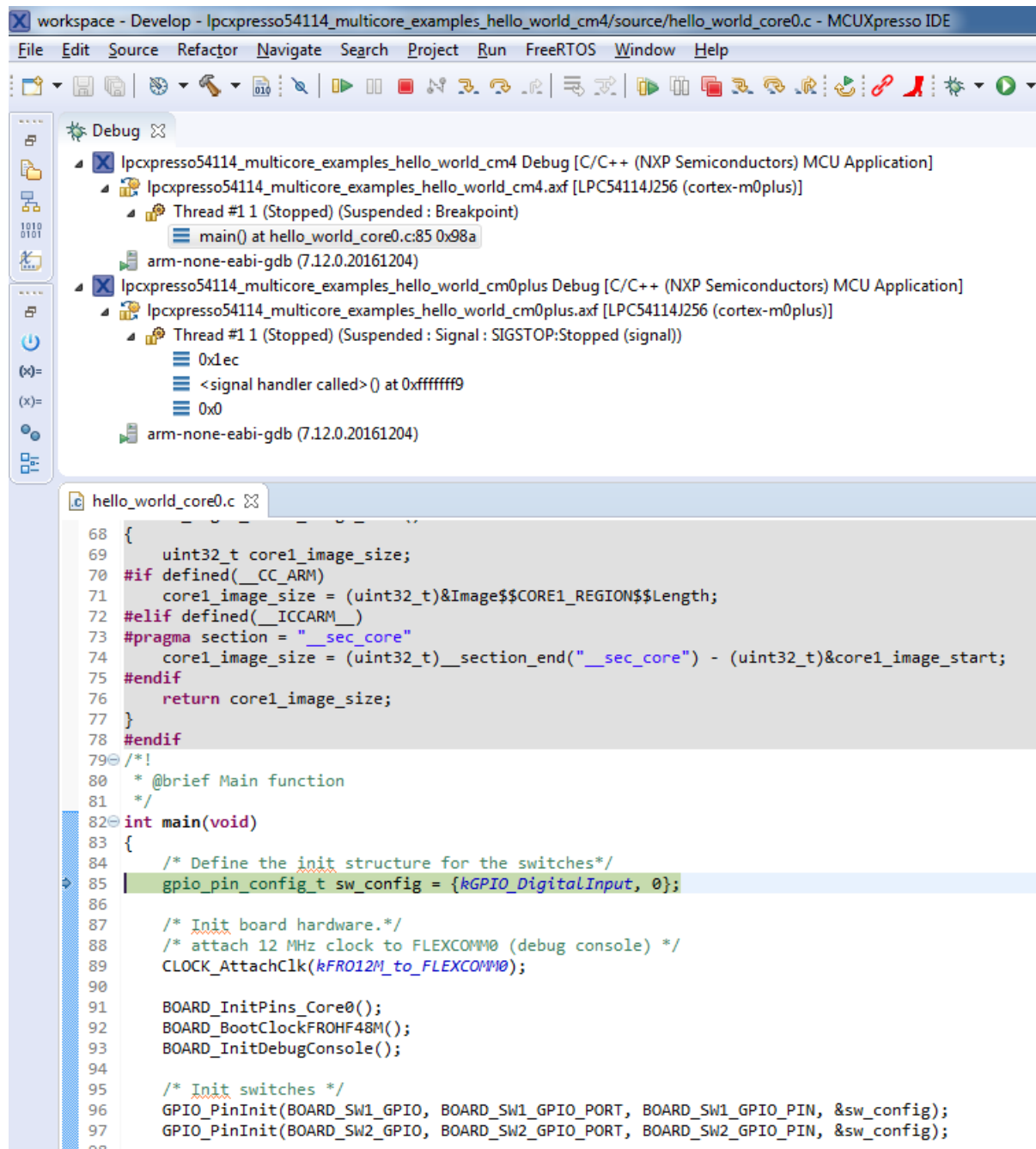
After clicking the “Resume All Debug sessions” button, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



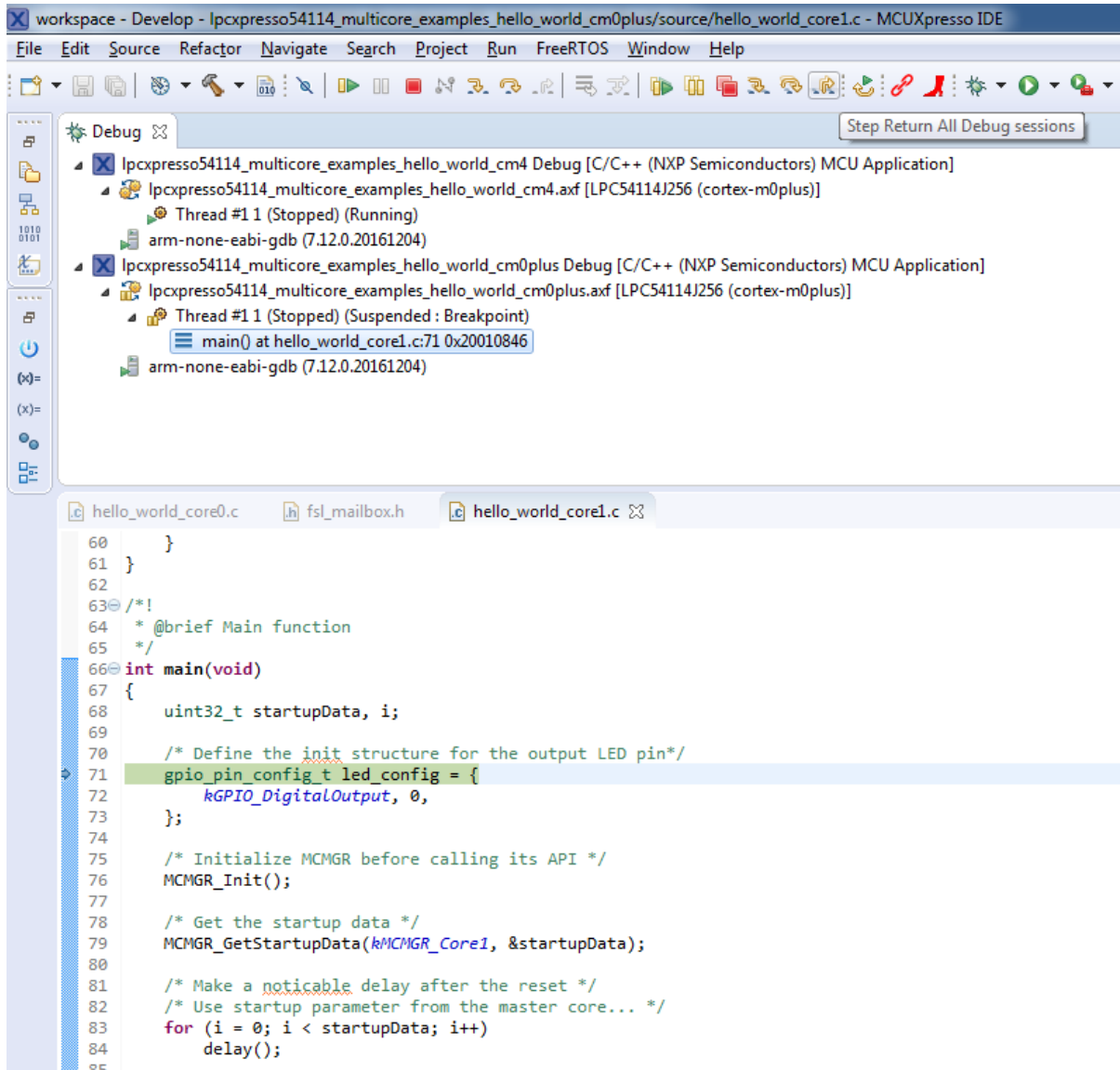
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps also for the auxiliary core application. Highlight the lpcxpresso54114_multicore_examples_hello_world_cm0plus project (multicore slave project) in the Project Explorer. On the Quickstart Panel, click “Debug ‘lpcxpresso54114_multicore_examples_hello_world_cm0plus’ [Debug]” to launch the second debug

session.

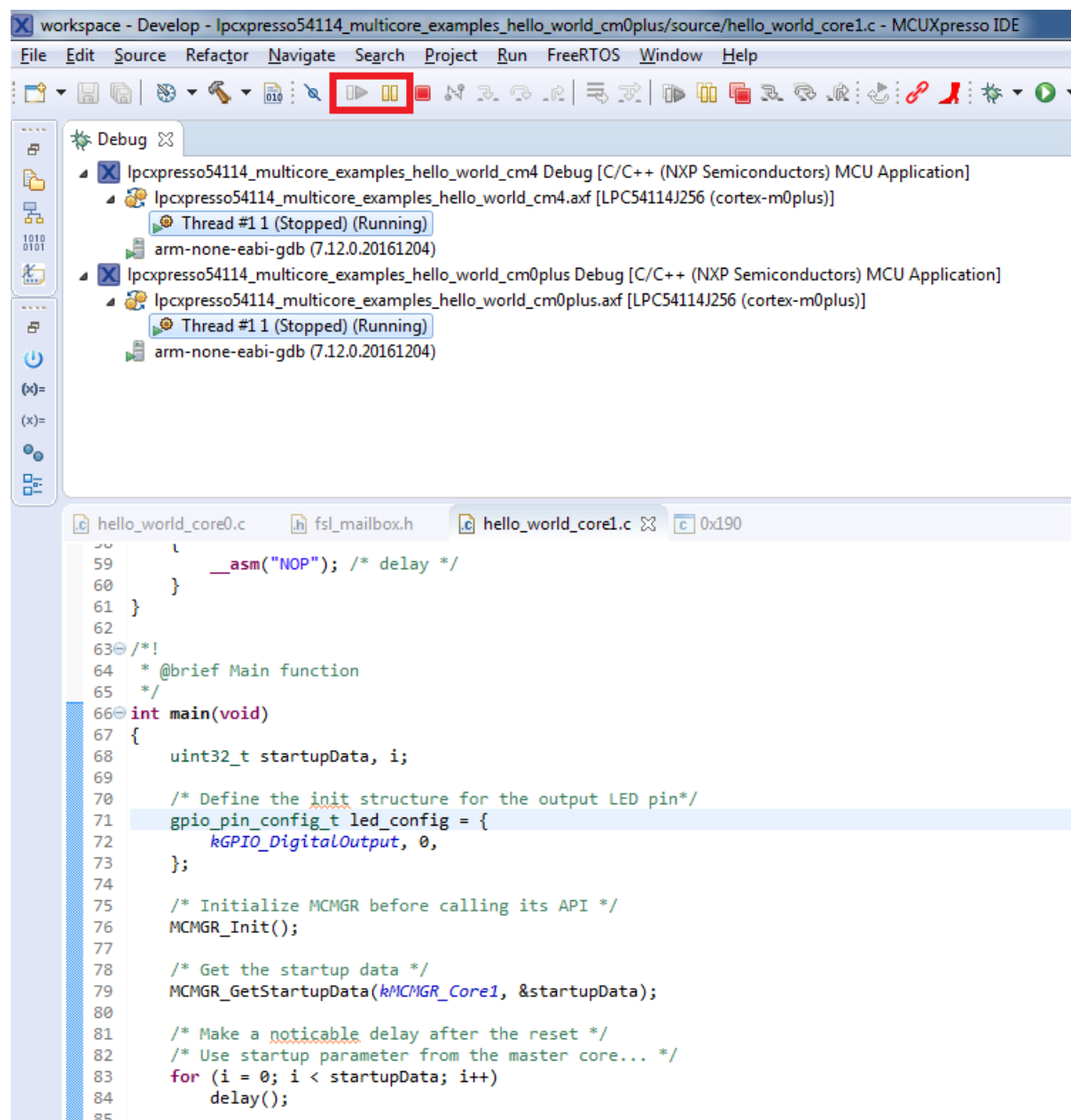


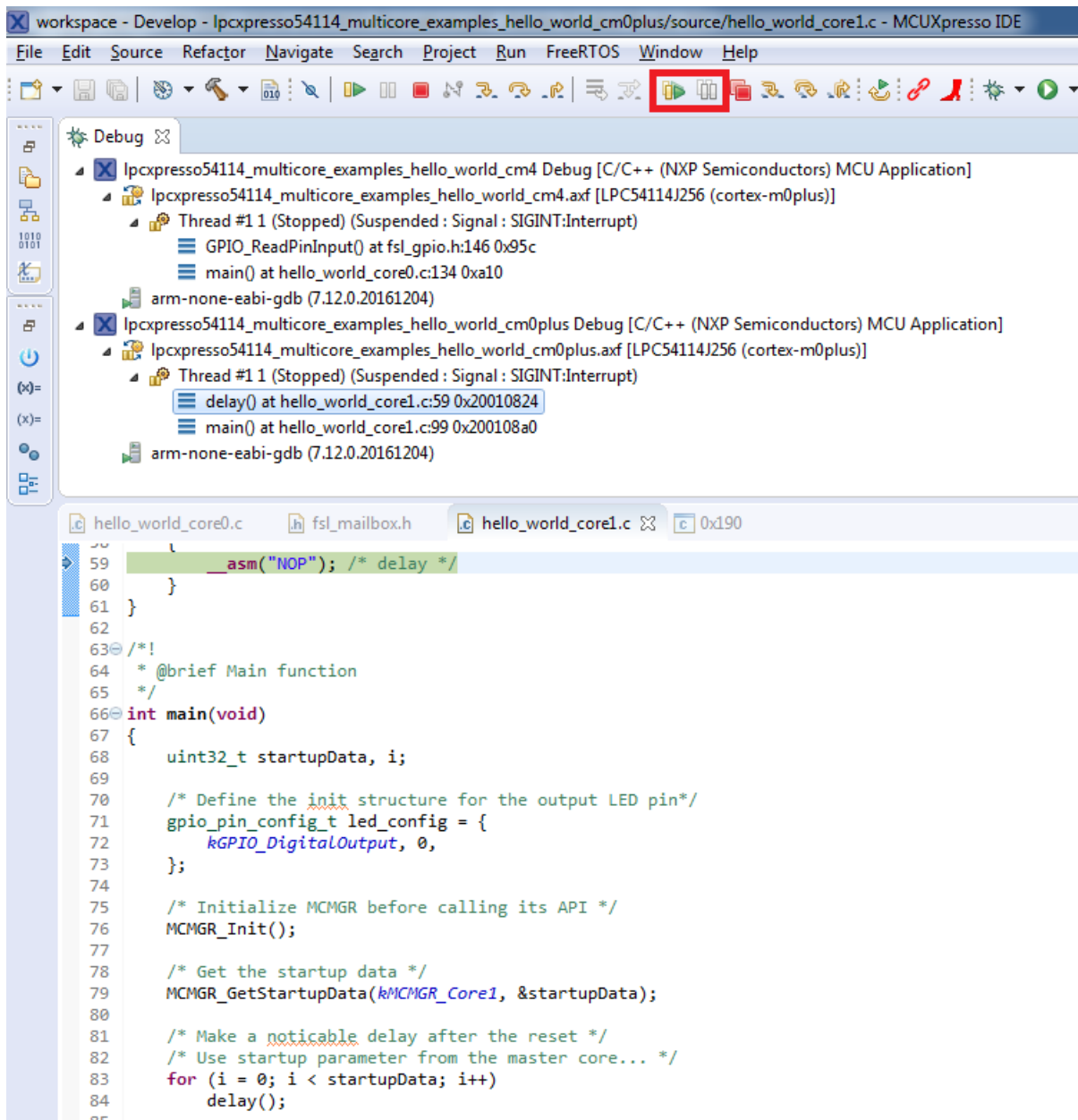


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello_world multicore application then starts running. The primary core application starts the auxiliary core application during runtime, and the auxiliary core application stops at the beginning of the main() function. The debug session of the auxiliary core application is highlighted. After clicking the “Resume” button, it is applied to the auxiliary core debug session. Therefore, the auxiliary core application continues its execution.



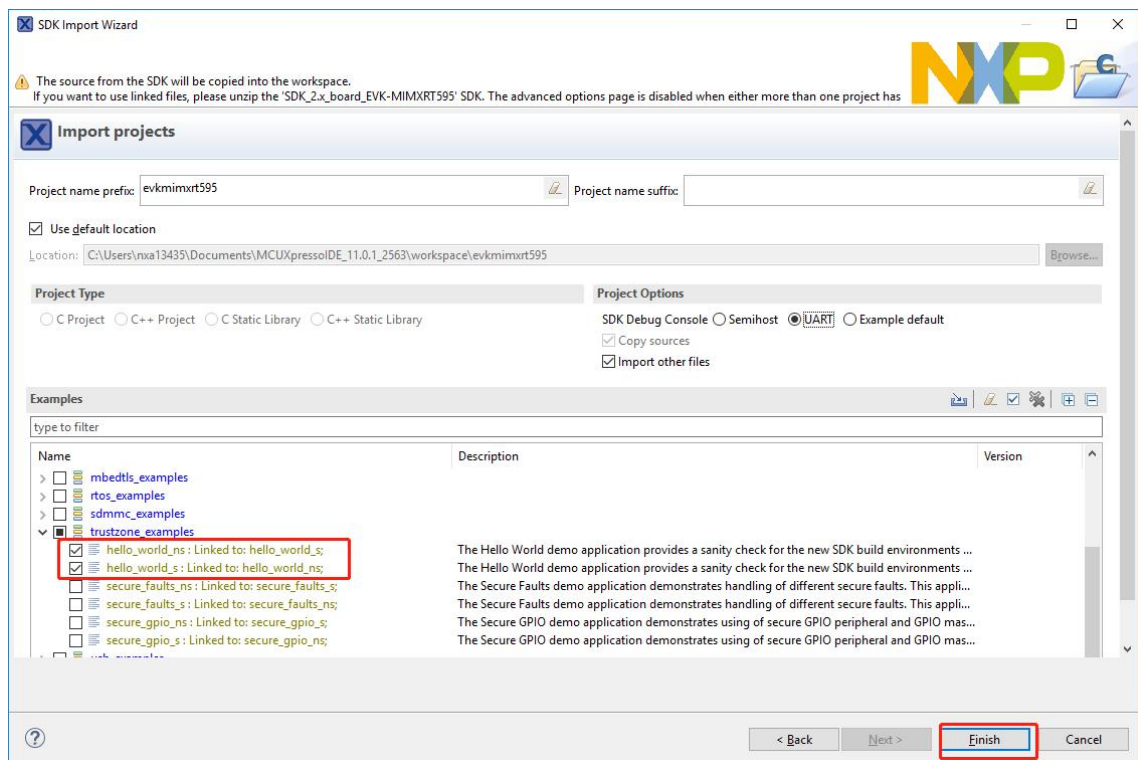
At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both the cores. This is done either by selecting both opened debug sessions (multiple selections) and clicking the “Suspend” / “Resume” control button,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



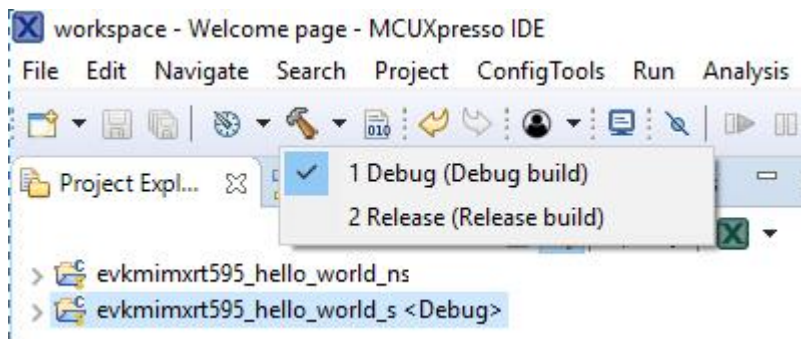


Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the `hello_world` example application targeted for the MIMXRT595-EVK hardware platform is used as an example, though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

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

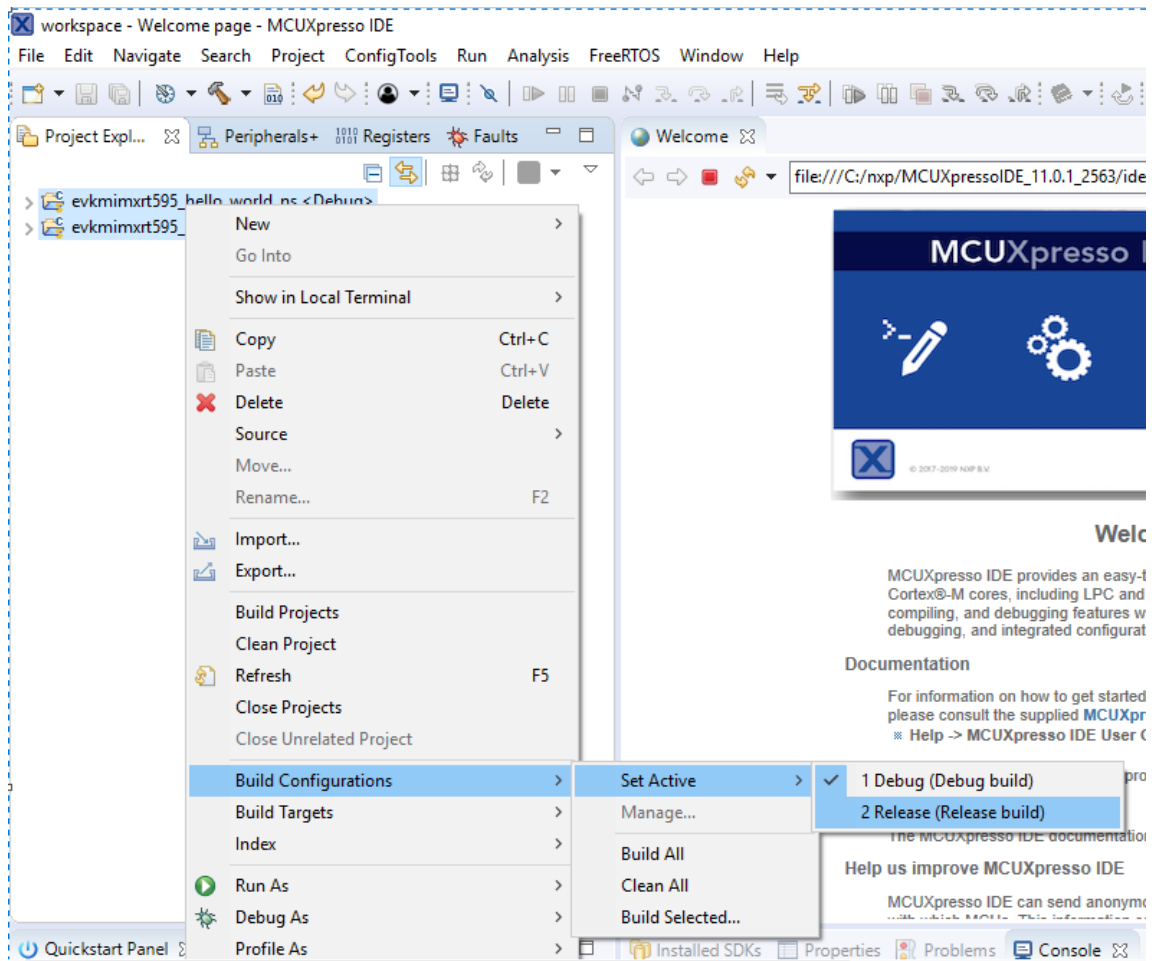


- Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward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



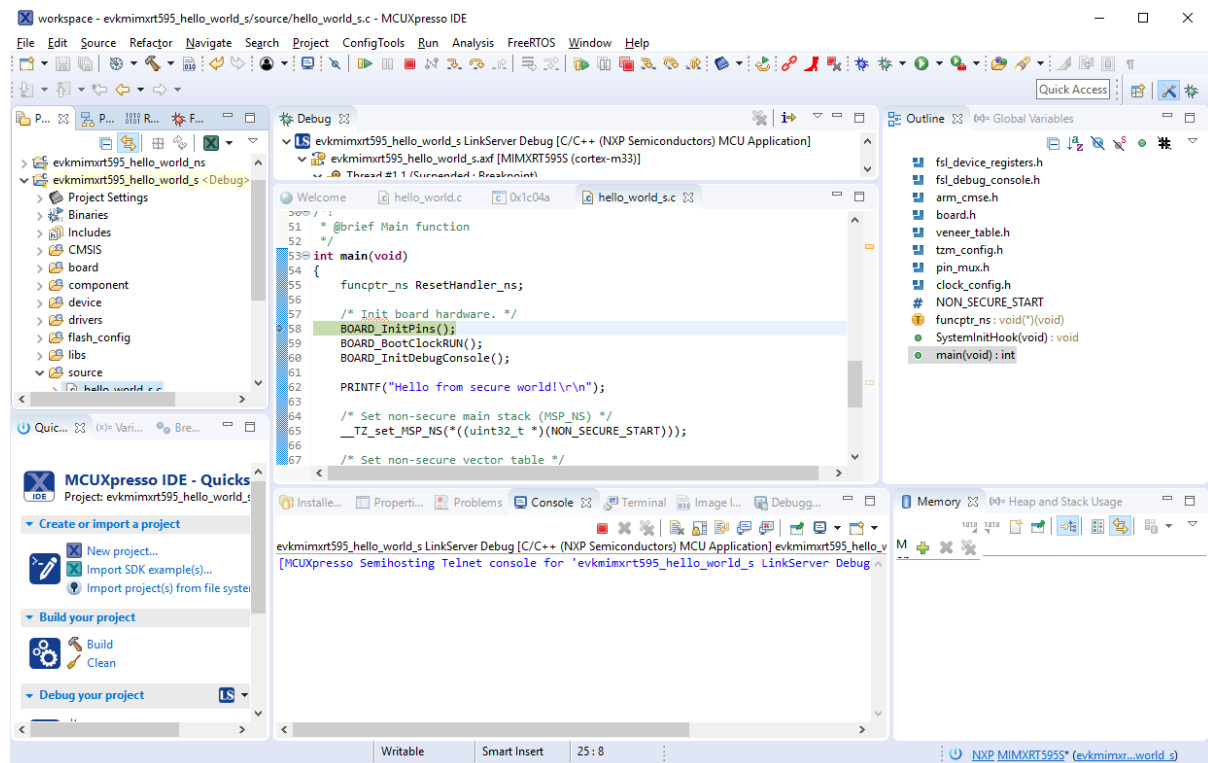
The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when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

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



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

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



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

Run a demo application using IAR

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

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

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

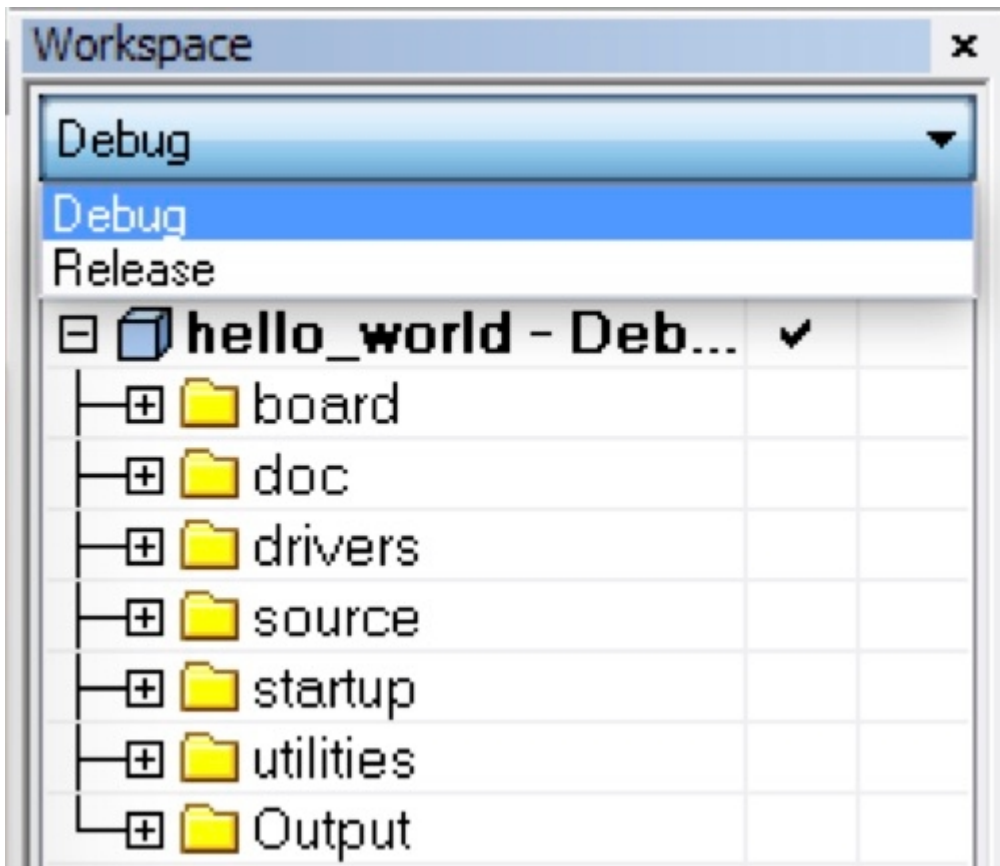
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

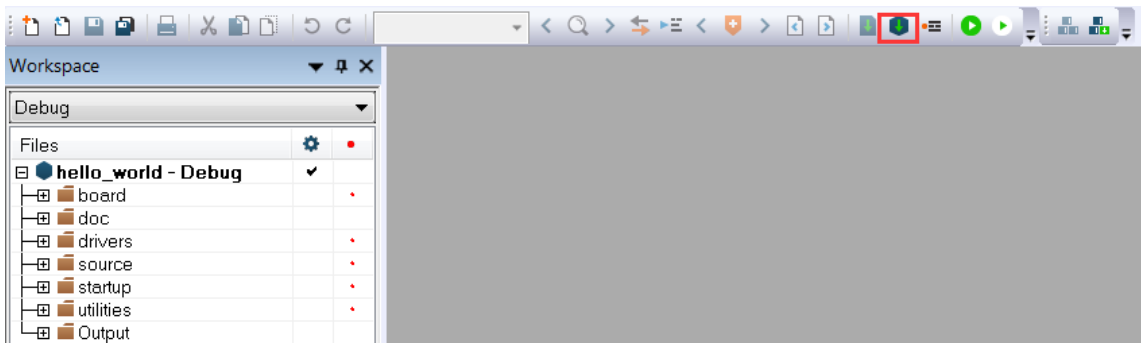
Other example applications may have additional folders in their path.

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

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



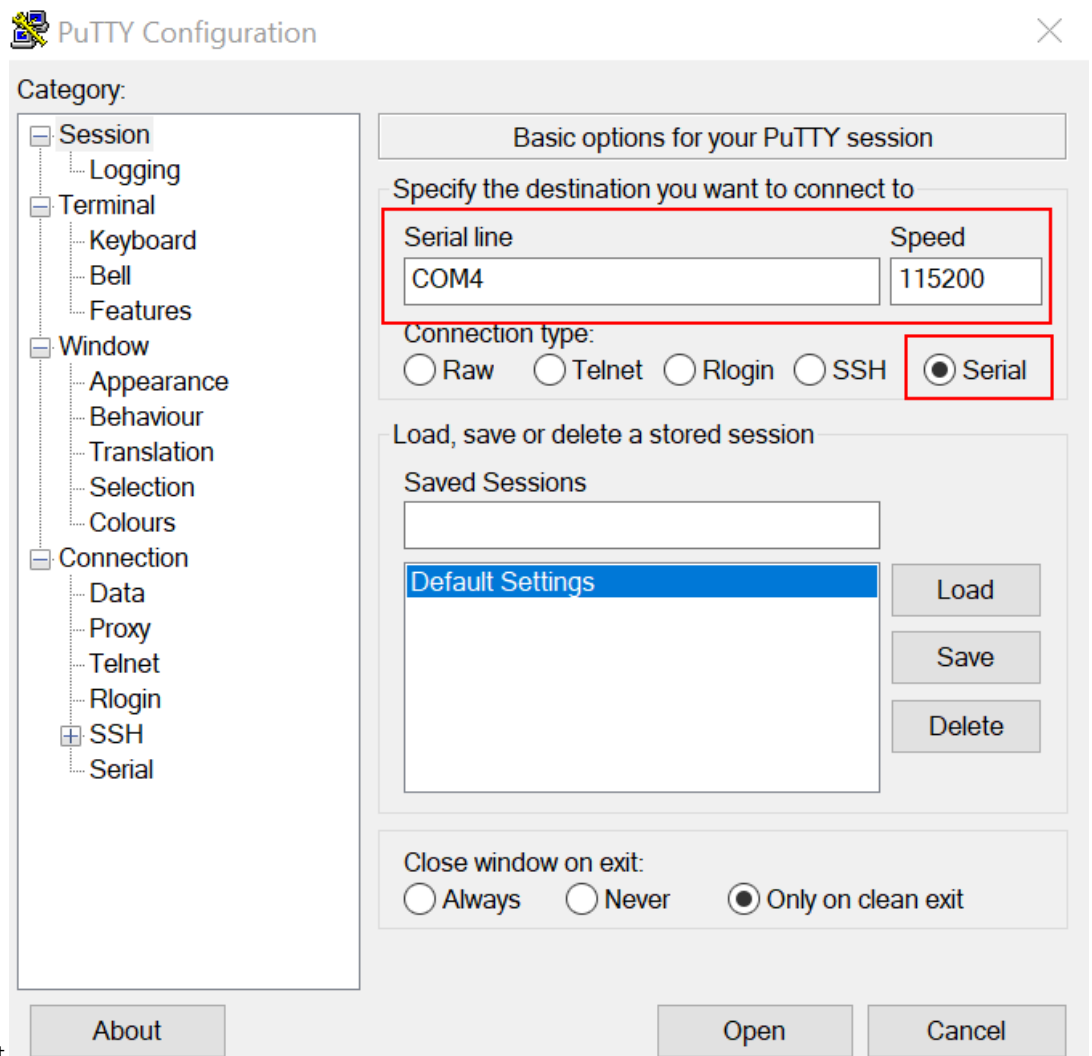
3. To build the demo application, click **Make**, highlighted in red in following figure.



4. The build completes without errors.

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

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

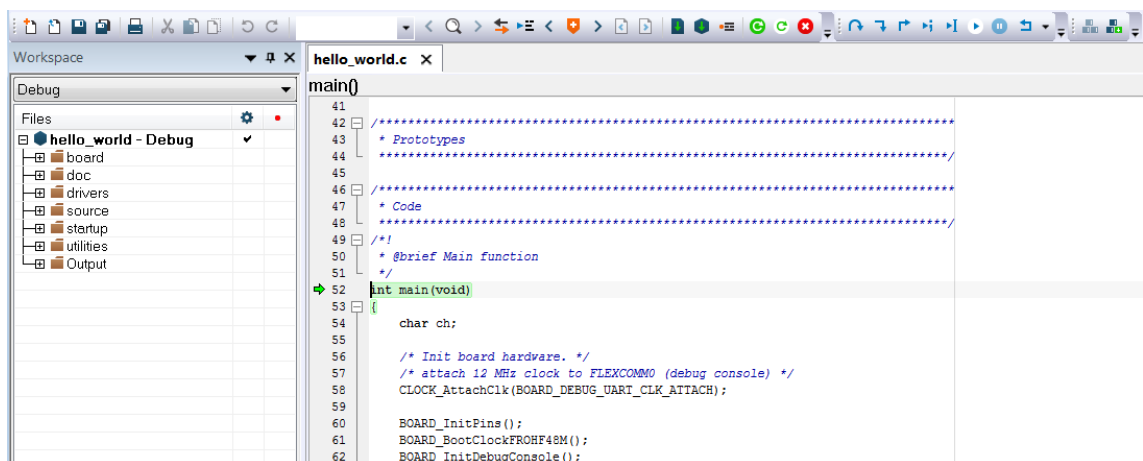


4. 1 stop bit

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



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



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

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



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

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

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

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

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

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

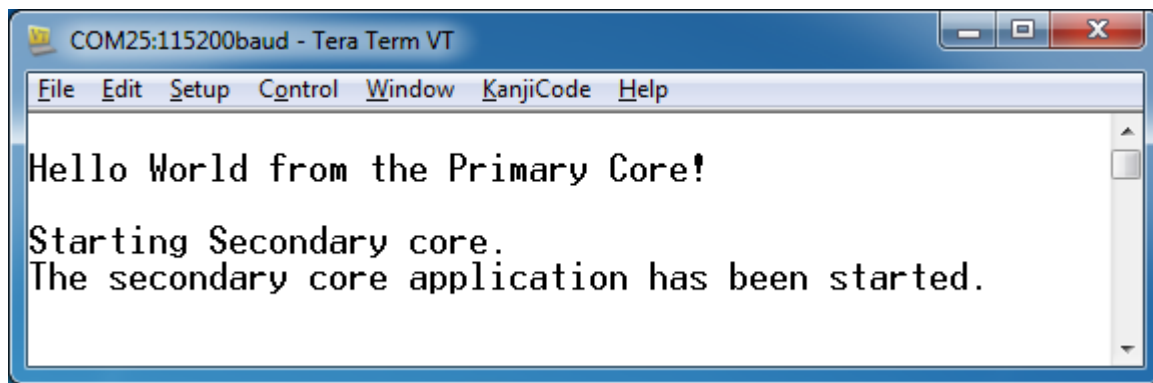
Run a multicore example application The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an example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

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

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



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



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

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

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

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

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

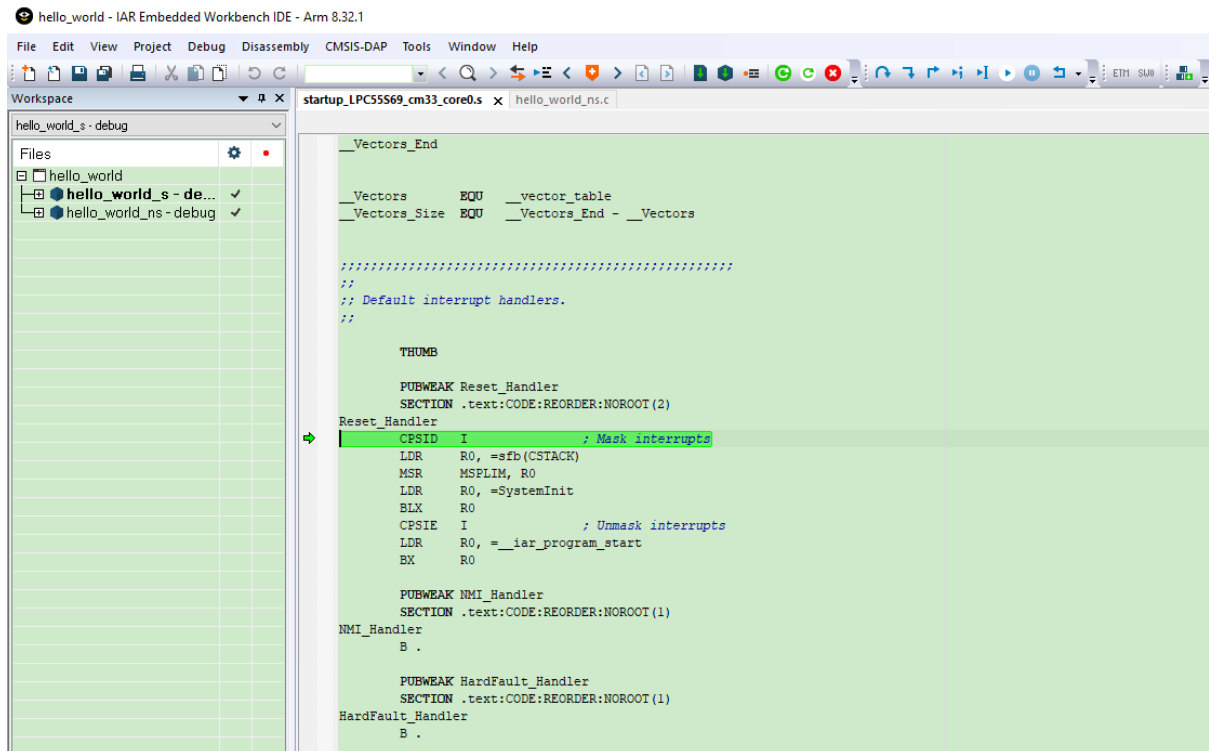
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_  
↪ns.eww
```

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

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

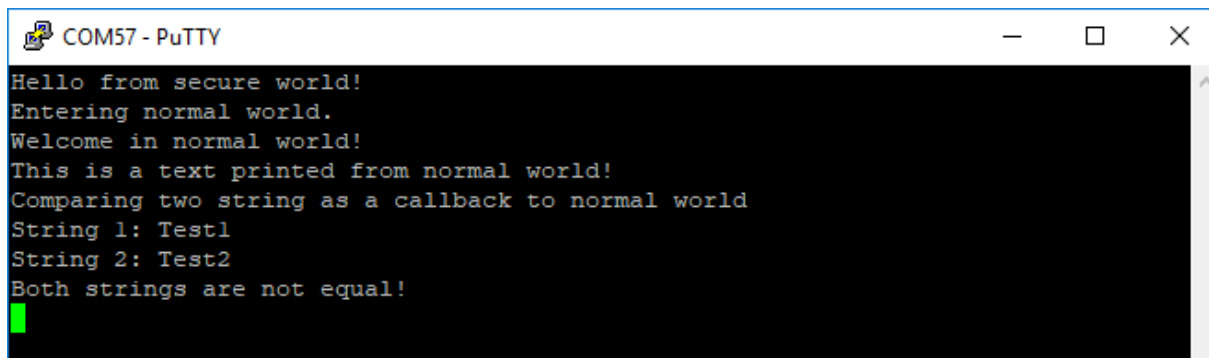
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because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

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

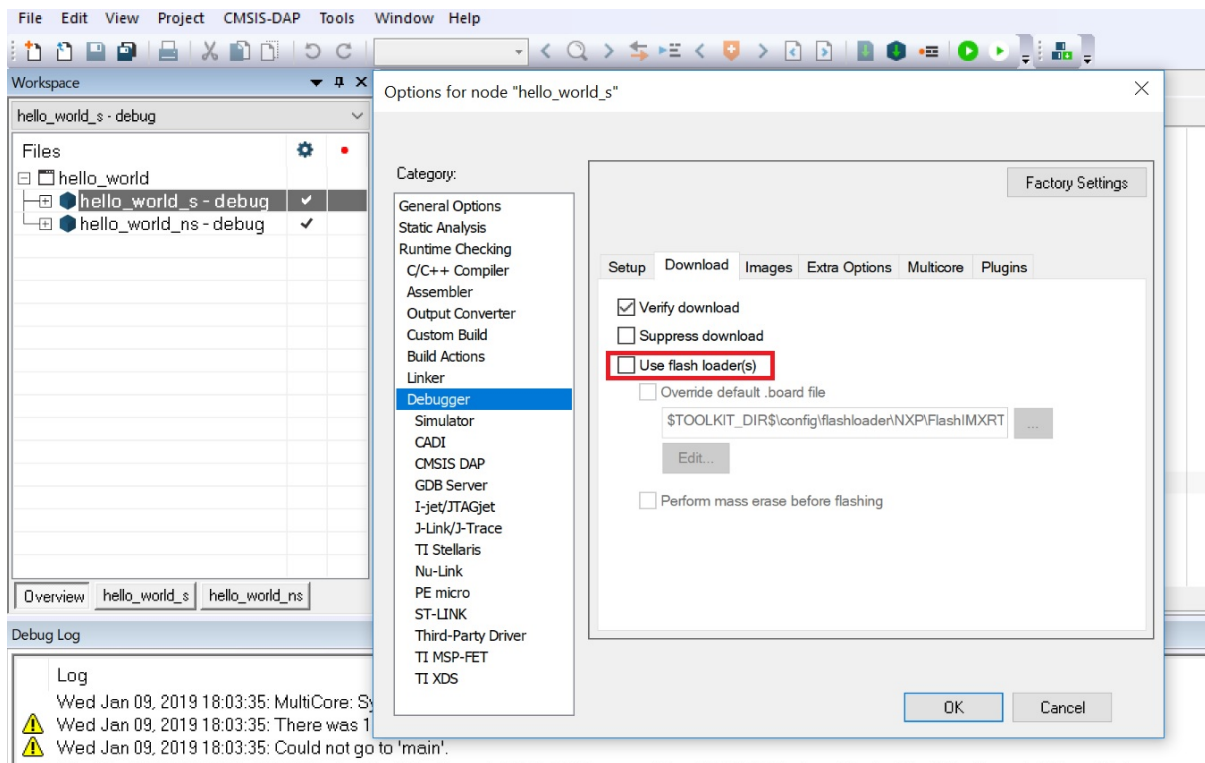


Run the code by clicking **Go** to start the application.

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



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

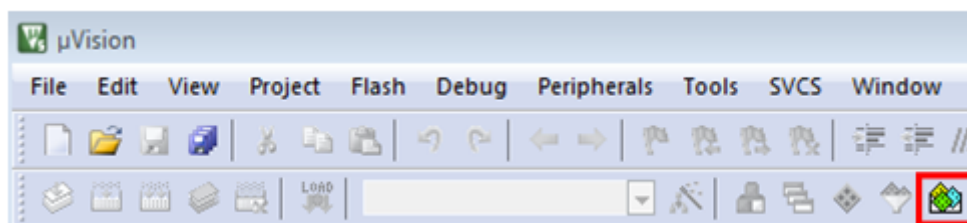


Run a demo using Keil MDK/μVision

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

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

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



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

Build an example application

1. Open the desired example application workspace in:

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

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

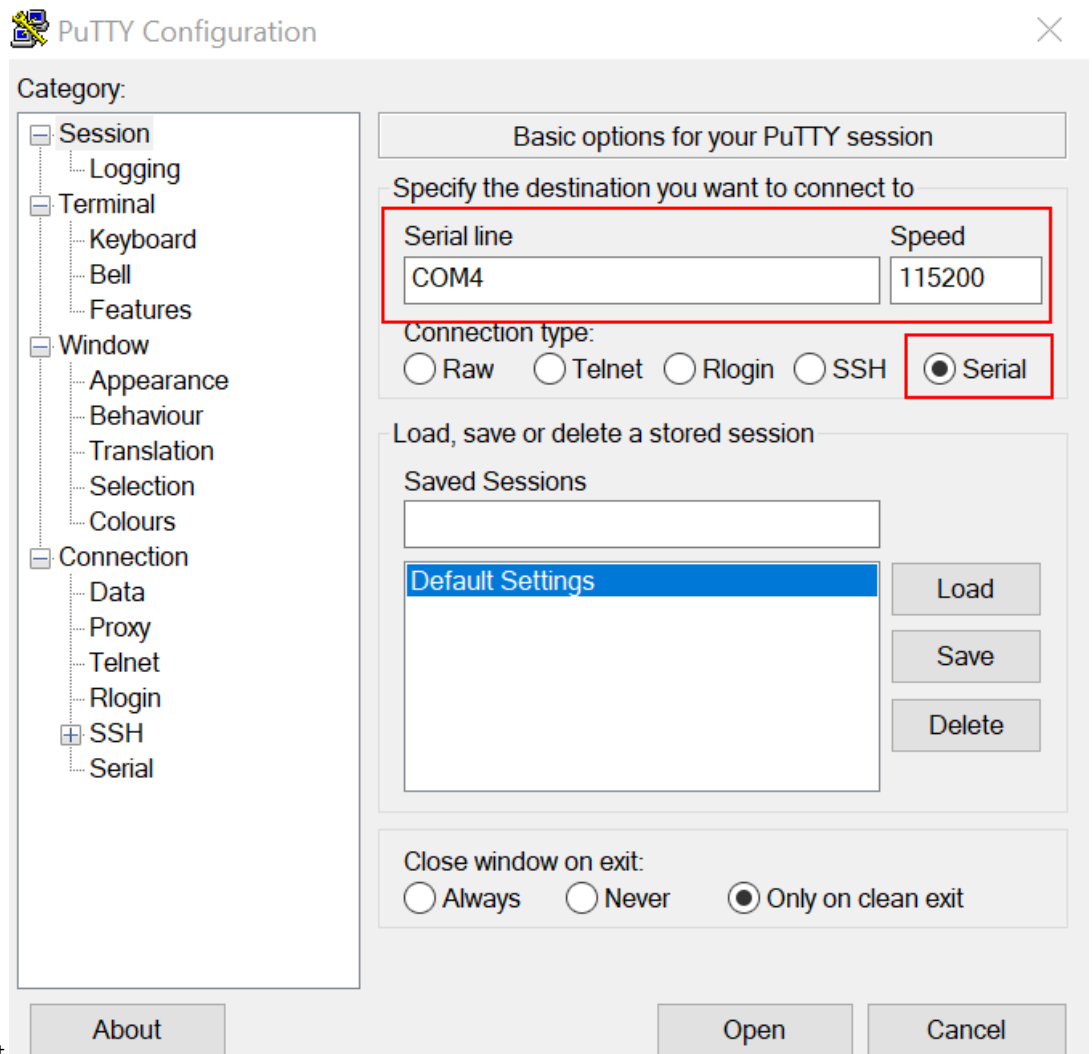
- To build the demo project, select **Rebuild**, highlighted in red.



- The build completes without errors.

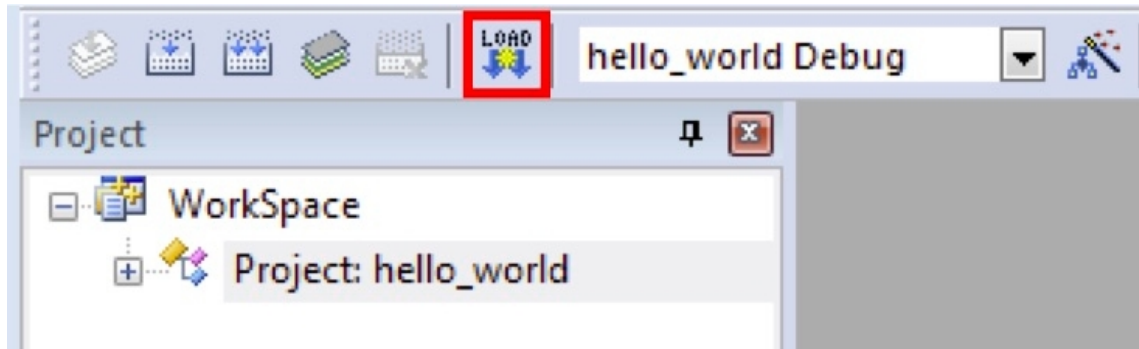
Run an example application To download and run the application, perform these steps:

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

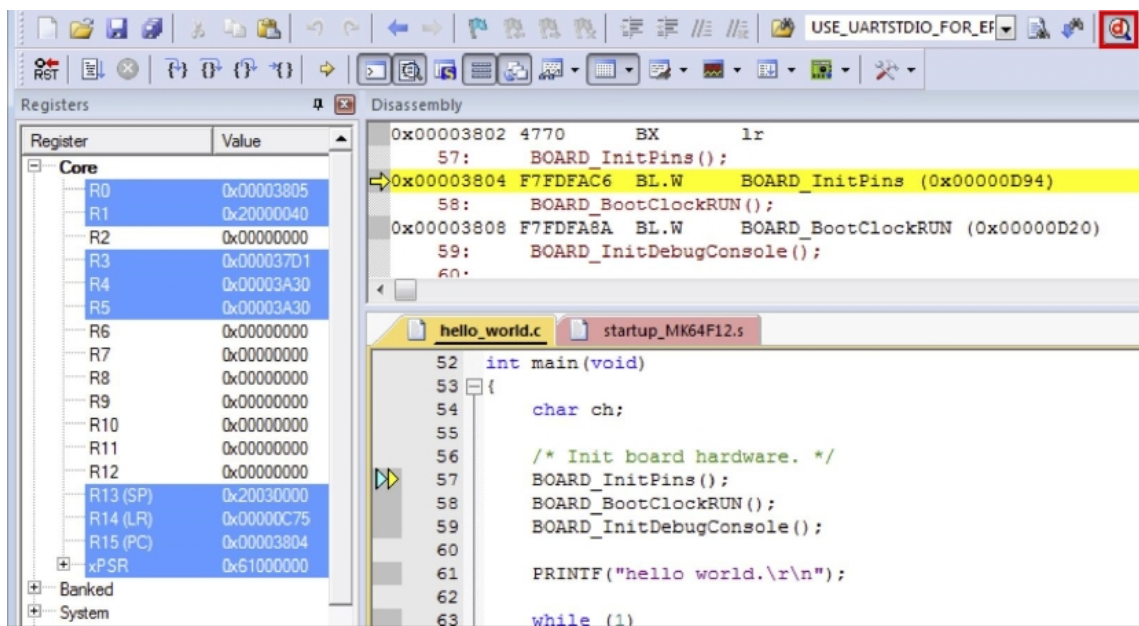


- 1 stop bit

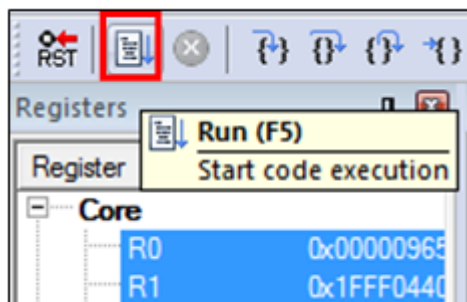
- In μ Vision, after the application is built, click the **Download** button to download the application to the target.



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



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



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



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

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

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

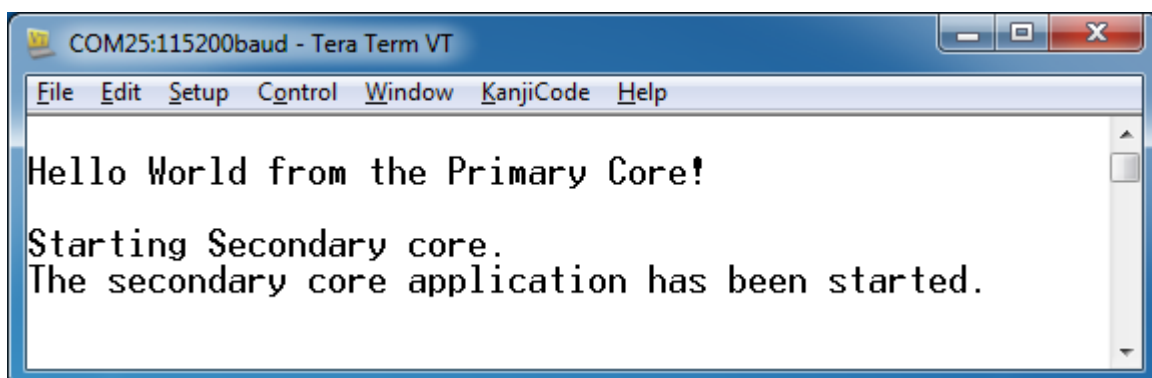
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
→cm0plus.uvmpw
```

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

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

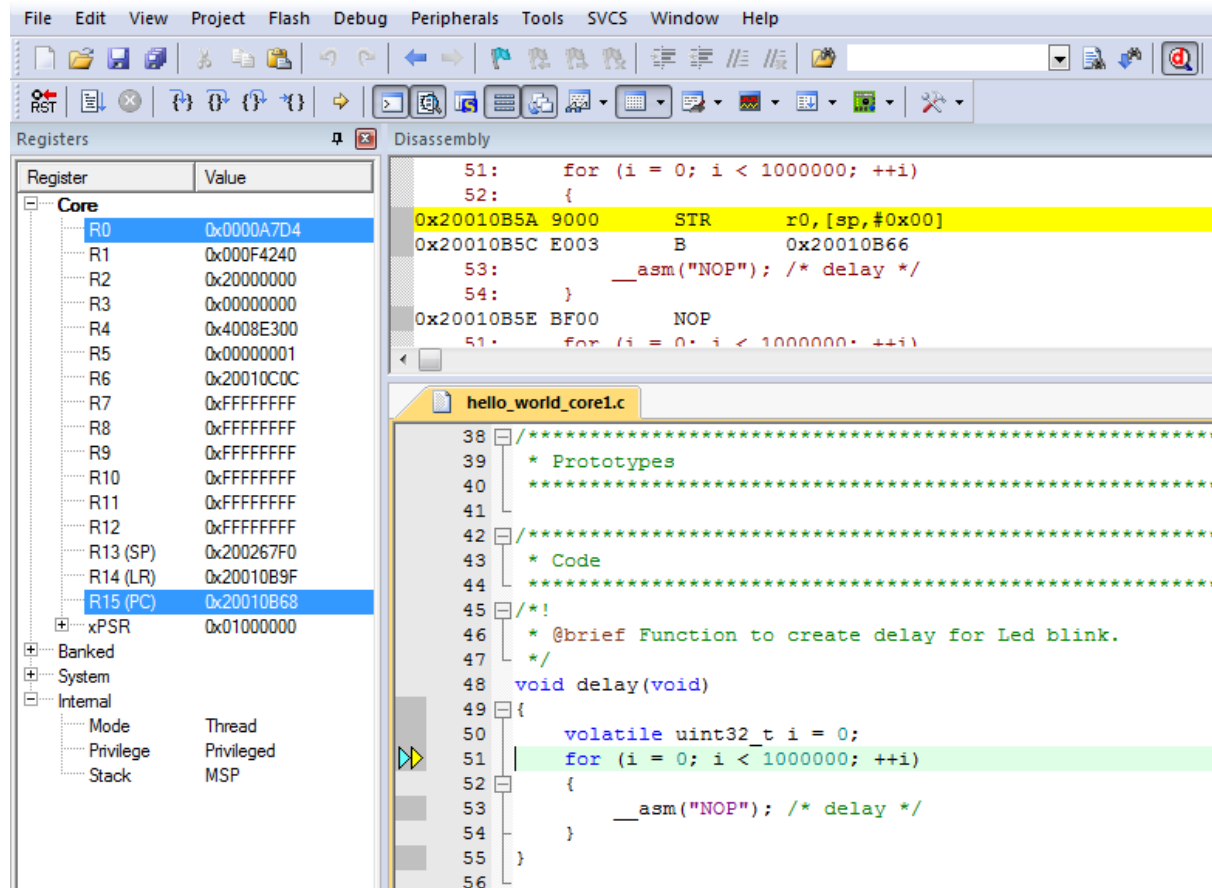
Run a multicore example application The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

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



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

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



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

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

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

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

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

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

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

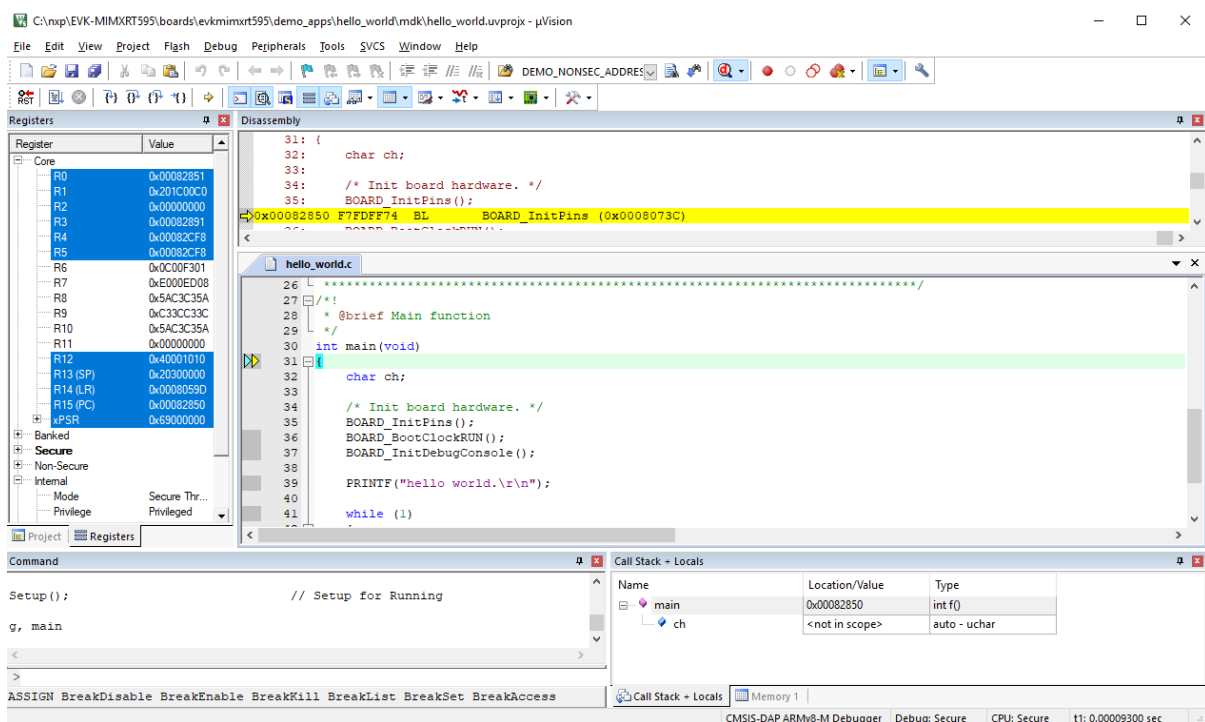
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
→ uvmpw
```

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

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

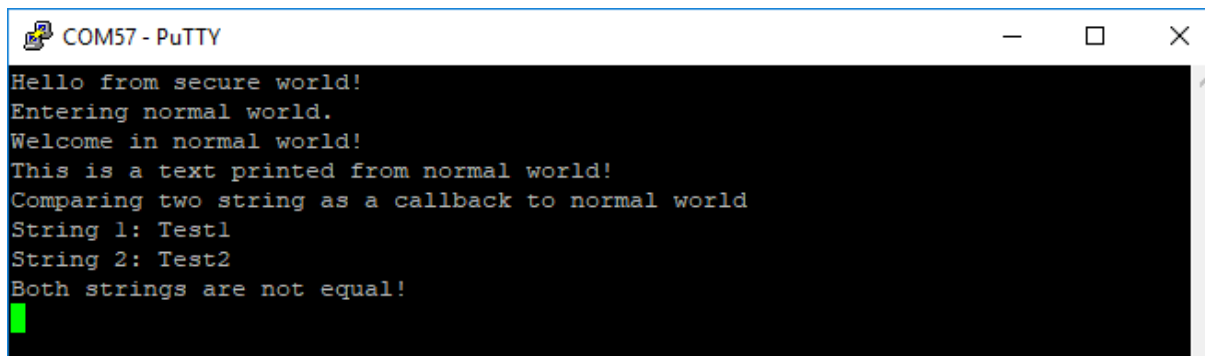
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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



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

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



Run a demo using 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 which is used as an example.

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.

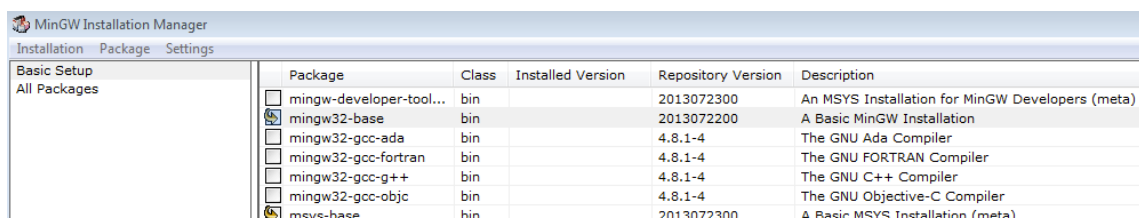
Install GCC Arm Embedded tool chain Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in **MCUXpresso SDK Release Notes**.

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.

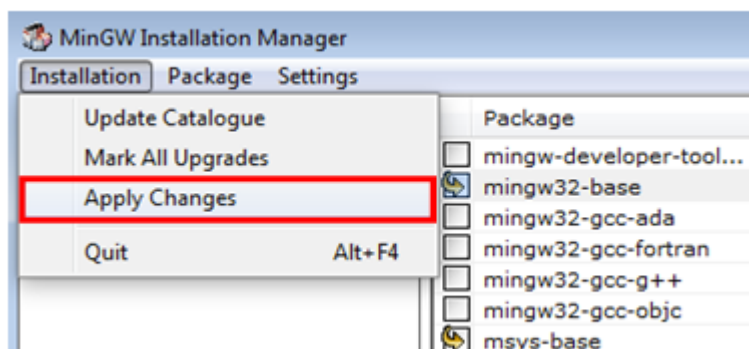
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.

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.

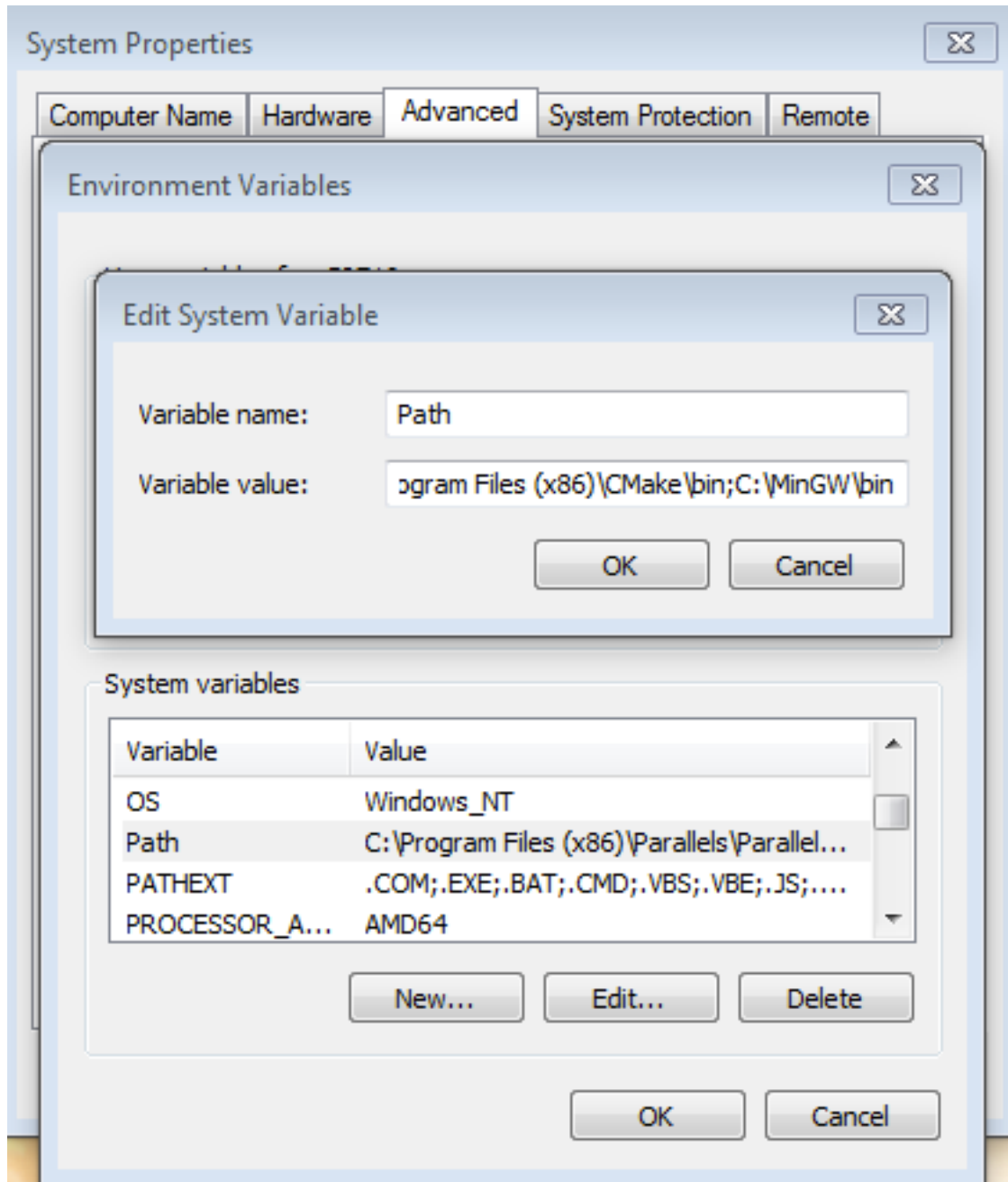


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:


```
<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.



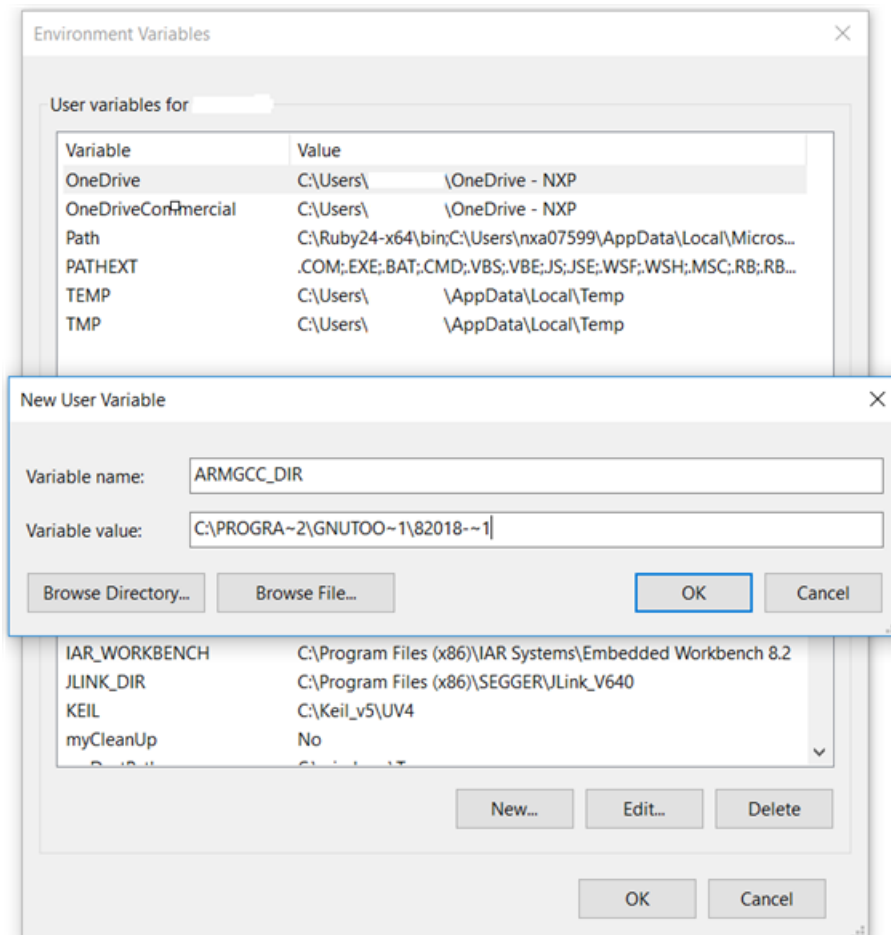
Add a new system environment variable for ARMGCC_DIR Create a new *system* environment variable and name it as ARMGCC_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

```
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 pathname of your installation.

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

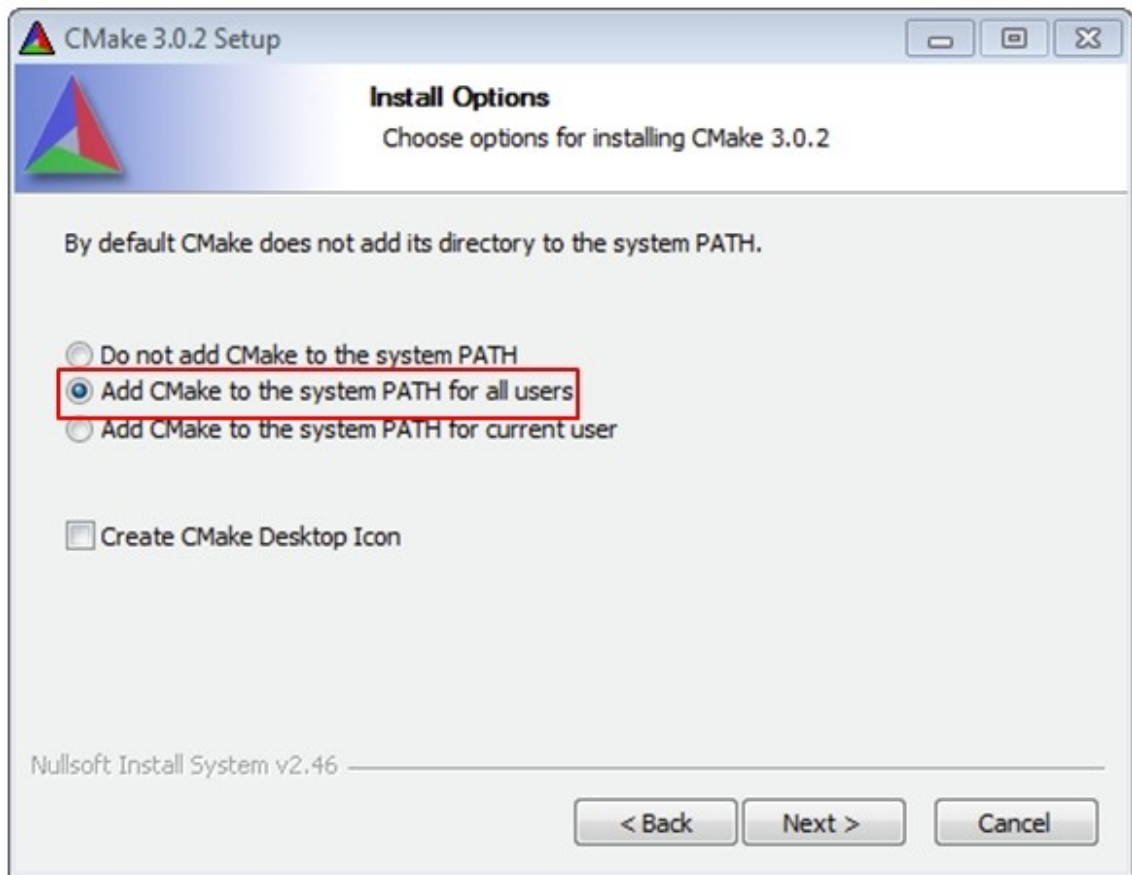
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```



Install CMake

Windows OS

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.

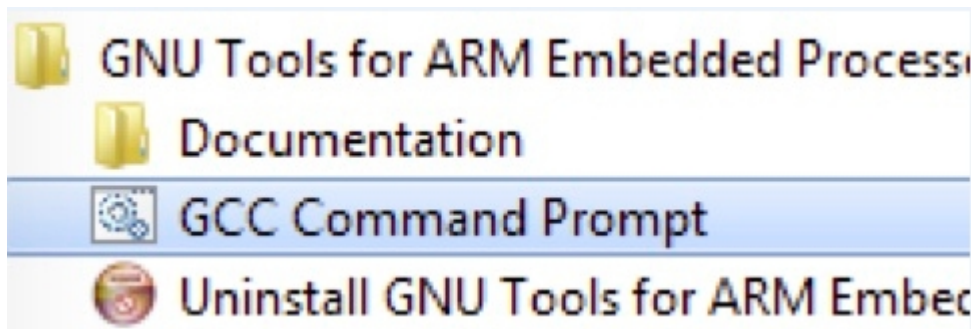
Linux OS It depends on the distributions of Linux Operation System. Here we use Ubuntu as an example.

Open shell and use following commands to install cmake and its version. Ensure the cmake version is above 3.0.x.

```
$ sudo apt-get install cmake
$ cmake --version
```

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:

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 following figure.

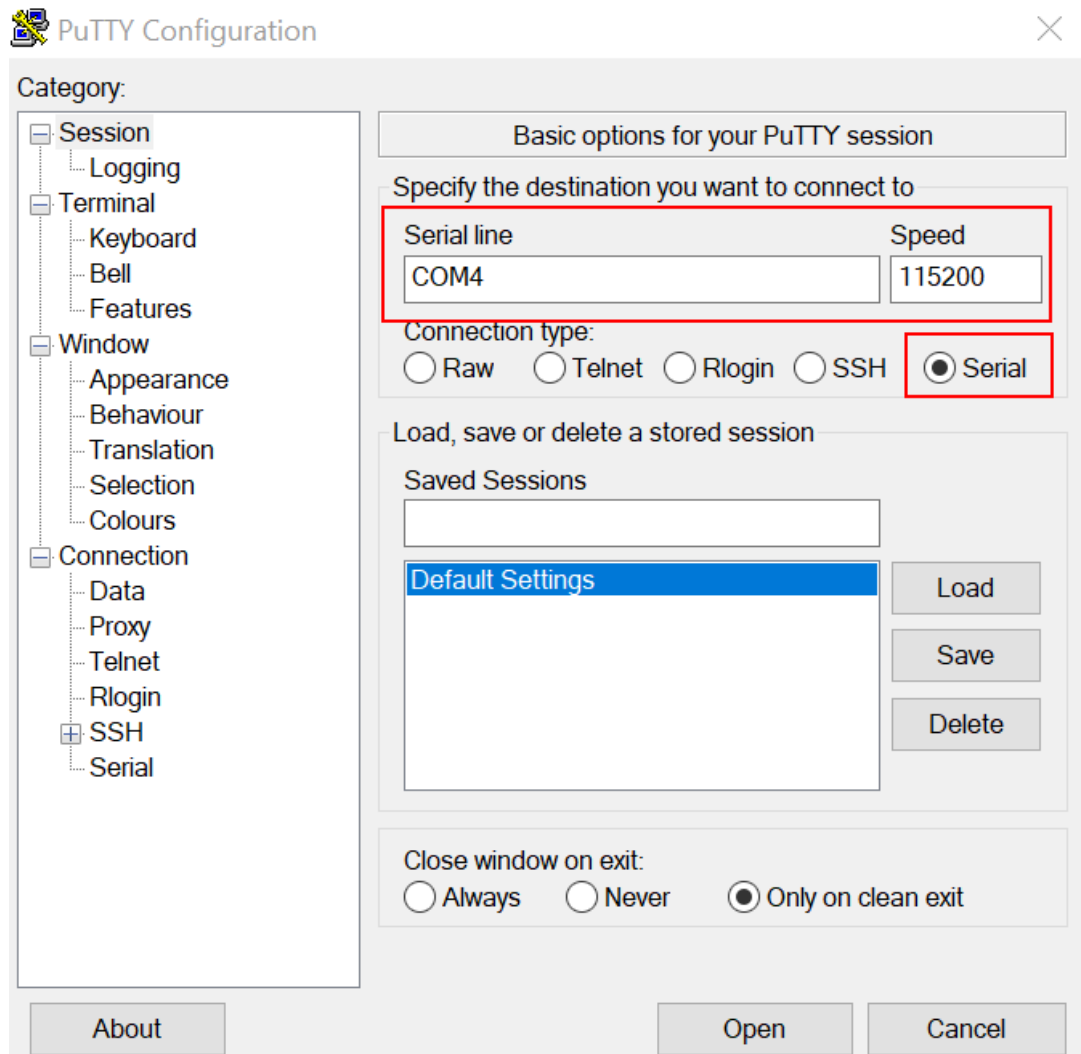
```
[ 84%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_smc.c.obj
[ 92%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_clock.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

C:\nxp\SDK_2.0_FRDM-K64F\boards\frdmk64f\demo_apps\hello_world\armgcc>IF "" == "
" <pause >
Press any key to continue . . .
```

Run an example application This section describes steps to run a demo application using J-Link GDB Server application. To install J-Link host driver and update the on-board debugger firmware to Jlink firmware, see [On-board debugger](#).

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 on-board debugger USB connector 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](#)). 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

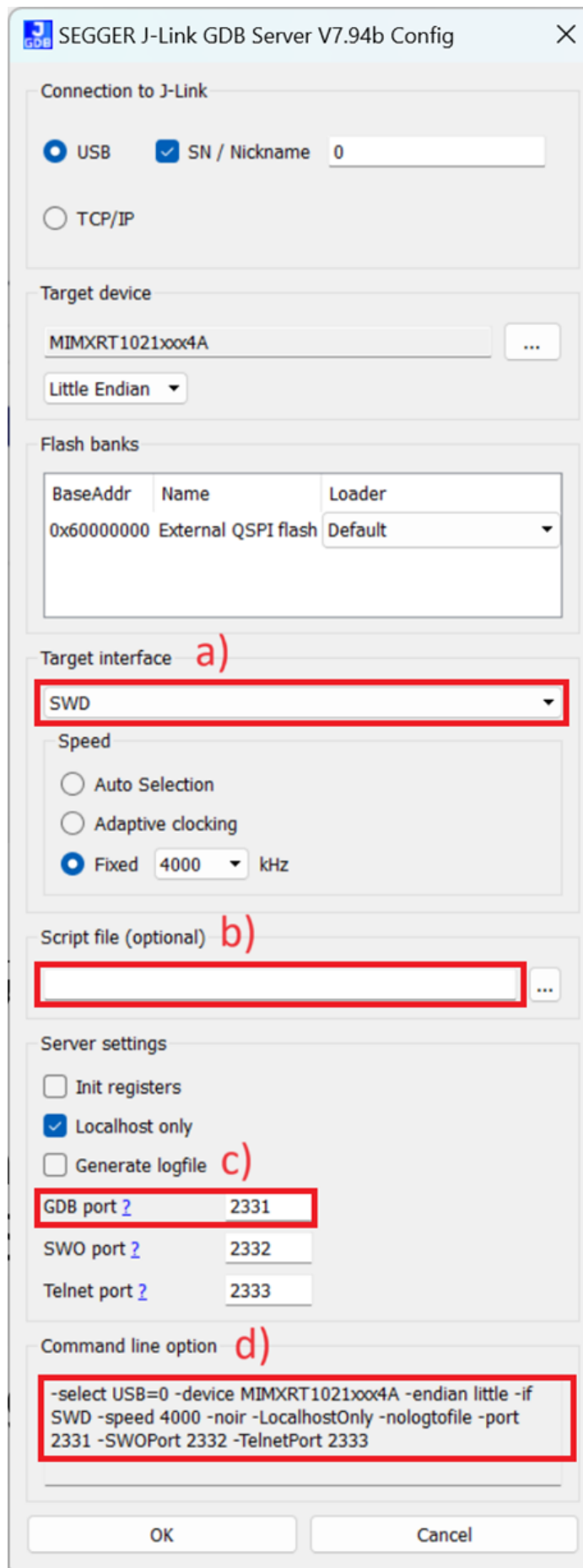


3. To launch the application, open the Windows **Start** menu and select **Programs > SEGGER > J-Link <version> J-Link GDB Server**.

Note: It is assumed that the J-Link software is already installed.

The **SEGGER J-Link GDB Server Config** settings dialog appears.

4. Make sure to check the following options.
 1. **Target interface:** The debug connection on board uses internal SWD signaling. In case of a wrong setting J-Link is unable to communicate with device under test.
 2. **Script file:** If required, a J-Link init script file can be used for board initialization. The file with the ".jlinkscript" file extension is located in the <install_dir>/boards/<board_name>/ directory.
 3. Under the **Server settings**, check the GDB port for connection with the gdb target remote command. For more information, see step 9.
 4. There is a command line version of J-Link GDB server "JLinkGDBServerCL.exe". Typical path is C:\Program Files\SEGGER\JLink\. To start the J-Link GDB server with the same settings as are selected in the UI, you can use these command line options.



The image shows the 'SEGGER J-Link GDB Server V7.94b Config' dialog box. It is divided into several sections: 'Connection to J-Link', 'Target device', 'Flash banks', 'Target interface', 'Script file (optional)', 'Server settings', and 'Command line option'. Red annotations 'a)', 'b)', 'c)', and 'd)' are placed next to the 'Target interface', 'Script file (optional)', 'Generate logfile', and 'Command line option' sections respectively. Red boxes highlight the 'SWD' dropdown, the empty 'Script file' field, the 'GDB port' field, and the 'Command line option' text area.

Connection to J-Link

☒ USB ☒ SN / Nickname 0

☐ TCP/IP

Target device

MIMXRT1021xxx4A ...

Little Endian ▾

Flash banks

BaseAddr	Name	Loader
0x60000000	External QSPI flash	Default ▾

Target interface a)

SWD ▾

Speed

☐ Auto Selection

☐ Adaptive clocking

☒ Fixed 4000 ▾ kHz

Script file (optional) b)

...

Server settings

☐ Init registers

☒ Localhost only

☐ Generate logfile c)

GDB port ? 2331

SWO port ? 2332

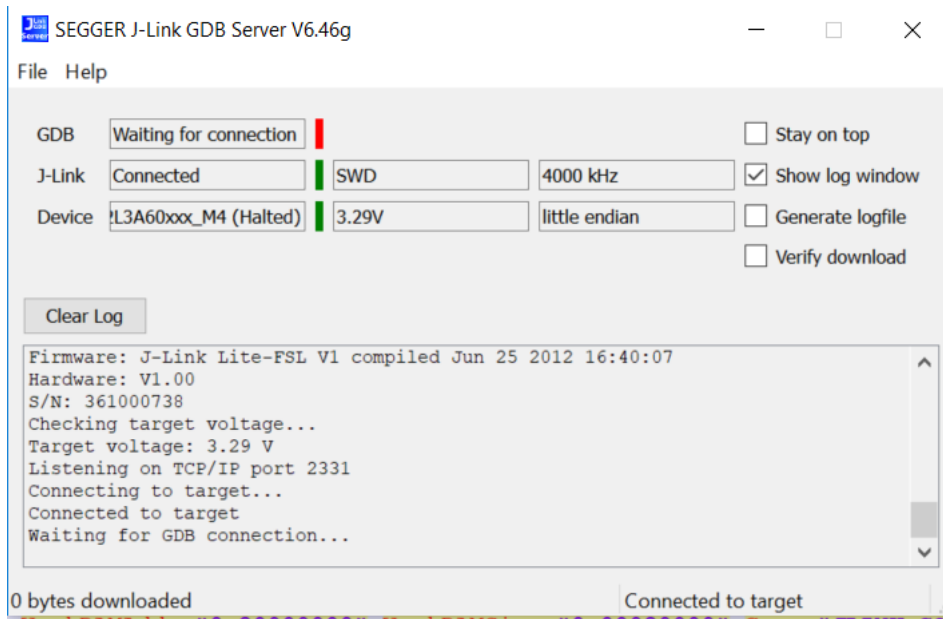
Telnet port ? 2333

Command line option d)

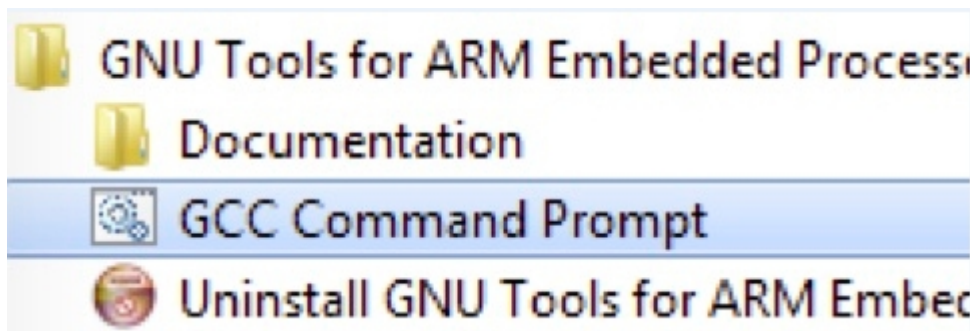
```
-select USB=0 -device MIMXRT1021xxx4A -endian little -if SWD -speed 4000 -noir -LocalhostOnly -nologtofile -port 2331 -SWOPort 2332 -TelnetPort 2333
```

OK Cancel

5. After it is connected, the screen should look like this figure:



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

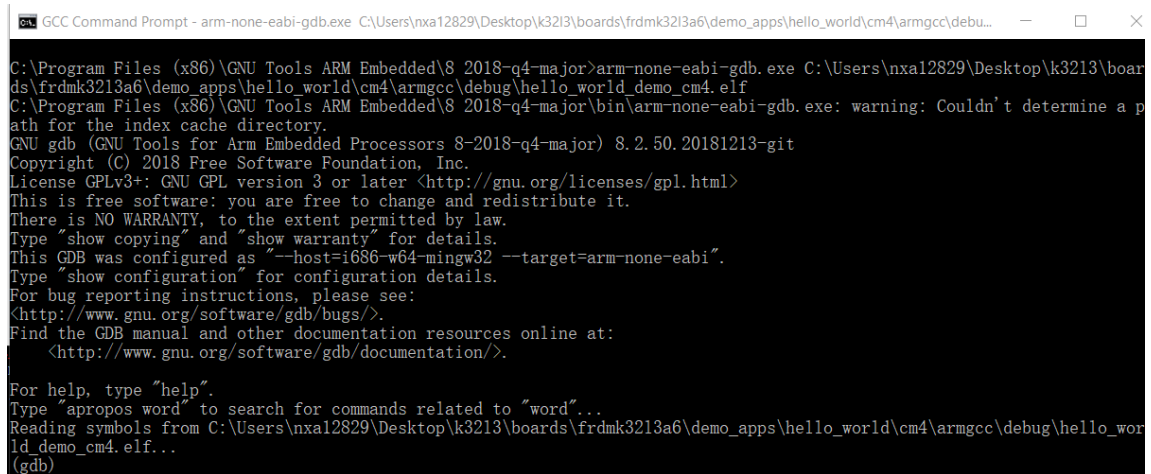


7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

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

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

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



```

GCC Command Prompt - arm-none-eabi-gdb.exe C:\Users\nxa12829\Desktop\k3213\boards\frdmk3213a6\demo_apps\hello_world\cm4\armgcc\debug\hello_world_demo_cm4.elf
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major>arm-none-eabi-gdb.exe C:\Users\nxa12829\Desktop\k3213\boards\frdmk3213a6\demo_apps\hello_world\cm4\armgcc\debug\hello_world_demo_cm4.elf
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin>arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Tools for Arm Embedded Processors 8-2018-q4-major) 8.2.50.20181213-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from C:\Users\nxa12829\Desktop\k3213\boards\frdmk3213a6\demo_apps\hello_world\cm4\armgcc\debug\hello_world_demo_cm4.elf...
(gdb)

```

9. Run these commands:

1. target remote localhost:2331
2. monitor reset
3. monitor halt
4. load
5. monitor reset

10. The application is now downloaded and halted. Execute the `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.



```

COM4 - PuTTY
hello world.

```

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

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

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

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/armgcc/build_debug.bat
```

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/armgcc/build_debug.bat
```

Build both applications separately following steps for single core examples as described in **Build an example application**.

```

GCC Command Prompt - build_debug.bat

[ 47%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_common.c.obj
[ 52%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_msmc.c.obj
[ 56%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/debug_console/fsl_debug_console.c.obj
[ 60%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/fsl_assert.c.obj
[ 65%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/str/fsl_str.c.obj
[ 69%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/uart/lpuart_adapter.c.obj
[ 73%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_manager.c.obj
[ 78%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_port_uart.c.obj
[ 82%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/lists/generic_list.c.obj
[ 86%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/system_K32L3A60_cm0plus.c.obj
[ 91%] Building ASM object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/gcc/startup_K32L3A60_cm0plus.S.obj
[ 95%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/middleware/multicore/mcmgr/src/mcmgr.c.obj
[100%] Linking C executable debug\hello_world_cm0plus.elf
[100%] Built target hello_world_cm0plus.elf

c:\packages\SDK_2.6.0_FRDM-K32L3A6_RC1\boards\frdmk32l3a6\multicore_examples\hello_world\cm0plus\armgcc>IF "" == "" (pause)
Press any key to continue . . .

GCC Command Prompt - build_debug.bat

[ 50%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_lpuart.c.obj
[ 54%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_common.c.obj
[ 58%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_msmc.c.obj
[ 62%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/str/fsl_str.c.obj
[ 66%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/fsl_assert.c.obj
[ 70%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/debug_console/fsl_debug_console.c.obj
[ 75%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/uart/lpuart_adapter.c.obj
[ 79%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_port_uart.c.obj
[ 83%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_manager.c.obj
[ 87%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/lists/generic_list.c.obj
[ 91%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/system_K32L3A60_cm4.c.obj
[ 95%] Building ASM object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/gcc/startup_K32L3A60_cm4.S.obj
[100%] Linking C executable debug\hello_world_cm4.elf
[100%] Built target hello_world_cm4.elf

c:\packages\SDK_2.6.0_FRDM-K32L3A6_RC1\boards\frdmk32l3a6\multicore_examples\hello_world\cm4\armgcc>IF "" == "" (pause)
Press any key to continue . . .

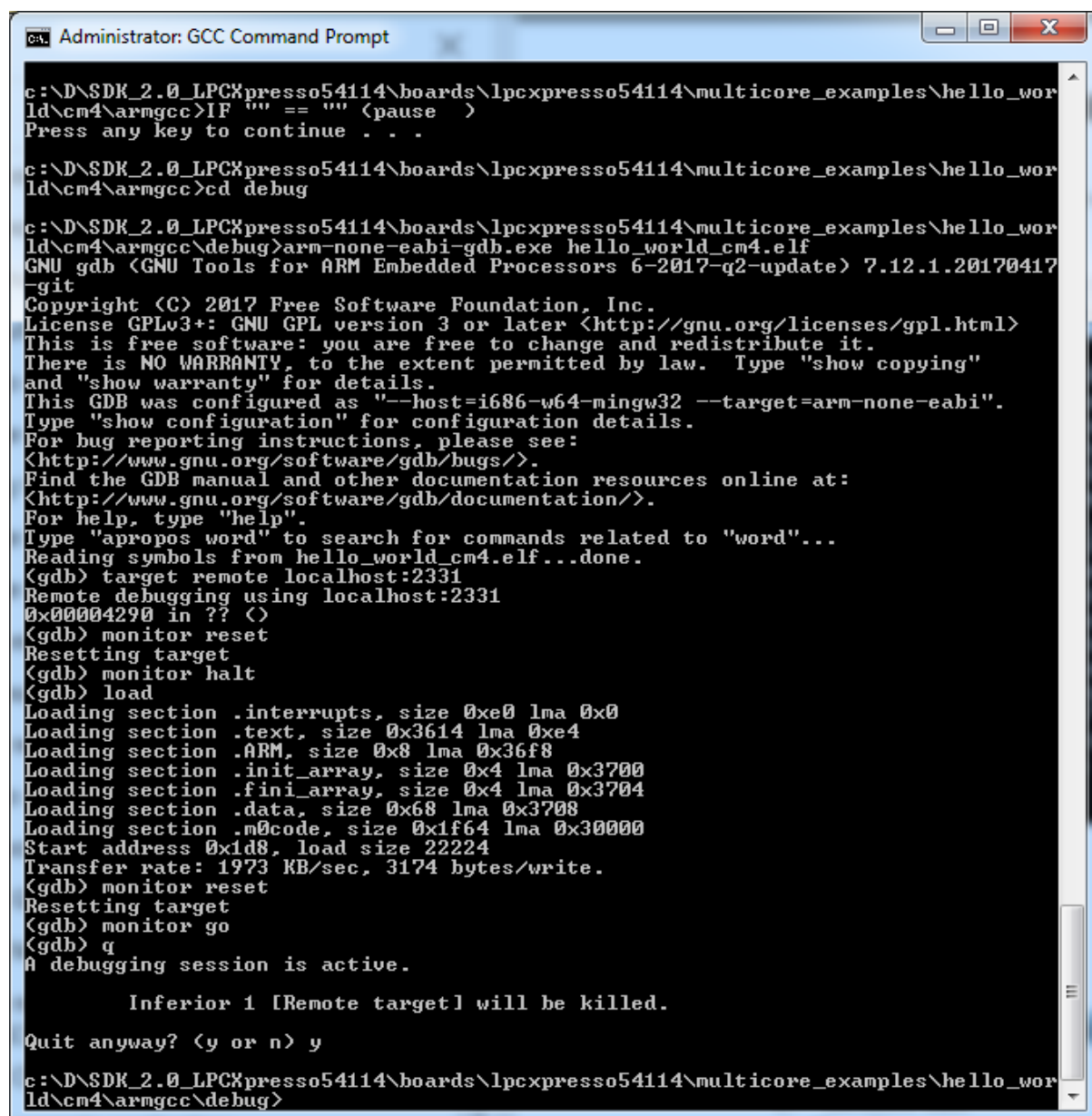
```

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

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 to 10, as described in **Run an example application**. These steps are common for both single-core and dual-core applications in Arm GCC.

Both the primary and the auxiliary image is loaded into the SPI flash memory. After execution of the monitor go command, the primary core application is executed. During the primary core code execution, the auxiliary core code is reallocated from the flash memory to the RAM, and the auxiliary core is released from the reset. The hello_world multicore application is now running

and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```

Administrator: GCC Command Prompt

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc>IF "" == "" <pause >
Press any key to continue . . .

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc>cd debug

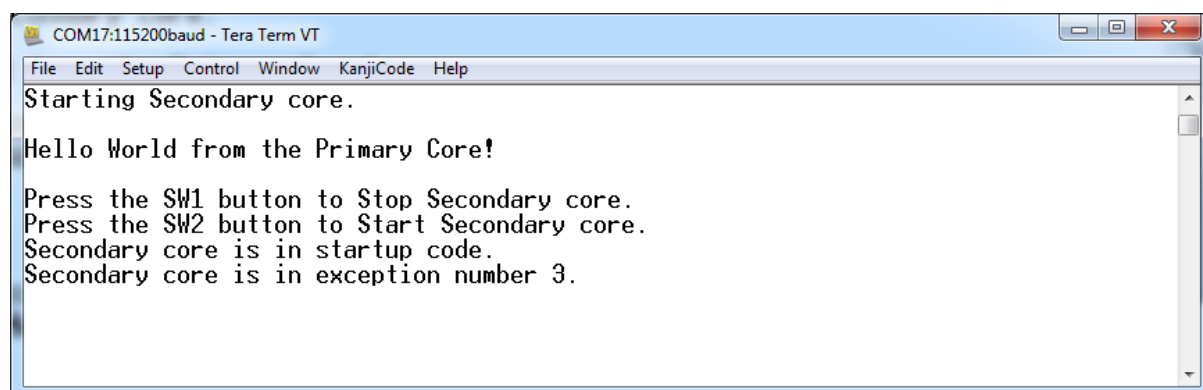
c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc\debug>arm-none-eabi-gdb.exe hello_world_cm4.elf
GNU gdb (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 7.12.1.20170417-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world_cm4.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00004290 in ?? ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0xe0 lma 0x0
Loading section .text, size 0x3614 lma 0xe4
Loading section .ARM, size 0x8 lma 0x36f8
Loading section .init_array, size 0x4 lma 0x3700
Loading section .fini_array, size 0x4 lma 0x3704
Loading section .data, size 0x68 lma 0x3708
Loading section .m0code, size 0x1f64 lma 0x30000
Start address 0x1d8, load size 22224
Transfer rate: 1973 KB/sec, 3174 bytes/write.
(gdb) monitor reset
Resetting target
(gdb) monitor go
(gdb) q
A debugging session is active.

    Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc\debug>

```



```

COM17:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Starting Secondary core.

Hello World from the Primary Core!

Press the SW1 button to Stop Secondary core.
Press the SW2 button to Start Secondary core.
Secondary core is in startup code.
Secondary core is in exception number 3.

```

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>]/
↪<application_name>_ns/armgcc
```

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/[<core_type>]/
↪<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/<board_name>/trustzone_examples/hello_world/hello_world_ns/armgcc/build_
↪debug.bat
```

```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/armgcc/build_
↪debug.bat
```

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 must know the secure project, since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because the CMSE library is not ready.

```
C:\WINDOWS\system32\cmd.exe
[ 55%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fsl_
l_assert.c.obj
[ 59%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c
.obj
[ 62%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexspi.c.obj
[ 66%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
cache.c.obj
[ 70%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
l_manager.c.obj
[ 74%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
l_port_uart.c.obj
[ 77%] Building ASM object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startu
p_MIMXRT595S_cm33.S.obj
[ 81%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c
.obj
[ 85%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
usart.c.obj
[ 88%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
iap.c.obj
[100%] Linking C executable debug\hello_world_s.elf
[100%] Built target hello_world_s.elf
C:\npx\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_s\armgcc>IF "" == "" (paus
e)
Press any key to continue . . .
```

```

C:\WINDOWS\system32\cmd.exe
[ 52%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c.obj
[ 56%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fsl_assert.c.obj
[ 60%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexspi.c.obj
[ 64%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_cache.c.obj
[ 68%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_manager.c.obj
[ 72%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_port_uart.c.obj
[ 76%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_usart.c.obj
[ 80%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c.obj
[ 84%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startup_MIMXRT595S_cm33.S.obj
[ 88%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_iap.c.obj
[100%] Linking C executable debug\hello_world_ns.elf
[100%] Built target hello_world_ns.elf

C:\npx\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_ns\armgcc>IF "" == "" (pause)
Press any key to continue . . .

```

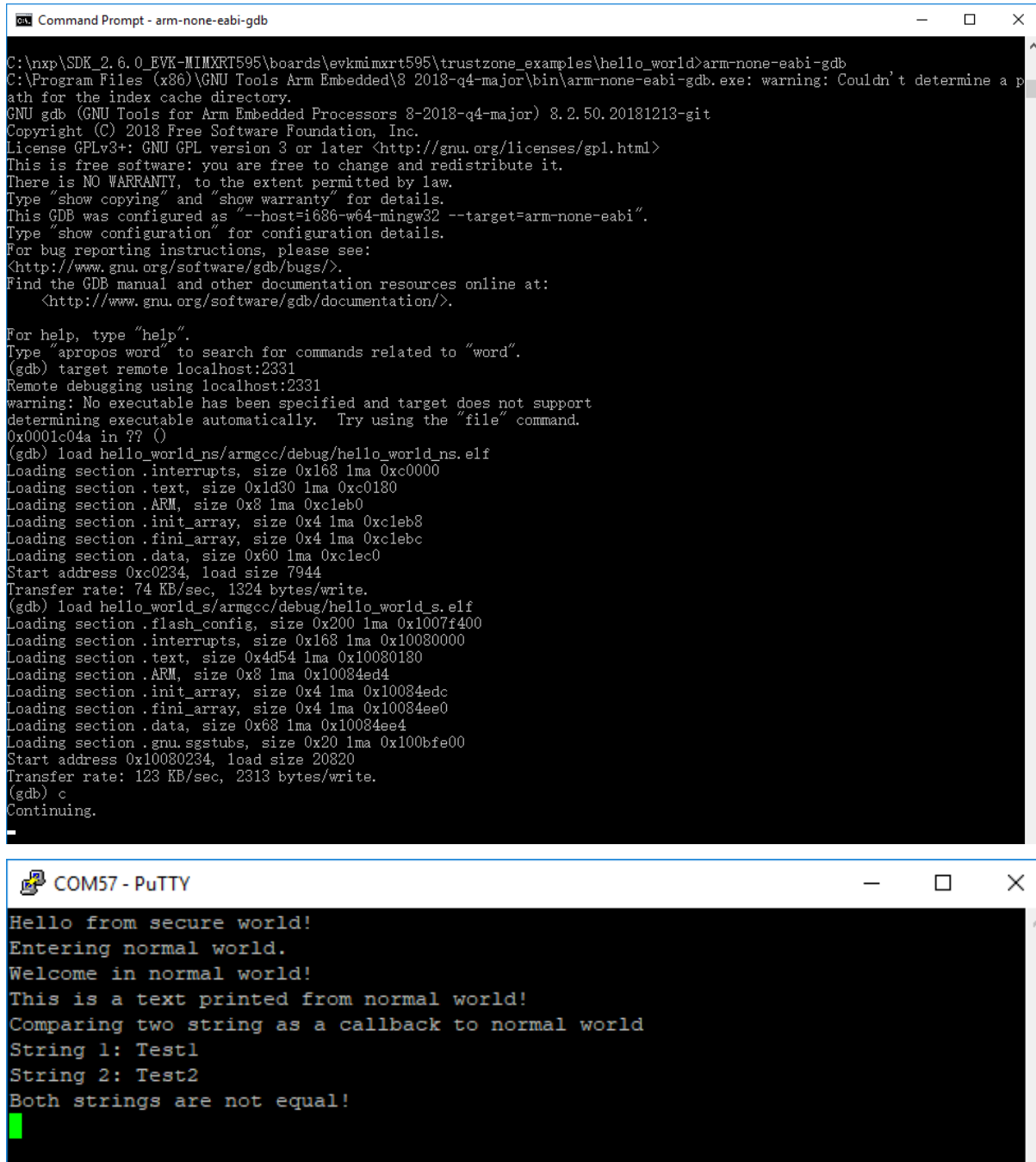
Run a TrustZone example application When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in **Run an example application**.

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. monitor exec SetFlashDLNoRMWThreshold = 0x20000
6. load <install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_ns/armgcc/debug/hello_world_ns.elf
7. load <install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_s/armgcc/debug/hello_world_s.elf
8. monitor reset

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



```

Command Prompt - arm-none-eabi-gdb

C:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world>arm-none-eabi-gdb
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Tools for Arm Embedded Processors 8-2018-q4-major) 8.2.50.20181213-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x0001c04a in ?? ()
(gdb) load hello_world_ns/armgcc/debug/hello_world_ns.elf
Loading section .interrupts, size 0x168 lma 0xc0000
Loading section .text, size 0x1d30 lma 0xc0180
Loading section .ARM, size 0x8 lma 0xc1eb0
Loading section .init_array, size 0x4 lma 0xc1eb8
Loading section .fini_array, size 0x4 lma 0xc1ebc
Loading section .data, size 0x60 lma 0xc1ec0
Start address 0xc0234, load size 7944
Transfer rate: 74 KB/sec, 1324 bytes/write.
(gdb) load hello_world_s/armgcc/debug/hello_world_s.elf
Loading section .flash_config, size 0x200 lma 0x1007f400
Loading section .interrupts, size 0x168 lma 0x10080000
Loading section .text, size 0x4d54 lma 0x10080180
Loading section .ARM, size 0x8 lma 0x10084ed4
Loading section .init_array, size 0x4 lma 0x10084edc
Loading section .fini_array, size 0x4 lma 0x10084ee0
Loading section .data, size 0x68 lma 0x10084ee4
Loading section .gnu.sgstubs, size 0x20 lma 0x100bfe00
Start address 0x10080234, load size 20820
Transfer rate: 123 KB/sec, 2313 bytes/write.
(gdb) c
Continuing.

```

```

COM57 - PuTTY

Hello from secure world!
Entering normal world.
Welcome in normal world!
This is a text printed from normal world!
Comparing two string as a callback to normal world
String 1: Test1
String 2: Test2
Both strings are not equal!

```

MCUXpresso Config Tools

MCUXpresso Config Tools can help configure the processor and generate initialization code for the on chip peripherals. The tools are able to modify any existing example project, or create a new configuration for the selected board or processor. The generated code is designed to be used with MCUXpresso SDK version 24.12.00 or later.

Following table describes the tools included in the MCUXpresso Config Tools.

Config Tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	
Clock tool	For system clock configuration	
Peripherals tools	For configuration of other peripherals	
TEE tool	Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.	
Device Configuration tool	Configures Device Configuration Data (DCD) contained in the program image that the Boot ROM code interprets to set up various on-chip peripherals prior to the program launch.	

MCUXpresso Config Tools can be accessed in the following products:

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

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

How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform. All NXP boards ship with a factory programmed, onboard debug interface, whether it is based on MCU-Link or the legacy OpenSDA, LPC-Link2, P&E Micro OSJTAG interface. To determine what your specific board ships with, see [Default debug interfaces](#).

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

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

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

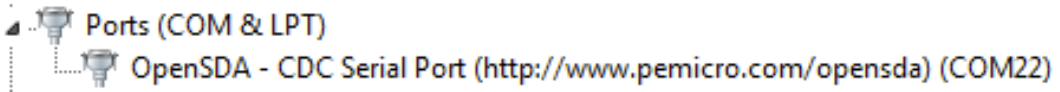
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names are different for all the NXP boards.

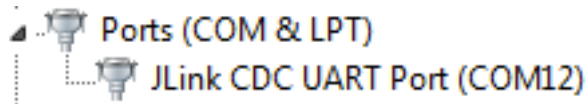
1. **CMSIS-DAP/mbd/DAPLink** interface:



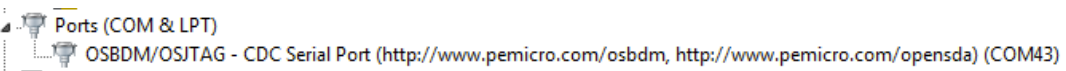
2. P&E Micro:



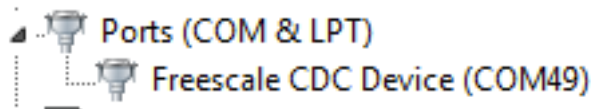
3. J-Link:



4. P&E Micro OSJTAG:



5. MRB-KW01:



On-board Debugger

This section describes the on-board debuggers used on NXP development boards.

On-board debugger MCU-Link MCU-Link is a powerful and cost effective debug probe that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol. MCU-Link also includes a USB to UART bridge feature (VCOM) that can be used to provide a serial connection between the target MCU and a host computer. MCU-Link features a high-speed USB interface for high performance debug. MCU-Link is compatible with Windows, MacOS and Linux. A free utility from NXP provides an easy way to install firmware updates.

On-board MCU-Link debugger supports CMSIS-DAP and J-Link firmware. See the table in [Default debug interfaces](#) to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.

Updating MCU-Link firmware This firmware in this debug interface may be updated using the host computer utility called MCU-Link. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to reprogram the debug probe firmware.

Note: If MCUXpresso IDE is used and the jumper making DFUlink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), MCU-Link debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the

CMSIS-DAP firmware to the probe before flash memory programming (after clicking **Debug**). Using DFU mode ensures that most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the MCU-Link utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link or NXP boards. The utility can be downloaded from [MCU-Link](#).

These steps show how to update the debugger firmware on your board for Windows operating system.

1. Install the MCU-Link utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labeled DFUlink).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the MCU-Link installation directory (<MCU-Link install dir>).
 1. To program CMSIS-DAP debug firmware: <MCU-Link install dir>/scripts/program_CMSIS
 2. To program J-Link debug firmware: <MCU-Link install dir>/scripts/program_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

On-board debugger LPC-Link LPC-Link 2 is an extensible debug probe that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol. MCU-Link also includes a USB to UART bridge feature (VCOM) that can be used to provide a serial connection between the target MCU and a host computer. LPC-Link 2 is compatible with Windows, MacOS and Linux. A free utility from NXP provides an easy way to install firmware updates.

On-board LPC-Link 2 debugger supports CMSIS-DAP and J-Link firmware. See the table in [Default debug interfaces](#) to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.

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

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

NXP provides the LPCScript utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from [LPCScript](#).

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScript user guide ([LPCScript](#), select **LPCScript**, and then the documentation tab).

1. Install the LPCScript utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labeled DFUlink).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the LPCScript installation directory (<LPCScript install dir>).
 1. To program CMSIS-DAP debug firmware: <LPCScript install dir>/scripts/program_CMSIS
 2. To program J-Link debug firmware: <LPCScript install dir>/scripts/program_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

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

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

See the table in [Default debug interfaces](#) to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

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

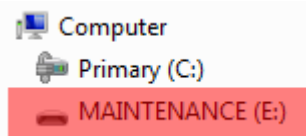
Updating OpenSDA firmware Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. This typically means to switch from the default application (either CMSIS-DAP or P&E Micro) to a SEGGER J-Link. This section contains the steps to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

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

- P&E Micro: Downloading P&E Micro OpenSDA firmware images requires registration with P&E Micro (www.pemicro.com).

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

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



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

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

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

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

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

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

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

The host driver must be installed before debugging.

- See [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

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

The host driver must be installed before debugging.

- See [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

Default debug interfaces

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

Hardware platform	Default debugger firmware	On-board debugger probe
EVK-MCIMX7ULP	N/A	N/A
EVK-MIMX8MM	N/A	N/A
EVK-MIMX8MN	N/A	N/A
EVK-MIMX8MND3L	N/A	N/A
EVK-MIMX8MP	N/A	N/A
EVK-MIMX8MQ	N/A	N/A
EVK-MIMX8ULP	N/A	N/A
EVK-MIMXRT1010	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1015	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1020	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1064	CMSIS-DAP	LPC-Link2
EVK-MIMXRT595	CMSIS-DAP	LPC-Link2
EVK-MIMXRT685	CMSIS-DAP	LPC-Link2
EVK9-MIMX8ULP	N/A	N/A
EVKB-IMXRT1050	CMSIS-DAP	LPC-Link2
FRDM-K22F	CMSIS-DAP	OpenSDA v2
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2
FRDM-K32L3A6	CMSIS-DAP	OpenSDA v2
FRDM-KE02Z40M	P&E Micro	OpenSDA v1
FRDM-KE15Z	CMSIS-DAP	OpenSDA v2
FRDM-KE16Z	CMSIS-DAP	OpenSDA v2
FRDM-KE17Z	CMSIS-DAP	OpenSDA v2
FRDM-KE17Z512	CMSIS-DAP	MCU-Link
FRDM-MCXA153	CMSIS-DAP	MCU-Link
FRDM-MCXA156	CMSIS-DAP	MCU-Link
FRDM-MCXA346	CMSIS-DAP	MCU-Link
FRDM-MCXC041	CMSIS-DAP	MCU-Link
FRDM-MCXC242	CMSIS-DAP	MCU-Link
FRDM-MCXC444	CMSIS-DAP	MCU-Link
FRDM-MCXE247	CMSIS-DAP	MCU-Link
FRDM-MCXN236	CMSIS-DAP	MCU-Link
FRDM-MCXN947	CMSIS-DAP	MCU-Link
FRDM-MCXW23	CMSIS-DAP	MCU-Link
FRDM-MCXW71	CMSIS-DAP	MCU-Link
FRDM-MCXW72	CMSIS-DAP	MCU-Link
FRDM-RW612	CMSIS-DAP	MCU-Link
IMX943-EVK	N/A	N/A
IMX95LP4XEVK-15	N/A	N/A
IMX95LPD5EVK-19	N/A	N/A
IMX95VERDINEVK	N/A	N/A
KW45B41Z-EVK	CMSIS-DAP	MCU-Link
KW45B41Z-LOC	CMSIS-DAP	MCU-Link
KW47-EVK	CMSIS-DAP	MCU-Link
KW47-LOC	CMSIS-DAP	MCU-Link
LPC845BREAKOUT	CMSIS-DAP	LPC-Link2
LPCXpresso51U68	CMSIS-DAP	LPC-Link2
LPCXpresso54628	CMSIS-DAP	LPC-Link2

continues on next page

Table 1 – continued from previous page

Hardware platform	Default debugger firmware	On-board debugger probe
LPCXpresso54S018	CMSIS-DAP	LPC-Link2
LPCXpresso54S018M	CMSIS-DAP	LPC-Link2
LPCXpresso55S06	CMSIS-DAP	LPC-Link2
LPCXpresso55S16	CMSIS-DAP	LPC-Link2
LPCXpresso55S28	CMSIS-DAP	LPC-Link2
LPCXpresso55S36	CMSIS-DAP	MCU-Link
LPCXpresso55S69	CMSIS-DAP	LPC-Link2
LPCXpresso802	CMSIS-DAP	LPC-Link2
LPCXpresso804	CMSIS-DAP	LPC-Link2
LPCXpresso824MAX	CMSIS-DAP	LPC-Link2
LPCXpresso845MAX	CMSIS-DAP	LPC-Link2
LPCXpresso860MAX	CMSIS-DAP	LPC-Link2
MC56F80000-EVK	P&E Micro	Multilink
MC56F81000-EVK	P&E Micro	Multilink
MC56F83000-EVK	P&E Micro	OSJTAG
MCIMX93-EVK	N/A	N/A
MCIMX93-QSB	N/A	N/A
MCIMX93AUTO-EVK	N/A	N/A
MCX-N5XX-EVK	CMSIS-DAP	MCU-Link
MCX-N9XX-EVK	CMSIS-DAP	MCU-Link
MCX-W71-EVK	CMSIS-DAP	MCU-Link
MCX-W72-EVK	CMSIS-DAP	MCU-Link
MIMXRT1024-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1040-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1060-EVKB	CMSIS-DAP	LPC-Link2
MIMXRT1060-EVKC	CMSIS-DAP	MCU-Link
MIMXRT1160-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1170-EVKB	CMSIS-DAP	MCU-Link
MIMXRT1180-EVK	CMSIS-DAP	MCU-Link
MIMXRT685-AUD-EVK	CMSIS-DAP	LPC-Link2
MIMXRT700-EVK	CMSIS-DAP	MCU-Link
RD-RW612-BGA	CMSIS-DAP	MCU-Link
TWR-KM34Z50MV3	P&E Micro	OpenSDA v1
TWR-KM34Z75M	P&E Micro	OpenSDA v1
TWR-KM35Z75M	CMSIS-DAP	OpenSDA v2
TWR-MC56F8200	P&E Micro	OSJTAG
TWR-MC56F8400	P&E Micro	OSJTAG

How to define IRQ handler in CPP files

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

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

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

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

1.3 Getting Started with MCUXpresso SDK GitHub

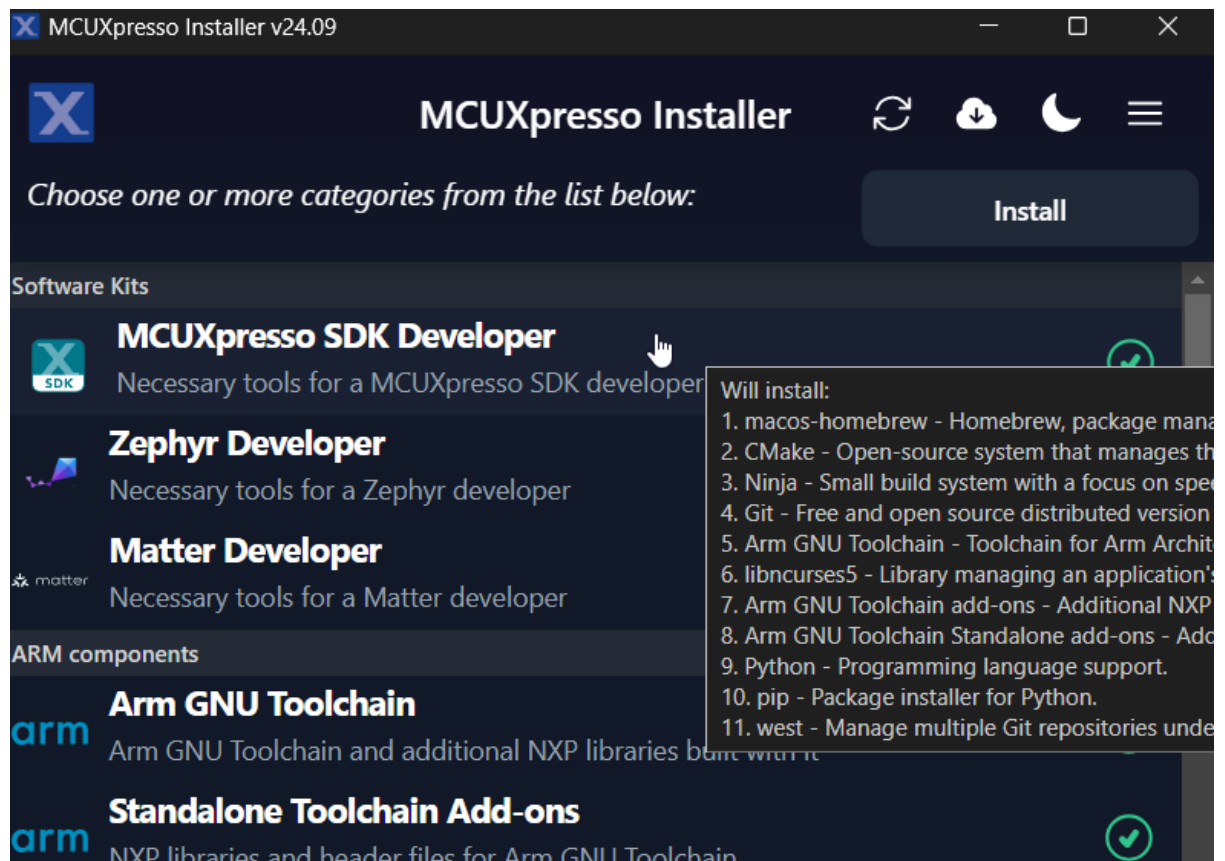
1.3.1 Getting Started with MCUXpresso SDK Repository

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 OOB 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	AR-MGCC_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	ARM-CLANG_DIR	C:\ArmCompilerforEmbedded6.22 for Windows/opt/ArmCompilerforEmbedded6.21 for Linux	– toolchain mdk
Zephyr	ZEPHYR_DIR	c:\NXP\zephyr-sdk-<version> for windows/opt/zephyr-sdk-<version> for Linux	– toolchain zephyr
CodeWarrior	CW_DIR	C:\Freescall\CW MCU v11.2 for windowsCodeWarrior is not supported with Linux	– toolchain code-warrior
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	RISCV-LVM_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 riscv-llvm

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

```

└─ arm
└─ common
└─ install-info

```

- 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 <code>west manifest config group.filter +optional</code> and <code>west update mcux-soc-svd</code> 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

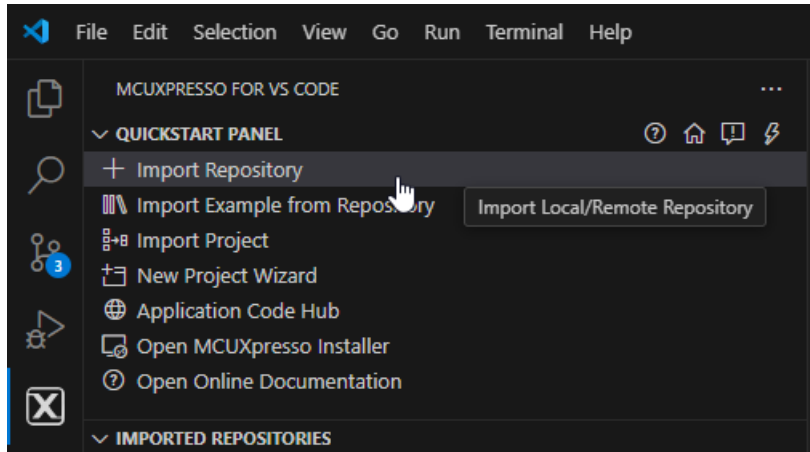
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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](#).

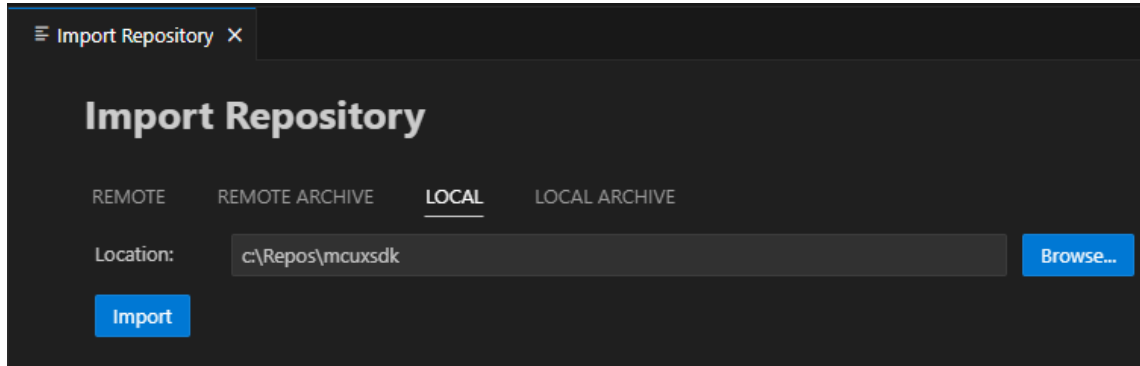
To build an example application:

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

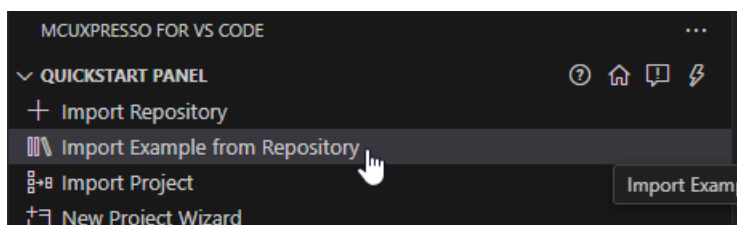


Note: You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

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**.

Import Example from Repository

Repository: c:\Repos\mcuxsdk (MCUXpresso SDK Repository) | v

Toolchain: (Arm GNU Toolchain 13.2.rel1 (Build arm-13.7)) 13.2.1 20231009 | v

Board: FRDM-MCXC444 | v

 FRDM-MCXC444

Template: demo_apps/hello_world | v

The HelloWorld demo prints the "Hello World" string to the terminal using the SDK UART drivers and repeat what user input. The purpose of this demo is to show how to use the UART, and to provide a simple project for debugging and further development.
Please refer to [README](#) file for more details.

App type: Freestanding application | v

Name: frdmmcxc444_hello_world

Location: c:\nxp_examples [Browse...](#)

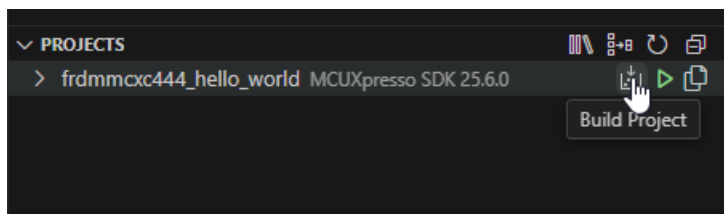
Note: Path doesn't exist. Folder(s) will be created.

☐ Open readme file after project is imported

[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.

- VS Code will prompt you to confirm if the imported files are trusted. Click **Yes**.
- 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.

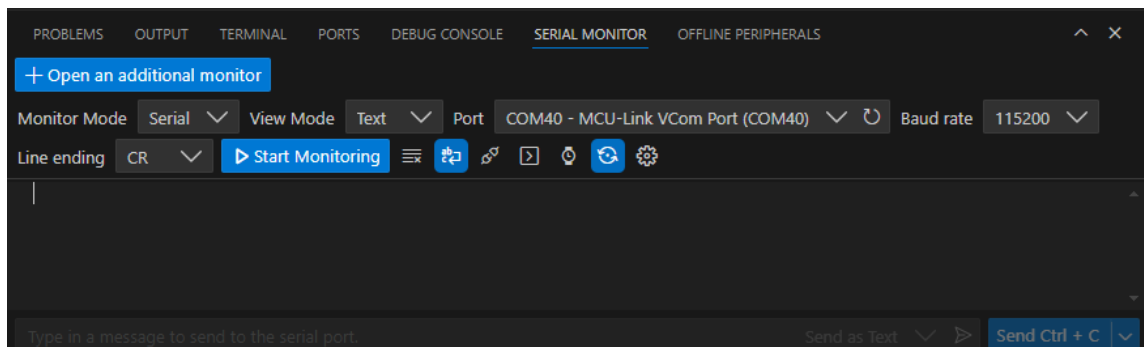
```

[17/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/components/debug_console_lite/fsl_debug_console.c.obj
[18/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/devices/MCXC/MCXC444/drivers/fsl_clock.c.obj
[19/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/drivers/lpuart/fsl_lpuart.c.obj
[20/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/drivers/uart/fsl_uart.c.obj
[21/21] Linking C executable hello_world.elf
Memory region      Used Size  Region Size  %age Used
  m_interrupts:     192 B      512 B      37.50%
  m_flash_config:    16 B       16 B     100.00%
  m_text:          7892 B    261104 B      3.02%
  m_data:          2128 B     32 KB      6.49%
build finished successfully.
Terminal will be reused by tasks, press any key to close it.

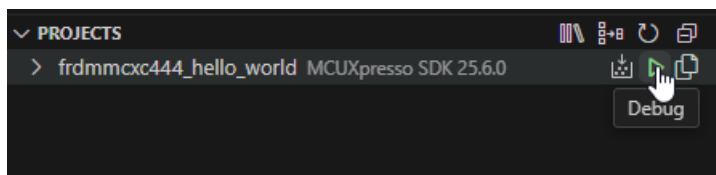
```

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

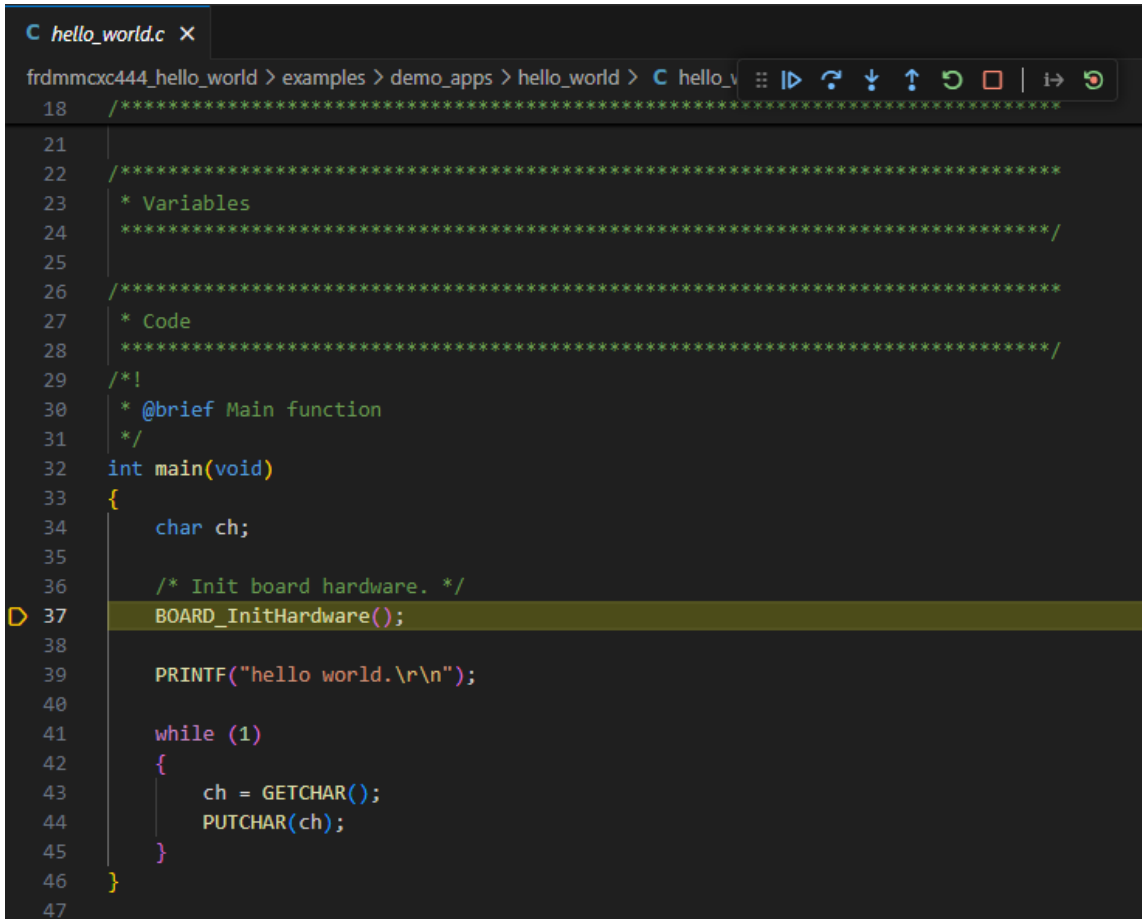
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



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.

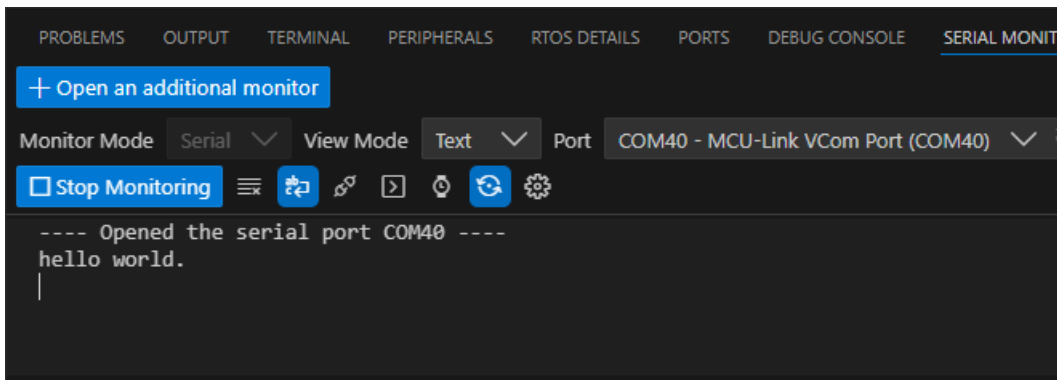


```

18  /*****
21
22  /*****
23  * Variables
24  *****/
25
26  /*****
27  * Code
28  *****/
29  /*!
30  * @brief Main function
31  */
32  int main(void)
33  {
34      char ch;
35
36      /* Init board hardware. */
37      BOARD_InitHardware();
38
39      PRINTF("hello world.\r\n");
40
41      while (1)
42      {
43          ch = GETCHAR();
44          PUTCHAR(ch);
45      }
46  }
47

```

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



```

PROBLEMS  OUTPUT  TERMINAL  PERIPHERALS  RTOS DETAILS  PORTS  DEBUG CONSOLE  SERIAL MONITOR
+ Open an additional monitor
Monitor Mode Serial View Mode Text Port COM40 - MCU-Link VCom Port (COM40)
[Stop Monitoring] [Icons]
---- Opened the serial port COM40 ----
hello world.
|

```

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)

```

↪ evkbnimxrt1060]
INFO: [ 4][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm4]
INFO: [ 5][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm7]
INFO: [ 6][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1060]
INFO: [ 7][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt7ulp]
...

```

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 evkbnimxrt1170 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 evkbnimxrt1170 --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

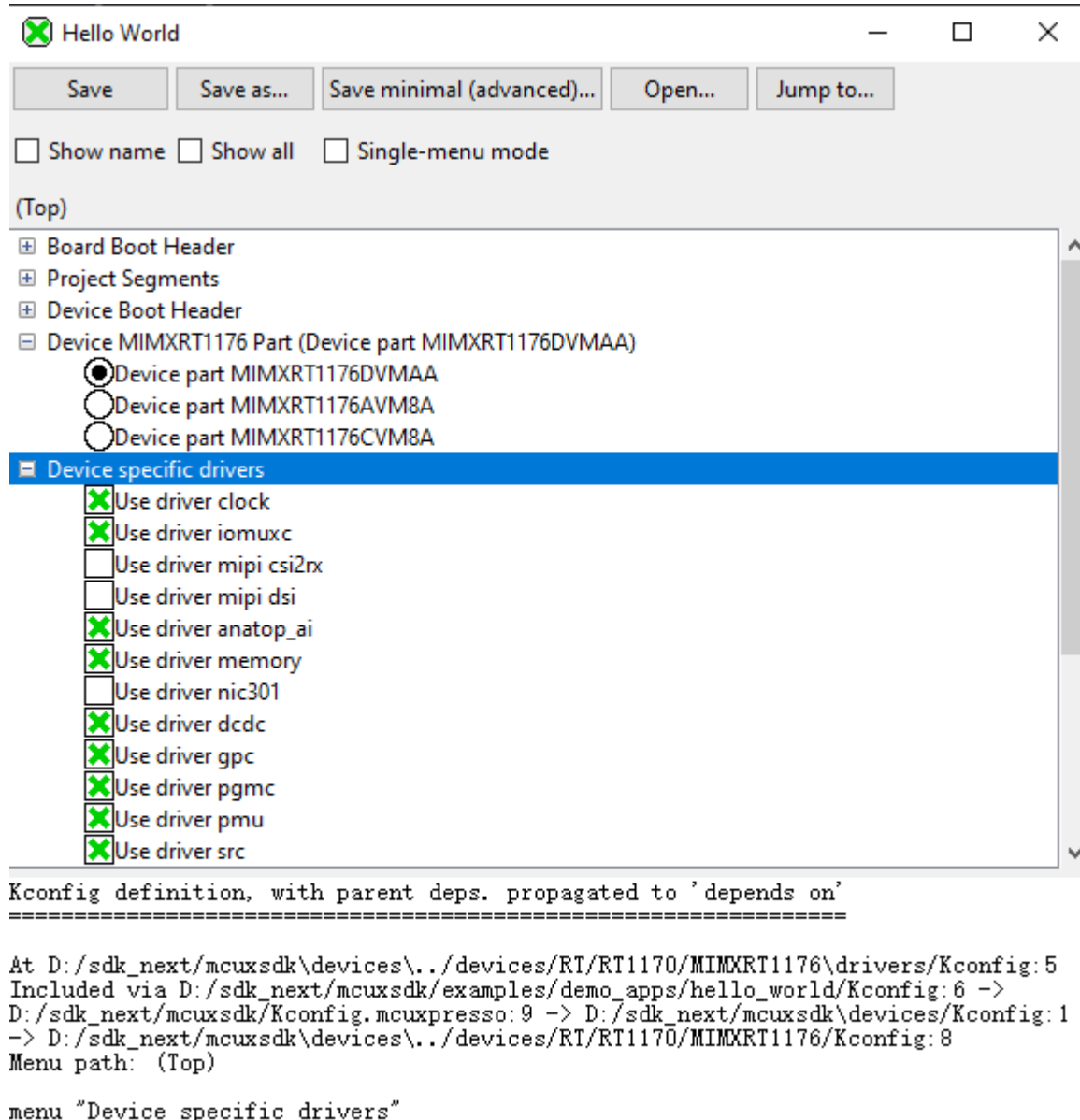
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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



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:

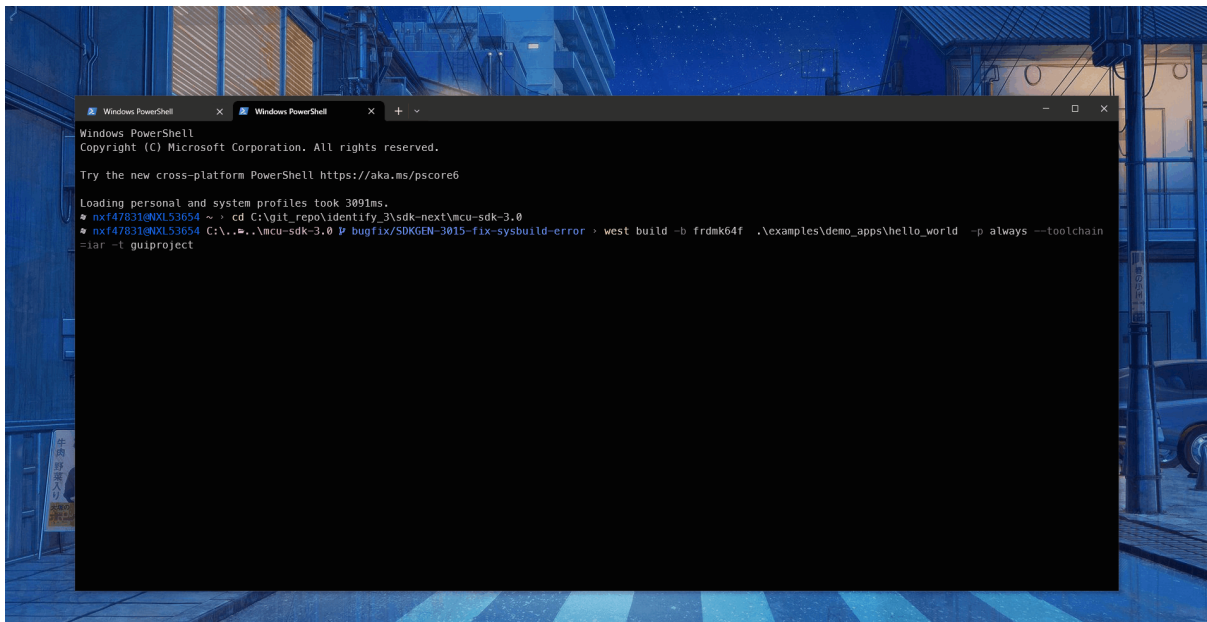
```
west debug -r linkserver
```

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.

- IAR Embedded Workbench for Arm, version is 9.60.4

Supported development systems

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

Development boards	MCU devices
FRDM-MCXL255	MCXL255VDF

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.

Middleware

Multicore Multicore Software Development Kit

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Release contents

Provides an overview of the MCUXpresso SDK release package contents and locations.

Deliverable	Location
Boards	INSTALL_DIR/boards
Demo Applications	INSTALL_DIR/boards/<board_name>/demo_apps
Driver Examples	INSTALL_DIR/boards/<board_name>/driver_examples
eiQ examples	INSTALL_DIR/boards/<board_name>/eiq_examples
Board Project Template for MCUXpresso IDE NPW	INSTALL_DIR/boards/<board_name>/project_template
Driver, SoC header files, extension header files and feature header files, utilities	INSTALL_DIR/devices/<device_name>
CMSIS drivers	INSTALL_DIR/devices/<device_name>/cmsis_drivers
Peripheral drivers	INSTALL_DIR/devices/<device_name>/drivers
Toolchain linker files and startup code	INSTALL_DIR/devices/<device_name>/<toolchain_name>
Utilities such as debug console	INSTALL_DIR/devices/<device_name>/utilities
Device Project Template for MCUXpresso IDE NPW	INSTALL_DIR/devices/<device_name>/project_template
CMSIS Arm Cortex-M header files, DSP library source	INSTALL_DIR/CMSIS
Components and board device drivers	INSTALL_DIR/components
RTOS	INSTALL_DIR/rtos
Release Notes, Getting Started Document and other documents	INSTALL_DIR/docs
Tools such as shared cmake files	INSTALL_DIR/tools
Middleware	INSTALL_DIR/middleware

Known Issues

This section lists the known issues, limitations, and/or workarounds.

KPP Examples have a problem with keypad switch bounces.

OStimer The example doesn't work because it depends on power mode which is not supported by its driver jet.

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
-

AOI

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

AON_LPADC

[2.0.1]

- Improvements
 - Split the ADC_Enable from the ADC_Init function.
 - Rename autoChaIncEn to autoChanIncEn.
- Bug Fixes
 - Fix ADC_SetCmdConfig function cannot correctly configure command.

[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.
-

MCX_CMC**[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

[2.6.0]

- Bug Fixes
 - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

- Improvements
 - Enable irq's that mount under `irqsteer` interrupt extender.

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.3]

- New Features
 - Added `NETC` into status group.

[2.3.2]

- Improvements
 - Make driver `aarch64` compatible

[2.3.1]

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

[2.3.0]

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

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Other Change
 - Added MECC status group definition.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.4]

- New Features
 - Added OTFAD into status group.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
 - Deleted and optimized repeated macro.

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

CRC

[2.0.4]

- 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.
-

DMAMUX**[2.1.2]**

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

[2.1.1]

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

[2.1.0]

- Improvements
 - Modify the type of parameter source from `uint32_t` to `int32_t` in the `DMA-MUX_SetSource`.

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

EDMA

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- Improvements
 - Modify the structures `edma_core_mp_t`, `edma_core_channel_t`, `edma_core_tcd_t` to adapt to edma5.
 - Add TCD register macro to facilitate confirmation of tcd type.
 - Modify the mask macro to a fixed value.
 - Add `EDMA_TCD_TYPE` macro to determine edma tcd type.
 - Add extension API to the following API to determine edma tcd type.
 - * `EDMA_ConfigChannelSoftwareTCD` -> `EDMA_ConfigChannelSoftwareTCDExt`
 - * `EDMA_TcdReset` -> `EDMA_TcdResetExt`

- * EDMA_TcdSetTransferConfig -> EDMA_TcdSetTransferConfigExt
- * EDMA_TcdSetMinorOffsetConfig -> EDMA_TcdSetMinorOffsetConfigExt
- * EDMA_TcdSetChannelLink -> EDMA_TcdSetChannelLinkExt
- * EDMA_TcdSetBandWidth -> EDMA_TcdSetBandWidthExt
- * EDMA_TcdSetModulo -> EDMA_TcdSetModuloExt
- * EDMA_TcdEnableAutoStopRequest -> EDMA_TcdEnableAutoStopRequestExt
- * EDMA_TcdEnableInterrupts -> EDMA_TcdEnableInterruptsExt
- * EDMA_TcdDisableInterrupts -> EDMA_TcdDisableInterruptsExt
- * EDMA_TcdSetMajorOffsetConfig -> EDMA_TcdSetMajorOffsetConfigExt

[2.9.2]

- Improvements
 - Remove tcd alignment check in API that is low level and does not necessarily use 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 avoid DONE bit cleared unexpectedly.

[2.4.0]

- Improvements
 - Added api EDMA_EnableContinuousChannelLinkMode to support continuous link mode.
 - Added apis EDMA_SetMajorOffsetConfig/EDMA_TcdSetMajorOffsetConfig to support major loop address offset feature.
 - Added api EDMA_EnableChannelMinorLoopMapping for minor loop offset feature.
 - Removed the redundant IRQ Handler in edma driver.

[2.3.2]

- Improvements
 - Fixed HIS ccm issue in function EDMA_PrepareTransferConfig.
 - Fixed violations of MISRA C-2012 rule 11.6, 10.7, 10.3, 18.1.
- Bug Fixes

- Added ACTIVE & BITER & CITER bitfields to determine the channel status to fixed the issue of the transfer request cannot submit by function EDMA_SubmitTransfer when channel is idle.

[2.3.1]

- Improvements
 - Added source/destination address alignment check.
 - Added driver IRQ handler support for multi DMA instance in one SOC.

[2.3.0]

- Improvements
 - Added new api EDMA_PrepareTransferConfig to allow different configurations of width and offset.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.4, 10.1.
 - Fixed the Coverity issue regarding out-of-bounds write.

[2.2.0]

- Improvements
 - Added peripheral-to-peripheral support in EDMA driver.

[2.1.9]

- Bug Fixes
 - Fixed MISRA issue: Rule 10.7 and 10.8 in function EDMA_DisableChannelInterrupts and EDMA_SubmitTransfer.
 - Fixed MISRA issue: Rule 10.7 in function EDMA_EnableAsyncRequest.

[2.1.8]

- Bug Fixes
 - Fixed incorrect channel preemption base address used in EDMA_SetChannelPreemptionConfig API which causes incorrect configuration of the channel preemption register.

[2.1.7]

- Bug Fixes
 - Fixed incorrect transfer size setting.
 - * Added 8 bytes transfer configuration and feature for RT series;
 - * Added feature to support 16 bytes transfer for Kinetis.
 - Fixed the issue that EDMA_HandleIRQ would go to incorrect branch when TCD was not used and callback function not registered.

[2.1.6]

- Bug Fixes
 - Fixed KW3X MISRA Issue.
 - * Rule 14.4, 10.8, 10.4, 10.7, 10.1, 10.3, 13.5, and 13.2.
- Improvements
 - Cleared the IRQ handler unavailable for specific platform with macro `FSL_FEATURE_EDMA_MODULE_CHANNEL_IRQ_ENTRY_SHARED_OFFSET`.

[2.1.5]

- Improvements
 - Improved EDMA IRQ handler to support half interrupt feature.

[2.1.4]

- Bug Fixes
 - Cleared enabled request, status during `EDMA_Init` for the case that EDMA is halted before reinitialization.

[2.1.3]

- Bug Fixes
 - Added clear DONE bit in IRQ handler to avoid overwrite TCD issue.
 - Optimized above solution for the case that transfer request occurs in callback.

[2.1.2]

- Improvements
 - Added interface to get next TCD address.
 - Added interface to get the unused TCD number.

[2.1.1]

- Improvements
 - Added documentation for eDMA data flow when scatter/gather is implemented for the `EDMA_HandleIRQ` API.
 - Updated and corrected some related comments in the `EDMA_HandleIRQ` API and `edma_handle_t` struct.

[2.1.0]

- Improvements
 - Changed the `EDMA_GetRemainingBytes` API into `EDMA_GetRemainingMajorLoopCount` due to eDMA IP limitation (see API comments/note for further details).

[2.0.5]

- Improvements
 - Added pubweak DriverIRQHandler for K32H844P (16 channels shared).

[2.0.4]

- Improvements
 - Added support for SoCs with multiple eDMA instances.
 - Added pubweak DriverIRQHandler for KL28T DMA1 and MCIMX7U5_M4.

[2.0.3]

- Bug Fixes
 - Fixed the incorrect pubweak IRQHandler name issue, which caused re-definition build errors when client set his/her own IRQHandler, by changing the 32-channel IRQHandler name to DriverIRQHandler.

[2.0.2]

- Bug Fixes
 - Fixed incorrect minorLoopBytes type definition in `_edma_transfer_config` struct, and defined minorLoopBytes as `uint32_t` instead of `uint16_t`.

[2.0.1]

- Bug Fixes
 - Fixed the eDMA callback issue (which did not check valid status) in `EDMA_HandleIRQ` API.

[2.0.0]

- Initial version.
-

EQDC**[2.3.1]**

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

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.
-

GLIKEY**[2.0.1]**

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

[2.0.0]

- Initial version.
-

GPIO**[2.8.2]**

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

[2.8.1]

- Bug Fixes
 - Fixed CERT INT31-C issues.

[2.8.0]

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

[2.8.0]

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

[2.7.3]

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

[2.7.2]

- New Features
 - Support devices without PORT module.

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

- Bug Fixes
 - Fixed wrong macro definition.
 - Fixed MISRA C-2012 rule issues in the FGPIO_CheckAttributeBytes() function.
 - Defined the new macro to separate the scene when the width of registers is different.
 - Removed some redundant macros.
- New Features

- Added some APIs to get/clear the interrupt status flag when the port doesn't control pins' interrupt.

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
 - Removed deprecated APIs.

[2.3.0]

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

[2.2.1]

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

[2.1.1]

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

[2.1.0]

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

INPUTMUX

[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.
-

KPP**[2.0.1]**

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

[2.0.0]

- Initial version.
-

LPADC**[2.9.3]**

- 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.
-

LPI2C**[2.6.1]**

- Bug Fixes
 - Fixed coverity issues.

[2.6.0]

- New Feature
 - Added common IRQ handler entry LPI2C_DriverIRQHandler.

[2.5.7]

- Improvements
 - Added support for separated IRQ handlers.

[2.5.6]

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

[2.5.5]

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

[2.5.4]

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

[2.5.3]

- Improvements
 - Added handler for LPI2C7 and LPI2C8.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
 - Updated LPI2C_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes
 - Fixed bug in LPI2C_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
 - Fixed bug in LPI2C_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.
 - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
 - Fixed bug in the usage of LPI2C_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.

- Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
- Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

[2.3.0]

- New Features

- Supported reading more than 256 bytes of data in one transfer as master.
- Added API LPI2C_GetInstance.

- Bug Fixes

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

[2.2.0]

- Bug Fixes

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

[2.1.12]

- Bug Fixes

- Fixed MISRA advisory 15.5 issues.

[2.1.11]

- Bug Fixes

- Fixed the bug that, during master non-blocking transfer, after the last byte is sent/received, the kLPI2C_MasterNackDetectFlag is expected, so master should not check and clear kLPI2C_MasterNackDetectFlag when remainingBytes is zero, in case FIFO is emptied when stop command has not been sent yet.

- Fixed the bug that, during non-blocking transfer slave may nack master while master is busy filling tx FIFO, and NDF may not be handled properly.

[2.1.10]

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

[2.1.9]

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

[2.1.8]

- Bug Fixes
 - Fixed the transfer issue with LPI2C_MasterTransferNonBlocking, kLPI2C_TransferNoStopFlag, with the wait transfer done through callback in a way of not doing a blocking transfer.
 - Fixed the issue that STOP signal did not appear in the bus when NAK event occurred.

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

LPI2C_EDMA

[2.4.4]

- Improvements
 - Added support for 2KB data transfer

[2.4.3]

- Improvements
 - Added support for separated IRQ handlers.

[2.4.2]

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

[2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1
-

LPSPI**[2.7.2]**

- Bug Fixes
 - Fixed coverity issues.

[2.7.1]

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

[2.7.0]

- New Feature
 - Added common IRQ handler entry LPSPI_DriverIRQHandler.

[2.6.10]

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

[2.6.9]

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

[2.6.8]

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

[2.6.7]

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

[2.6.6]

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

[2.6.5]

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

[2.6.4]

- Bug Fixes
 - Added LPSPI6_DriverIRQHandler for LPSPI6 instance.

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

LPSPi_EDMA

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

- Improvements
 - Add the TXMSK wait after TCR setting.

[2.4.0]

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

LPTMR**[2.2.1]**

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

LPUART

[2.9.1]

- Bug Fixes
 - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.7]

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

[2.7.6]

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

[2.7.5]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Bug Fixes
 - Added missing interrupt enable masks kLPUART_Match1InterruptEnable and kLPUART_Match2InterruptEnable.
 - Fixed bug in LPUART_EnableInterrupts, LPUART_DisableInterrupts and LPUART_GetEnabledInterrupts that the BAUD[LBKDIE] bit field should be soc specific.
 - Fixed bug in LPUART_TransferHandleIRQ that idle line interrupt should be disabled when rx data size is zero.
 - Deleted unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag, since firstly their function are the same as kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag, secondly to obtain them one data word must be read out thus interfering with the receiving process.
 - Fixed bug in LPUART_GetStatusFlags that the STAT[LBKDIF], STAT[MA1F] and STAT[MA2F] should be soc specific.
 - Fixed bug in LPUART_ClearStatusFlags that tx/rx FIFO is reset by mistake when clearing flags.
 - Fixed bug in LPUART_TransferHandleIRQ that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
 - Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
 - Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
 - Added APIs LPUART_GetRxFifoCount/LPUART_GetTxFifoCount to get rx/tx FIFO data count.
 - Added APIs LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
 - Fixed MISRA advisory 17.7 issues.

[2.4.0]

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

[2.3.1]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.3.0]

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

[2.2.8]

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

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

- Improvements
 - Added hardware flow control function support.
 - Added idle-line-detecting feature in LPUART_TransferNonBlocking function. If an idle line is detected, a callback is triggered with status kStatus_LPUART_IdleLineDetected returned. This feature may be useful when the received Bytes is less than the expected received data size. Before triggering the callback, data in the FIFO (if has FIFO) is read out, and no interrupt will be disabled, except for that the receive data size reaches 0.
 - Enabled the RX FIFO watermark function. With the idle-line-detecting feature enabled, users can set the watermark value to whatever you want (should be less than the RX FIFO size). Data is received and a callback will be triggered when data receive ends.

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
 - Update transactional APIs.
-

LPUART_EDMA**[2.4.0]**

- Refer LPUART driver change log 2.1.0 to 2.4.0
-

MU**[2.7.0]**

- New Features
 - Added API MU_GetRxStatusFlags.

[2.6.0]

- New Features
 - Added API MU_GetInterruptsPending.

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
 - Fixed MISRA C-2012 issues.

[2.1.0]

- Improvements
 - Added new enum `mu_msg_reg_index_t`.

[2.0.0]

- Initial version.
-

OSTIMER**[2.2.4]**

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.2.3]

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

[2.2.2]

- Improvements
 - Support devices with different OSTIMER instance name.

[2.2.1]

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

[2.2.0]

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

[2.1.2]

- Bug Fixes
 - Fixed MISRA-2012 rule 10.8.

[2.1.1]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

- Improvements
 - Added support for OSTIMER0

[2.0.1]

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

[2.0.0]

- Initial version.
-

PORT

[2.5.1]

- Bug Fixes
 - Fix CERT INT31-C issues.

[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.
-

PWM

[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_SetFilterSampleCount to set number of consecutive samples that must agree prior to the input filter.
 - Added API PWM_SetFilterSamplePeriod to 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_SetOutputToIdle 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 with a special phase shift and 50% duty cycle.
 - Added API PWM_SetVALxValue/PWM_GetVALxValue to set/get PWM VALs registers values directly.

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

QTMR**[2.2.2]**

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

SLCD

[2.0.0]

- Initial version.
-

SMM

[2.0.1]

- Bug Fixes
 - Added missed “ of FSL_COMPONENT_ID

[2.0.0]

- Initial version.
-

TRDC

[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.
-

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.
-

WUU**[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

[2.1.9]

- 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 WWWDAT 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 Currently supported drivers

- clock
- reset
- power (Limited. Currently supports only PD1, PD2 and DPD2.)
- lptmr
- qtmr
- lpi2c
- lpspi
- lpuart

- gpio
- kpp
- acmp (Driver and examples are named lpcmp. lpcmp periph. is not supported in this release)
- aon_lpadc
- flash (flash iap)
- mu
- edma
- slcd
- inputmux
- ostimer
- glikey
- utick
- crc

1.7 Driver API Reference Manual

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

[MCXL255](#)

1.8 Middleware Documentation

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

1.8.1 Multicore

[Multicore SDK](#)

1.8.2 FreeRTOS

[FreeRTOS](#)

Chapter 2

MCXL255

2.1 AON_LPADC: 12-bit single-ended SAR Analog-to-Digital Converter Driver

void ADC_GetDefaultConfig(*adc_config_t* *config)

Gets an available pre-defined settings for ADC configuration.

Parameters

- config – Pointer to ADC configuration structure, *adc_config_t*.

void ADC_Init(LPADC_Type *base, const *adc_config_t* *config)

Initializes the ADC module.

Parameters

- base – ADC peripheral base address.
- config – Pointer to configuration structure, *adc_config_t*.

void ADC_Deinit(LPADC_Type *base)

De-initializes the ADC module.

Parameters

- base – ADC peripheral base address.

static inline void ADC_Enable(LPADC_Type *base, bool enable)

Enable/disable the ADC module.

Parameters

- base – ADC peripheral base address.
- enable – Indicates whether to enable the ADC module. - true Enable the ADC module. - false Disable the ADC module.

static inline void ADC_DoFifoRst(LPADC_Type *base, *adc_fifo_index_t* index)

Reset ADC FIFO.

Parameters

- base – ADC peripheral base address.
- index – ADC FIFO index, *adc_fifo_index_t*

static inline void ADC_DoSoftWareRst(LPADC_Type *base)

Software reset ADC module.

Parameters

- base – ADC peripheral base address.

static inline uint32_t ADC_GetStatusFlags(LPADC_Type *base)

Get ADC status flags.

Parameters

- base – ADC peripheral base address.

Returns

Mask value for ADC status flags to be got, should be the OR'ed value of _adc_status_flags.

static inline void ADC_ClearStatusFlags(LPADC_Type *base, uint32_t mask)

Clear ADC status flags.

Note: Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – ADC peripheral base address.
- mask – Mask value for ADC status flags to be cleared, should be the OR'ed value of _adc_status_flags.

static inline uint32_t ADC_GetTriggerStatusFlags(LPADC_Type *base)

Get ADC trigger status flags.

Parameters

- base – ADC peripheral base address.

Returns

Mask value for ADC trigger status flags to be got, should be the OR'ed value of _adc_trig_status_flags.

static inline void ADC_ClearTriggerStatusFlags(LPADC_Type *base, uint32_t mask)

Clear ADC trigger status flags.

Parameters

- base – ADC peripheral base address.
- mask – Mask value for ADC trigger status flags to be cleared, should be the OR'ed value of _adc_trig_status_flags.

static inline void ADC_EnableInt(LPADC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – ADC peripheral base address.
- mask – Mask value for interrupt events to be enabled, should be the OR'ed value of _adc_int_en.

static inline void ADC_DisableInt(LPADC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – ADC peripheral base address.

- **mask** – Mask value for interrupt events to be disabled, should be the OR'ed value of `_adc_int_en`.

static inline void ADC__EnableFifoWatermarkDma(LPADC_Type *base, *adc_fifo_index_t* index, bool enable)

Enable/disable the specified ADC FIFO watermark DAM transfer.

Parameters

- **base** – ADC peripheral base address.
- **index** – ADC FIFO index, `adc_fifo_index_t`
- **enable** – Indicates whether to enable the specified ADC FIFO watermark DAM transfer. **true** Enable the specified ADC FIFO watermark DMA transfer. **false** Disable the specified ADC FIFO watermark DMA transfer.

void ADC__GetDefaultTrigConfig(*adc_trig_config_t* *config)

Gets an available pre-defined settings for ADC trigger's configuration.

Parameters

- **config** – Pointer to ADC trigger configuration structure, `adc_trig_config_t`.

void ADC__SetTrigConfig(LPADC_Type *base, *adc_trig_index_t* index, const *adc_trig_config_t* *config)

Configure the ADC trigger source.

Parameters

- **base** – ADC peripheral base address.
- **index** – ADC trigger index, `adc_trig_index_t`.
- **config** – Pointer to ADC trigger configuration structure, `adc_trig_config_t`.

static inline void ADC__DoSoftwareTrig(LPADC_Type *base, *adc_trig_index_t* index)

Do software trigger.

Parameters

- **base** – ADC peripheral base address.
- **index** – ADC trigger index, `adc_trig_index_t`.

static inline uint32_t ADC__GetResFifoCnt(LPADC_Type *base, *adc_fifo_index_t* index)

Get the count of result kept in specified ADC conversion result FIFO.

Parameters

- **base** – ADC peripheral base address.
- **index** – ADC FIFO index, `adc_fifo_index_t`

Returns

The count of result kept in specified ADC conversion result FIFO.

bool ADC__GetConvRes(LPADC_Type *base, *adc_conv_res_t* *result, *adc_fifo_index_t* index)

Get the result in specified conversion FIFO.

Parameters

- **base** – ADC peripheral base address.
- **result** – Pointer to ADC result structure, `adc_conv_res_t`.
- **index** – ADC FIFO index, `adc_fifo_index_t`

Returns

Status whether FIFO entry is valid. - true Conversion FIFO result is valid. - false Conversion FIFO result is invalid.

void ADC_GetDefaultCmdConfig(*adc_cmd_config_t* *config)

Gets an available pre-defined settings for ADC command's configuration.

Parameters

- config – Pointer to ADC command configuration structure, *adc_cmd_config_t*.

void ADC_SetCmdConfig(LPADC_Type *base, *adc_cmd_index_t* index, const *adc_cmd_config_t* *config)

Do ADC command configuration.

Parameters

- base – ADC peripheral base address.
- index – ADC command index, *adc_cmd_index_t*.
- config – Pointer to ADC command configuration structure, *adc_cmd_config_t*.

static inline void ADC_SetOffsetTrimVal(LPADC_Type *base, uint8_t value)

Set trim value for offset.

Parameters

- base – ADC peripheral base address.
- value – 8-bit unsigned value that should be limited to values between 0h to A0h, with a nominal value of 50h.

static inline void ADC_GetOffsetTrimVal(LPADC_Type *base, uint8_t *pValue)

Get trim value of offset.

Parameters

- base – ADC peripheral base address.
- pValue – Pointer to the variable in type of *uint8_t* to store offset value.

void ADC_DoOffsetCal(LPADC_Type *base)

Do offset calibration.

Parameters

- base – ADC peripheral base address.

FSL_AON_LPADC_DRIVER_VERSION

AON_LPADC driver version 2.0.1.

enum _adc_status_flags

ADC status flags enumeration.

Values:

enumerator kADC_ResFifo0RdyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kADC_ResFifo0OverflowFlag

Indicates that more data has been written to the result FIFO 0 than it can hold.

enumerator kADC_ResFifo1RdyFlag

Indicates when the number of valid datawords in the result FIFO 1 is greater than the setting watermark level.

enumerator kADC_ResFifo1OverflowFlag

Indicates that more data has been written to the result FIFO 1 than it can hold.

enumerator kADC_TrigExcFlag

Indicates that a high priority trigger exception event has occurred.

enumerator kADC_TrigCompFlag

Indicates that a trigger sequence has occurred.

enumerator kADC_CalRdyFlag

Indicates that the calibration process is done.

enumerator kADC_ActFlag

Indicates that the ADC is in active state.

enumerator kADC_TrigAct

Indicates that the trigger is actively being processed.

enumerator kADC_CmdAct

Indicates that the command is actively being processed.

enum _adc_trig_status_flags

ADC trigger status flags enumeration.

Values:

enumerator kADC_Trig0IntFlag

Indicates trigger 0 is interrupted by a high priority exception.

enumerator kADC_Trig1IntFlag

Indicates trigger 1 is interrupted by a high priority exception.

enumerator kADC_Trig2IntFlag

Indicates trigger 2 is interrupted by a high priority exception.

enumerator kADC_Trig3IntFlag

Indicates trigger 3 is interrupted by a high priority exception.

enumerator kADC_Trig0CompFlag

Indicates trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kADC_Trig1CompFlag

Indicates trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kADC_Trig2CompFlag

Indicates trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kADC_Trig3CompFlag

Indicates trigger 3 is completed and trigger 3 has enabled completion interrupts.

enum _adc_int_en

ADC interrupt enablement mask enumeration.

Values:

enumerator kADC_ResFifo0WatermarkIntEn

Configures ADC to generate overflow interrupt requests when FIFO0 ready flag is asserted.

enumerator kADC_ResFifo0OverflowIntEn

Configures ADC to generate overflow interrupt requests when FIFO0 overflow flag is asserted.

enumerator kADC_ResFifo1WatermarkIntEn

Configures ADC to generate overflow interrupt requests when FIFO1 ready flag is asserted.

enumerator kADC_ResFifo1OverflowIntEn

Configures ADC to generate overflow interrupt requests when FIFO1 overflow flag is asserted.

enumerator kADC_TrigExcIntEn

Configures ADC to generate trigger exception interrupt.

enumerator kADC_Trig0CompIntEn

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kADC_Trig1CompIntEn

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kADC_Trig2CompIntEn

Configures ADC to generate interrupt when trigger 2 completion.

enumerator kADC_Trig3CompIntEn

Configures ADC to generate interrupt when trigger 3 completion.

enum _adc_cal_conv_avg

ADC calibration conversion averages enumeration.

Values:

enumerator kADC_CalConvAvg0

Single conversion.

enumerator kADC_CalConvAvg2

2 conversions averaged.

enumerator kADC_CalConvAvg4

4 conversions averaged.

enumerator kADC_CalConvAvg8

8 conversions averaged.

enumerator kADC_CalConvAvg16

16 conversions averaged.

enumerator kADC_CalConvAvg32

32 conversions averaged.

enumerator kADC_CalConvAvg64

64 conversions averaged.

enumerator kADC_CalConvAvg128

128 conversions averaged.

enumerator kADC_CalConvAvg256

256 conversions averaged.

enumerator kADC_CalConvAvg512

512 conversions averaged.

enumerator kADC_CalConvAvg1024

1024 conversions averaged.

enum _adc_cal_conv_sample_time

ADC calibration conversion sample time enumeration.

Values:

enumerator kADC_CalConvSampTime_AdckCycle0

1 ADCK cycles sample time.

enumerator kADC__CalConvSampTime__AdckCycle1
2 ADCK cycles sample time.

enumerator kADC__CalConvSampTime__AdckCycle2
3 ADCK cycles sample time.

enumerator kADC__CalConvSampTime__AdckCycle3
4 ADCK cycles sample time.

enumerator kADC__CalConvSampTime__AdckCycle4
5 ADCK cycles sample time.

enumerator kADC__CalConvSampTime__AdckCycle5
6 ADCK cycles sample time.

enumerator kADC__CalConvSampTime__AdckCycle6
7 ADCK cycles sample time.

enumerator kADC__CalConvSampTime__AdckCycle7
8 ADCK cycles sample time.

enum __adc_fifo_index
ADC FIFO enumeration.

Values:

enumerator kADC__Fifo0
FIFO 0.

enumerator kADC__Fifo1
FIFO 1.

enum __adc_trig_index
ADC trigger enumeration.

Values:

enumerator kADC__Trig0
Trigger 0.

enumerator kADC__Trig1
Trigger 1.

enumerator kADC__Trig2
Trigger 2.

enumerator kADC__Trig3
Trigger 3.

enum __adc_trig_cmd_index
ADC trigger command enumeration.

Values:

enumerator kADC__CmdNone
Not a valid selection from the command buffer, trigger event is ignored.

enumerator kADC__Cmd1
Trigger commad 1.

enumerator kADC__Cmd2
Trigger commad 2.

enumerator kADC_Cmd3

Trigger command 3.

enumerator kADC_Cmd4

Trigger command 4.

enumerator kADC_Cmd5

Trigger command 5.

enumerator kADC_Cmd6

Trigger command 6.

enumerator kADC_Cmd7

Trigger command 7.

enum _adc_conv_avg

ADC hardware conversion averages enumeration.

Values:

enumerator kADC_ConvAvg1

Single conversion.

enumerator kADC_ConvAvg2

2 conversions averaged.

enumerator kADC_ConvAvg4

4 conversions averaged.

enumerator kADC_ConvAvg8

8 conversions averaged.

enumerator kADC_ConvAvg16

16 conversions averaged.

enumerator kADC_ConvAvg32

32 conversions averaged.

enumerator kADC_ConvAvg64

64 conversions averaged.

enumerator kADC_ConvAvg128

128 conversions averaged.

enumerator kADC_ConvAvg256

256 conversions averaged.

enumerator kADC_ConvAvg512

512 conversions averaged.

enumerator kADC_ConvAvg1024

1024 conversions averaged.

enum _adc_sample_time

ADC conversion sample time enumeration.

Values:

enumerator kADC_ConvSampleTime_AdcCycle1

1 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle2

2 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle3
3 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle4
4 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle5
5 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle6
6 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle7
7 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle8
8 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle10
10 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle12
12 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle16
16 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle24
24 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle40
40 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle72
72 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle136
136 ADCK cycles.

enumerator kADC_ConvSampleTime_AdcCycle264
264 ADCK cycles.

enum _adc_comp_mode
ADC hardware compare mode enumeration.

Note: 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 kADC_CompareDisabled
Compare disabled.

enumerator kADC_CompareStoreOnTrue
Compare enabled. Store on true.

enumerator kADC_CompareRepeatUntilTrue
Compare enabled. Repeat channel acquisition until true.

enum _adc_ref_voltage_src

ADC reference voltage source enumeration.

Values:

enumerator kADC_RefVoltageAlt1

Option 1 setting.

enumerator kADC_RefVoltageAlt2

Option 2 setting.

enum _adc_dac_settle_delay

ADC DAC settle time enumeration.

Values:

enumerator kADC_FastDacSettleTime

Fastest settling

enumerator kADC_SlowDacSettleTime

Slowest settling

enum _adc_trig_priority_policy

ADC trigger priority exception handle policy enumeration.

Values:

enumerator kADC_ConvPreemptImmedNotAutoResumed

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 kADC_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 kADC_ConvPreemptImmedAutoRestarted

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 kADC_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 kADC_ConvPreemptImmedAutoResumed

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 kADC_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 `kADC_ConvPreemptSubsequentlyNotAutoResumed`

If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger; when higher priority conversion finishes, the preempted conversion will not automatically be restarted or resumed.

enumerator `kADC_ConvPreemptSubsequentlyAutoRestarted`

If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger; when higher priority conversion finishes, the preempted conversion will be automatically restarted.

enumerator `kADC_ConvPreemptSubsequentlyAutoResumed`

If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger; when higher priority conversion finishes, the preempted conversion will be automatically resumed.

enumerator `kADC_TriggerPriorityExceptionDisabled`

High priority trigger exception disabled.

typedef enum `_adc_cal_conv_avg` `adc_cal_conv_avg_t`

ADC calibration conversion averages enumeration.

typedef enum `_adc_cal_conv_sample_time` `adc_cal_conv_sample_time_t`

ADC calibration conversion sample time enumeration.

typedef enum `_adc_fifo_index` `adc_fifo_index_t`

ADC FIFO enumeration.

typedef enum `_adc_trig_index` `adc_trig_index_t`

ADC trigger enumeration.

typedef enum `_adc_trig_cmd_index` `adc_cmd_index_t`

ADC trigger command enumeration.

typedef enum `_adc_conv_avg` `adc_conv_avg_t`

ADC hardware conversion averages enumeration.

typedef enum `_adc_sample_time` `adc_conv_sample_time_t`

ADC conversion sample time enumeration.

typedef enum `_adc_comp_mode` `adc_comp_mode_t`

ADC hardware compare mode enumeration.

Note: 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 `_adc_ref_voltage_src` `adc_ref_voltage_src_t`

ADC reference voltage source enumeration.

typedef enum `_adc_dac_settle_delay` `adc_dac_settle_delay_t`

ADC DAC settle time enumeration.

```
typedef enum _adc_trig_priority_policy adc_trig_priority_policy_t
```

ADC trigger priority exception handle policy enumeration.

```
typedef struct _adc_conv_res adc_conv_res_t
```

ADC conversion result structure.

```
typedef struct _adc_trig_config adc_trig_config_t
```

ADC trigger configuration structure.

```
typedef struct _adc_cmd_config adc_cmd_config_t
```

ADC command configuration structure.

```
typedef struct _adc_config adc_config_t
```

ADC configuration structure.

```
ADC_TRIG_PRIORITY_EXC_HANDLE_POLICY_EN(tprictrl, tres, tcmdres, hpted)
```

Trigger priority exception handle policy helper macro.

tprictrl: ADC Trigger Priority Control. **00b** - 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. **01b** - If a higher priority trigger is received during command processing, the current command is stopped after completing the current conversion. If averaging is enabled, the averaging loop will be completed. However, CMDHa[LOOP] will be ignored and the higher priority trigger will be serviced. **10b** - If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger.

tres: Trigger Resume Enable. **0b** - Trigger sequences interrupted by a high priority trigger exception are not automatically resumed or restarted. **1b** - Trigger sequences interrupted by a high priority trigger exception are automatically resumed or restarted.

tcmdres: Trigger Command Resume, tres must be asserted for tcmdres to be used. **0b** - Trigger sequences interrupted by a high priority trigger exception is automatically restarted. **1b** - Trigger sequences interrupted by a high priority trigger exception is resumed from the command executing before the exception.

hpted: High Priority Trigger Exception Disable **0b** - High priority trigger exceptions are enabled. **1b** - High priority trigger exceptions are disabled.

```
ADC_TRIG_PRIORITY_EXC_HANDLE_POLICY_DE(policy)
```

```
struct _adc_conv_res
```

#include <fsl_aon_lpadc.h> ADC conversion result structure.

Public Members

```
uint8_t trigSrc
```

Indicate the trigger source that initiated a conversion and generated this result.

```
uint8_t cmdSrc
```

Indicate the command buffer being executed that generated this result.

```
uint8_t loopCnt
```

Indicate the loop count value during command execution that generated this result.

```
uint16_t convVal
```

ADC conversion result.

```
struct _adc_trig_config
```

#include <fsl_aon_lpadc.h> ADC trigger configuration structure.

Public Members

bool trigEn

Controls hardware trigger source to initiate conversion on the rising edge of the input trigger source.

uint8_t trigDelay

Sets the trigger delay duration to wait at the start of servicing a trigger event.

uint8_t trigPriority

Sets the priority of the associated trigger source.

adc_fifo_index_t resFifo

Selects the FIFO to store the conversion result.

adc_cmd_index_t cmdIndex

Selects the trigger command from command buffer to execute upon detect of the associated trigger event.

struct __adc_cmd_config

#include <fsl_aon_lpadc.h> ADC command configuration structure.

Public Members

bool waitTrigEn

Controls whether commands are automatically executed or a trigger must be received before execution.

bool autoChanIncEn

Controls whether enable automatic channel incrementing.

uint8_t loopCnt

Sets how many times this command executes (and stores conversion result to RESFIFO) before finish and transition to the next command or Idle state.

uint8_t chanIndex

Sets conversion channel.

uint16_t compValLow

Sets compare low value.

uint16_t compValHigh

Sets compare high value.

adc_conv_avg_t convAvg

Selects hardware average value.

adc_comp_mode_t compMode

Selects hardware compare mode.

adc_cmd_index_t nextCmdIndex

Selects the next command to execute after this command completes.

adc_conv_sample_time_t convSampleTime

Selects sample time value.

struct __adc_config

#include <fsl_aon_lpadc.h> ADC configuration structure.

Public Members

- bool** dozeModeEn
Controls system transition to low power modes while ADC is converting.
- bool** convPauseEn
Controls whether to enable the ADC pause function.
- bool** dynamicEleMatchEn
Controls whether to enable dynamic element match.
- uint8_t** compTime
Sets a configurable number of ADCK cycles to complete the compare phase of a conversion.
- uint8_t** convPauseDelay
Sets the duration of pausing during command execution sequencing.
- uint8_t** fifo0Watermark
Sets FIFO 0 watermark value.
- adc_cal_conv_avg_t** calConvAvg
Selects calibration conversion averages.
- adc_ref_voltage_src_t** refVoltageSrc
Selects the reference voltage source.
- adc_dac_settle_delay_t** dacSettleTime
Selects DAC settle delay.
- adc_cal_conv_sample_time_t** calConvSampTime
Selects calibration conversion sample time.
- adc_trig_priority_policy_t** trigPriorityPolicy
Selects trigger priority exception handle policy.

2.2 Clock Driver

- enum** _clock_ip_name
Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
Values:
- enumerator** kCLOCK_InputMux
Clock gate name: AON INPUTMUX (only)
- enumerator** kCLOCK_GateAonUART
Clock gate name: AON UART
- enumerator** kCLOCK_GateAonI2C
Clock gate name: AON I2C
- enumerator** kCLOCK_GateAonAPB
Clock gate name: AON APB
- enumerator** kCLOCK_GateAonPORT
Clock gate name: AON PORT
- enumerator** kCLOCK_GateAonGPIO
Clock gate name: AON GPIO

enumerator kCLOCK_GateAonQTMR0
Clock gate name: AON QTMR0

enumerator kCLOCK_GateAonQTMR1
Clock gate name: AON QTMR1

enumerator kCLOCK_GateAonLPTMR
Clock gate name: AON LPTMR

enumerator kCLOCK_GateAonKPP
Clock gate name: AON KPP

enumerator kCLOCK_GateAonLPADC
Clock gate name: AON LPADC

enumerator kCLOCK_GateAonSYS
Clock gate name: AON SYS (tick)

enumerator kCLOCK_GateAonACMP0
Clock gate name: AON comparator

enumerator kCLOCK_GateAonLCD
Clock gate name: AON LCD

enumerator kCLOCK_GateAonAVDC2P0
Clock gate name: AON AVDC2P0

enumerator kCLOCK_GateAonINPUTMUX1
Clock gate name: AON INPUTMUX

enumerator kCLOCK_GateNotAvail
Clock gate name: None

enum _clock_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_RtcOscClk
RTC OSC clock.

enumerator kCLOCK_Fro16k
AON PAC and SMM clock.

enumerator kCLOCK_FroAON
AON free runnin clock.

enum _clock_select_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCKC_SelAonROOT_AUX
AON ROOT_AUX clock selection

enumerator kCLOCKC_SelAonROOT
AON ROOT clock selection

enumerator kCLOCKC_SelAonACMP0
AON ACMP0 clock selection

enumerator kCLOCKC_SelAonLCD
AON LCD clock selection

enumerator kCLOCKC_SelAonLPADC

AON LPADC clock selection

enumerator kCLOCKC_SelAonSYSTICK

AON SYS tick clock selection

enumerator kCLOCKC_SelAonKPP

AON KPP clock selection

enumerator kCLOCKC_SelAonLPTMR

AON LPTMR GRP clock selection

enumerator kCLOCKC_SelAonTMR

AON TMR GRP clock selection

enumerator kCLOCKC_SelAonCOM

AON COM comaprator (aon_per_clk) clock selection

enum _clock_attach_id

The enumerator of clock attach Id.

Values:

enumerator kXTAL32K_to_AON_ROOT_AUX

Attach XTAL32K to AON AUX.

enumerator kAUX_to_AON_ROOT_AUX

Attach AUX to AON AUX.

enumerator kFROdiv1_to_AON_CPU

Attach FRO div 1 to AON_CPU.

enumerator kFROdiv2_to_AON_CPU

Attach FRO div 2 to AON_CPU.

enumerator kFROdiv4_to_AON_CPU

Attach FRO div 4 to AON_CPU.

enumerator kROOT_AUX_to_AON_CPU

Attach ROOT AUX to AON_CPU.

enumerator kFROdiv1_to_AON_CMP0

Attach FRO div 1 to AON ACMP0.

enumerator kFROdiv2_to_AON_CMP0

Attach FRO div 2 to AON ACMP0.

enumerator kFROdiv4_to_AON_CMP0

Attach FRO div 4 to AON ACMP0.

enumerator kROOT_AUX_to_AON_CMP0

Attach ROOT AUX to AON ACMP0.

enumerator kCLK_16K_to_AON_LCD

Attach FRO clk_16k to AON LCD.

enumerator kFRO16K_to_AON_LCD

Attach FRO fro16k to AON LCD.

enumerator kFROdiv1_to_AON_LPADC

Attach FRO div 1 to AON LPADC.

enumerator kFROdiv2_to_AON_LPADC
Attach FRO div 2 to AON LPADC.

enumerator kFROdiv4_to_AON_LPADC
Attach FRO div 4 to AON LPADC.

enumerator kROOT_AUX_to_AON_LPADC
Attach ROOT AUX to AON LPADC.

enumerator kXTAL32K_to_AON_LPADC
Attach FRO RTC to AON LPADC.

enumerator kFRO16K_to_AON_LPADC
Attach FRO fro16k to AON LPADC.

enumerator kFROdiv1_to_AON_SYSTICK
Attach FRO div 1 to AON SYSTICK.

enumerator kFROdiv2_to_AON_SYSTICK
Attach FRO div 2 to AON SYSTICK.

enumerator kFROdiv4_to_AON_SYSTICK
Attach FRO div 4 to AON SYSTICK.

enumerator kROOT_AUX_to_AON_SYSTICK
Attach ROOT AUX to AON SYSTICK.

enumerator kXTAL32K_to_AON_KPP
Attach XTAL32K to AON KPP.

enumerator kFRO16K_to_AON_KPP
Attach FRO fro16k to AON KPP.

enumerator AON_TMR_to_AON_LPTMR
Attach AON TIMER CLK to AON LPTMR GRP.

enumerator kFRO16K_to_AON_LPTMR
Attach FRO16K to AON LPTMR GRP.

enumerator kCLK_16K_to_AON_LPTMR
Attach CLK_16K to AON LPTMR GRP.

enumerator kFROdiv1_to_AON_TMR
Attach FRO div 1 to AON TMR GRP.

enumerator kFROdiv2_to_AON_TMR
Attach FRO div 2 to AON TMR GRP.

enumerator kFROdiv4_to_AON_TMR
Attach FRO div 4 to AON TMR GRP.

enumerator kROOT_AUX_to_AON_TMR
Attach ROOT AUX to AON TMR GRP.

enumerator kFROdiv1_to_AON_COM
Attach FRO div 1 to AON COM GRP.

enumerator kFROdiv2_to_AON_COM
Attach FRO div 2 to AON COM GRP.

enumerator kFROdiv4_to_AON_COM
Attach FRO div 4 to AON COM GRP.

enumerator kROOT_AUX_to_AON_COM

Attach ROOT AUX to AON COM GRP.

enumerator kNONE_to_NONE

Attach NONE to NONE.

enum _clock_div_name

Clock dividers.

Values:

enumerator kCLOCK_DIVAonCPU

Aon CPU clock divider

enumerator kCLOCK_DIVAonCMP

Aon Comp grp clock divider

enumerator kCLOCK_DIVAonSYS

Aon SYSTICK clock divider

enumerator kCLOCK_DIVAonACMP0CLK0

Aon CMP0 CLK0 clock divider

enumerator kCLOCK_DIVAonACMP0CLK1

Aon CMP0 CLK1 clock divider

enumerator kCLOCK_DivMax

MAX clock divider

typedef enum _clock_ip_name clock_ip_name_t

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

typedef enum _clock_name clock_name_t

Clock name used to get clock frequency.

typedef enum _clock_select_name clock_select_name_t

Clock name used to get clock frequency.

typedef enum _clock_attach_id clock_attach_id_t

The enumerator of clock attach Id.

typedef enum _clock_div_name clock_div_name_t

Clock dividers.

typedef struct _aon_fro_trim_config aon_fro_trim_config_t

AON FRO trim configuration.

volatile uint32_t g_xtal32Freq

External XTAL32/EXTAL32 clock frequency.

The XTAL32/EXTAL32 clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal32Freq to set the value in the clock driver.

This is important for the multicore platforms where only one core needs to set up the clock. All other cores need to call the CLOCK_SetXtal32Freq to get a valid clock frequency.

static inline void CLOCK_EnableClock(clock_ip_name_t clk)

Enable the clock for specific IP.

Parameters

- clk – : Clock to be enabled.

Returns

Nothing

static inline void CLOCK_DisableClock(*clock_ip_name_t* clk)

Disable the clock for specific IP.

Parameters

- *clk* – : Clock to be Disabled.

Returns

Nothing

void CLOCK_AttachClk(*clock_attach_id_t* connection)

Configure the clock selection muxes.

Parameters

- *connection* – : Clock to be configured.

Returns

Nothing

clock_attach_id_t CLOCK_GetClockAttachId(*clock_attach_id_t* connection)

Get the actual clock attach id. This function uses the offset in input attach id, then it reads the actual source value in the register and combine the offset to obtain an actual attach id.

Parameters

- *connection* – : Clock attach id to get.

Returns

Clock source value.

void CLOCK_SetClockSelect(*clock_select_name_t* sel_name, uint32_t value)

Set the clock select value. This function set the peripheral clock select value.

Parameters

- *sel_name* – : Clock select.
- *value* – : value to be set.

uint32_t CLOCK_GetClockSelect(*clock_select_name_t* sel_name)

Get the clock select value. This function get the peripheral clock select value.

Parameters

- *sel_name* – : Clock select.

Returns

Clock source value.

void CLOCK_SetClockDiv(*clock_div_name_t* div_name, uint32_t value)

Setup peripheral clock dividers.

Parameters

- *div_name* – : Clock divider name
- *value* – : Value to be divided

Returns

Nothing

uint32_t CLOCK_GetClockDiv(*clock_div_name_t* div_name)

Get peripheral clock dividers.

Parameters

- *div_name* – : Clock divider name

Returns

peripheral clock dividers

void CLOCK_HaltClockDiv(*clock_div_name_t* div_name)

Halt peripheral clock dividers.

Parameters

- div_name – : Clock divider name

Returns

Nothing

status_t CLOCK_SetupFROAonClocking(uint32_t iFreq)

Initialize the AON FRO to given frequency..

Parameters

- iFreq – : Desired frequency (10M,2M,0=off).

Returns

returns success or fail status.

uint32_t CLOCK_GetAonCoreSysClkFreq(void)

Return Frequency of the AON core.

Returns

Frequency of the core

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Return Frequency of selected clock.

Returns

Frequency of selected clock

uint32_t CLOCK_GetLpi2cClkFreq(uint32_t id)

Return Frequency of LPI2C functional Clock.

Returns

Frequency of LPI2C functional Clock

uint32_t CLOCK_GetLpuartClkFreq(uint32_t id)

Return Frequency of LPUART functional Clock.

Returns

Frequency of LPUART functional Clock

void CLOCK_AON_FRO_Trim_Set(uint8_t is_fro2m, *aon_fro_trim_config_t* config)

Sets AON FRO 10M or 2M trim.

Parameters

- is_fro2m – : 0 for FRO10M, 1 for FRO2M
- config – : trim value

void CLOCK_AON_FRO_Trim_Get(uint8_t is_fro2m, *aon_fro_trim_config_t* *config)

Reads AON FRO 10M or 2M trim values.

Parameters

- is_fro2m – : 0 for FRO10M, 1 for FRO2M
- config – : ptr to aon_fro_trim_config_t struct.

static inline void CLOCK_SetXtal32Freq(uint32_t freq)

Sets the XTAL32 frequency based on board settings.

Parameters

- freq – The XTAL32/EXTAL32 input clock frequency in Hz.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 1.1.0.

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL

Configure whether driver controls clock.

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note: All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

CLK_GATE_REG_PR_OFFSET(value)

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

The offset of the registers MRCC_GLB_PR0 and MRCC_GLB_PR1.

CLK_GATE_REG_CC_OFFSET(value)

The offset of the registers MRCC_GLB_CC0 and MRCC_GLB_CC1.

CLK_PERIPHERAL_BIT_SHIFT(value)

Bit definitions for the peripherals in MRCC_GLB_PR and MRCC_GLB_CC.

CLK_OF_AON(value)

True when clock gate belongs to AON domain.

REG_PWM0SUBCTL

SLCD_FAULT_DETECT_CLOCKS

Clock ip name array for SLCD.

SLCD_CONTROL_CLOCKS

LPACMP_CLOCKS

Clock ip name array for LPACMP.

AOI_CLOCKS

Clock ip name array for AOI.

ATX_CLOCKS

Clock ip name array for ATX.

CRC_CLOCKS

Clock ip name array for CRC.

CTIMER_CLOCKS

Clock ip name array for CTIMER.

DMA_CLOCKS

Clock ip name array for DMA.

ERM_CLOCKS

EDMA_CLOCKS

Clock gate name array for EDMA.

EZRAMC_CLOCKS

Clock ip name array for EZRAMC_RAMC.

FREQME_CLOCKS

Clock ip name array for FREQME.

GPIO_CLOCKS

Clock ip name array for GPIO.

INPUTMUX_CLOCKS

Clock ip name array for INPUTMUX.

LPCMP_CLOCKS

Clock ip name array for LPCMP.

AON_LPADC_CLOCKS

Clock ip name array for LPADC.

Clock ip name array for LPADC.

LPUART_CLOCKS

Clock ip name array for LPUART.

LPI2C_CLOCKS

Clock ip name array for LPI2C.

LPSPI_CLOCKS

Clock ip name array for LPSPI.

MTR_CLOCKS

Clock ip name array for MTR.

OSTIMER_CLOCKS

Clock ip name array for OSTIMER.

PERIPHGROUP_CLOCKS

Clock ip name array for PERIPH_GROUP.

PORT_CLOCKS

Clock ip name array for PORT.

ROMCP_CLOCKS

Clock ip name array for ROMCP.

TCU_CLOCKS

Clock ip name array for TCU.

UTICK_CLOCKS

Clock ip name array for UTICK.

WWDT_CLOCKS

Clock ip name array for WWDT.

BUS_CLK

Peripherals clock source definition.

TMR_CLOCKS

Clock ip name array for QTMRs.

KPP_CLOCKS

Clock ip name array for QTMRs.

CLK_ATTACH_REG_OFFSET(value)

Clock Mux Switches The encoding is as follows each connection identified is 32bits wide while 24bits are valuable starting from LSB upwards.

[4 bits for choice, 0 means invalid choice] [8 bits mux ID]*

CLK_ATTACH_CLK_SEL(value)

CLK_ATTACH_MUX(reg, sel)

CLK_OF_AON_SEL(value)

CLK_AON_SEL_REG_OFFSET(value)

CLK_AON_SEL_SHIFT(value)

CLK_AON_SEL_MASK(value)

CLK_AON_SEL(reg, shift, mask)

uint8_t fs_bp

Replica voltage fs_bp. 0-7

uint8_t fs_vcco

Replica voltage fs_vcco. 0-3

uint8_t tf

Relationship between the frequency and temperature. 0-7

uint8_t cltrim

Trim coarse freq value; 0-63

uint8_t ccotrim

Trim fine freq value; 0-63

struct _aon_fro_trim_config

#include <fsl_clock.h> AON FRO trim configuration.

2.3 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 default.

struct __crc_config

#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

CRC Polynomial, MSBit first. Example polynomial: 0x1021 = 1_0000_0010_0001 = $x^{12}+x^5+1$

uint32_t seed

Starting checksum value

bool reflectIn

Reflect bits on input.

`bool reflectOut`

Reflect bits on output.

`bool complementChecksum`

True if the result shall be complement of the actual checksum.

`crc_bits_t crcBits`

Selects 16- or 32- bit CRC protocol.

`crc_result_t crcResult`

Selects final or intermediate checksum return from `CRC_Get16bitResult()` or `CRC_Get32bitResult()`

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

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

Initializes the eDMA peripheral.

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

Note: This function enables the minor loop map feature.

Parameters

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

`void EDMA_Deinit(EDMA_Type *base)`

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- `base` – eDMA peripheral base address.

`void EDMA_InstallTCD(EDMA_Type *base, uint32_t channel, edma_tcd_t *tcd)`

Push content of TCD structure into hardware TCD register.

Parameters

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

`void EDMA_GetDefaultConfig(edma_config_t *config)`

Gets the eDMA default configuration structure.

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

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

Parameters

- config – A pointer to the eDMA configuration structure.

```
void EDMA_InitChannel(EDMA_Type *base, uint32_t channel, edma_channel_config_t
                    *channelConfig)
```

EDMA Channel initialization.

Parameters

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

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

Set channel memory attribute.

Parameters

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

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

Set channel sign extension.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- position – A non-zero value specifying the sign extend bit position. If 0, sign extension is disabled.

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

Set channel swap size.

Parameters

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

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

Set channel access type.

Parameters

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

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

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

Set channel request source.

Parameters

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

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

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

Parameters

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

Returns

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

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

Set channel master ID replication.

Parameters

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

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

Set channel security level.

Parameters

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

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

Sets all TCD registers to default values.

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

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Configures the eDMA transfer attribute.

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

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

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

Parameters

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

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

Configures the eDMA minor offset feature.

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

Parameters

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

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

Configures the eDMA channel preemption feature.

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

Parameters

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


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

Sets the channel link for the eDMA transfer.

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

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

Parameters

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

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

Sets the bandwidth for the eDMA transfer.

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

Parameters

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

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

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

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

Parameters

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

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

Enables an auto stop request for the eDMA transfer.

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

Parameters

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

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

Enables the interrupt source for the eDMA transfer.

Parameters

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

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

Disables the interrupt source for the eDMA transfer.

Parameters

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

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

Configures the eDMA channel TCD major offset feature.

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

Parameters

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

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

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

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

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

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

Note: This function enables the auto stop request feature.

Parameters

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

`void EDMA_TcdReset(edma_tcd_t *tcd)`

Sets all fields to default values for the TCD structure.

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

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

Note: This function enables the auto stop request feature.

Parameters

- `tcd` – Pointer to the TCD structure.

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

Configures the eDMA TCD transfer attribute.

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

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

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

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

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

Parameters

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

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

Configures the eDMA TCD minor offset feature.

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

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

Parameters

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

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

Sets the channel link for the eDMA TCD.

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

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

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

Parameters

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

```
static inline void EDMA__TcdSetBandWidth(edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

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

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

Parameters

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

```
void EDMA__TcdSetModulo(edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo)
```

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

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

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

Parameters

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

```
static inline void EDMA__TcdEnableAutoStopRequest(edma_tcd_t *tcd, bool enable)
```

Sets the auto stop request for the eDMA TCD.

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

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

Parameters

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

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

Enables the interrupt source for the eDMA TCD.

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

Parameters

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

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

Disables the interrupt source for the eDMA TCD.

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

Parameters

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

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

Configures the eDMA TCD major offset feature.

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

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

Parameters

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

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

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

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

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Sets all fields to default values for the TCD structure.

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Configures the eDMA TCD transfer attribute.

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

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

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

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

Parameters

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

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

Configures the eDMA TCD minor offset feature.

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

Parameters

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

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

Sets the channel link for the eDMA TCD.

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

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

Parameters

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

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

Sets the bandwidth for the eDMA TCD.

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

Parameters

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

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

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

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

Parameters

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

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

Sets the auto stop request for the eDMA TCD.

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

Parameters

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

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

Enables the interrupt source for the eDMA TCD.

Parameters

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

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

Disables the interrupt source for the eDMA TCD.

Parameters

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


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

Configures the eDMA TCD major offset feature.

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

Parameters

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

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

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

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

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

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

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

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

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

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

```
uint32_t EDMA__GetRemainingMajorLoopCount(EDMA_Type *base, uint32_t channel)
```

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

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

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

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

Parameters

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

Returns

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

```
static inline uint32_t EDMA_GetErrorStatusFlags(EDMA_Type *base)
```

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

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

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

Gets the eDMA channel status flags.

Parameters

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

Returns

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

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

Clears the eDMA channel status flags.

Parameters

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

```
void 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.

```
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_InstallTCDDMemory(&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_InstallTCDDMemory 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 2.10.6.

_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 position configuration

uint32_t channelRequestSource
hardware service request source for the channel

bool enableMasterIDReplication
enable master ID replication

edma_channel_protection_level_t protectionLevel
protection level

struct _edma_transfer_config
#include <fsl_edma.h> edma4 channel transfer configuration

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

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed 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 - (2³⁰ - 1) when minor loop mapping is enabled range: 1 - (2¹⁰ - 1) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - (2³² - 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 - (2¹⁵ - 1)) when minor loop channel link is disabled range: (0 - (2⁹ - 1)) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajorLoop * bytesEachMinorLoop

uint16_t enabledInterruptMask

channel interrupt to enable, can be OR'ed value of _edma_interrupt_enable

edma_modulo_t srcAddrModulo
source circular data queue range

int32_t srcMajorLoopOffset
source major loop offset

edma_modulo_t dstAddrModulo
destination circular data queue range

int32_t dstMajorLoopOffset
destination major loop offset

bool enableSrcMinorLoopOffset
enable source minor loop offset

bool enableDstMinorLoopOffset
enable dest minor loop offset

int32_t minorLoopOffset
burst offset, the offset will be applied after minor loop update

bool enableChannelMajorLoopLink
channel link when major loop complete

uint32_t majorLoopLinkChannel
major loop link channel number

bool enableChannelMinorLoopLink
channel link when minor loop complete

uint32_t minorLoopLinkChannel
minor loop link channel number

edma_tcd_t *linkTCD
pointer to the link transfer control descriptor

struct _edma_config
#include <fsl_edma.h> eDMA global configuration structure.

Public Members

bool enableMasterIdReplication
Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

bool enableGlobalChannelLink
Enable(true) channel linking is available and controlled by each channel's link settings.

bool enableHaltOnError
Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableDebugMode
Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

bool enableRoundRobinArbitration
Enable(true) channel linking is available and controlled by each channel's link settings.

edma_channel_config_t *channelConfig[1]
channel preemption configuration

```
struct _edma_handle
    #include <fsl_edma.h> eDMA transfer handle structure
```

Public Members

edma_callback callback
Callback function for major count exhausted.

void *userData
Callback function parameter.

EDMA_ChannelType *channelBase
eDMA peripheral channel base address.

EDMA_Type *base
eDMA peripheral base address

EDMA_TCDType *tcdBase
eDMA peripheral tcd base address.

edma_tcd_t *tcdPool
Pointer to memory stored TCDs.

uint32_t channel
eDMA channel number.

volatile int8_t header
The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail
The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed
The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize
The total number of TCD slots in the queue.

2.5 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_MUX_SOURCE(x)
DMA_ERR_DBE_FLAG
 DMA error flag.
DMA_ERR_SBE_FLAG
DMA_ERR_SGE_FLAG
DMA_ERR_NCE_FLAG
DMA_ERR_DOE_FLAG
DMA_ERR_DAE_FLAG
DMA_ERR_SOE_FLAG
DMA_ERR_SAE_FLAG
DMA_ERR_ERRCHAN_FLAG
DMA_ERR_ECX_FLAG
DMA_ERR_FLAG
DMA_CLEAR_DONE_STATUS(base, channel)
 get/clear DONE bit
DMA_GET_DONE_STATUS(base, channel)
DMA_ENABLE_ERROR_INT(base, channel)
 enable/disable error interrupt
DMA_DISABLE_ERROR_INT(base, channel)
DMA_CLEAR_ERROR_STATUS(base, channel)
 get/clear error status
DMA_GET_ERROR_STATUS(base, channel)
DMA_CLEAR_INT_STATUS(base, channel)
 get/clear INT status
DMA_GET_INT_STATUS(base, channel)
DMA_ENABLE_MAJOR_INT(base, channel)
 enable/disable MAJOR/HALF INT
DMA_ENABLE_HALF_INT(base, channel)
DMA_DISABLE_MAJOR_INT(base, channel)
DMA_DISABLE_HALF_INT(base, channel)
EDMA_TCD_ALIGN_SIZE
 EDMA tcd align size.
EDMA_CORE_BASE(base)
 EDMA base address convert macro.
EDMA_MP_BASE(base)

EDMA_CHANNEL_BASE(base, channel)

EDMA_TCD_BASE(base, channel)

EDMA_TCD_TYPE(x)

EDMA TCD type macro.

EDMA_TCD_SADDR(tcd, flag)

EDMA TCD address convert macro.

EDMA_TCD_SOFF(tcd, flag)

EDMA_TCD_ATTR(tcd, flag)

EDMA_TCD_NBYTES(tcd, flag)

EDMA_TCD_SLAST(tcd, flag)

EDMA_TCD_DADDR(tcd, flag)

EDMA_TCD_DOFF(tcd, flag)

EDMA_TCD_CITER(tcd, flag)

EDMA_TCD_DLAST_SGA(tcd, flag)

EDMA_TCD_CSR(tcd, flag)

EDMA_TCD_BITER(tcd, flag)

struct _edma_core_mp

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

Public Members

__IO uint32_t MP_CSR

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

__IO uint32_t MP_ES

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

struct _edma_core_channel

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

Public Members

__IO uint32_t CH_CSR

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

__IO uint32_t CH_ES

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

__IO uint32_t CH_INT

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

__IO uint32_t CH_SBR

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

__IO uint32_t CH_PRI

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

struct _edma5_core_tcd

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

Public Members

```

__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint32_t SADDR_HIGH
    SADDR HIGH register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t SLAST_SDA_HIGH
    SLAST SDA HIGH register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint32_t DADDR_HIGH
    DADDR HIGH register, used for destination address
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint32_t DLAST_SGA_HIGH
    DLASTSGA HIGH register, next tcd address used in scatter-gather mode
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
uint8_t RESERVED[16]
    Aligned 64 bytes
struct __edma4_core_tcd
    #include <fsl_edma_core.h> edma4 core TCD struture definition

```

Public Members

```

__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer

```


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

__IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

__IO uint32_t SLAST
SLAST register

__IO uint32_t DADDR
DADDR register, used for destination address

__IO uint16_t DOFF
DOFF register, used for destination offset

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

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

__IO uint16_t CSR
CSR register, for TCD control status

__IO uint16_t BITER
BITER register, begin minor loop count.

struct _edma_core_tcd
#include <fsl_edma_core.h> edma core TCD struture definition

union MP_REGS

Public Members

struct _edma_core_mp EDMA5_REG

struct EDMA5_REG

Public Members

__IO uint32_t MP_INT_LOW
Channel Control and Status, array offset: 0x10008, array step: 0x10000

__IO uint32_t MP_INT_HIGH
Channel Control and Status, array offset: 0x1000C, array step: 0x10000

__IO uint32_t MP_HRS_LOW
Channel Control and Status, array offset: 0x10010, array step: 0x10000

__IO uint32_t MP_HRS_HIGH
Channel Control and Status, array offset: 0x10014, array step: 0x10000

__IO uint32_t MP_STOPCH
Channel Control and Status, array offset: 0x10020, array step: 0x10000

__IO uint32_t MP_SSR_LOW
Channel Control and Status, array offset: 0x10030, array step: 0x10000

__IO uint32_t MP_SSR_HIGH
Channel Control and Status, array offset: 0x10034, array step: 0x10000

__IO uint32_t CH_GRPRI [64]
Channel Control and Status, array offset: 0x10100, array step: 0x10000

__IO uint32_t CH_MUX [64]
Channel Control and Status, array offset: 0x10200, array step: 0x10000

__IO uint32_t CH_PROT [64]
Channel Control and Status, array offset: 0x10400, array step: 0x10000

union CH_REGS

Public Members

struct *_edma_core_channel* EDMA5_REG

struct *_edma_core_channel* EDMA4_REG

struct EDMA5_REG

Public Members

__IO uint32_t CH_MATTR
Memory Attributes Register, array offset: 0x10018, array step: 0x8000

struct EDMA4_REG

Public Members

__IO uint32_t CH_MUX
Channel Multiplexor Configuration, array offset: 0x10014, array step: 0x10000

__IO uint16_t CH_MATTR
Memory Attributes Register, array offset: 0x10018, array step: 0x8000

union TCD_REGS

Public Members

edma4_core_tcd_t edma4_tcd

2.6 eDMA soc Driver

FSL_EDMA_SOC_DRIVER_VERSION
Driver version 1.0.0.

FSL_EDMA_SOC_IP_DMA3
DMA IP version.

FSL_EDMA_SOC_IP_DMA4

EDMA_BASE_PTRS
DMA base table.

EDMA_CHN_IRQS

FSL_FEATURE_EDMA_HAS_GLOBAL_MASTER_ID_REPLICATION
 FSL_FEATURE_EDMA_HAS_CONTINUOUS_LINK_MODE
 FSL_FEATURE_EDMA_MODULE_COUNT
 FSL_FEATURE_EDMA_HAS_CHANNEL_CONFIG
 FSL_FEATURE_EDMA_HAS_CHANNEL_SWAP_SIZE
 FSL_FEATURE_EDMA_HAS_CHANNEL_ACCESS_TYPE
 FSL_FEATURE_EDMA_HAS_CHANNEL_MEMORY_ATTRIBUTE
 FSL_FEATURE_EDMA_HAS_CHANNEL_SIGN_EXTENSION
 FSL_FEATURE_EDMA_MODULE_SUPPORT_MATTR(**base**)
 FSL_FEATURE_EDMA_MODULE_SUPPORT_SIGN_EXTENSION(**base**)
 FSL_FEATURE_EDMA_MODULE_SUPPORT_SWAP(**base**)
 FSL_FEATURE_EDMA_MODULE_SUPPORT_INSTR(**base**)
 EDMA_CHANNEL_OFFSET
 EDMA base address convert macro.
 EDMA_CHANNEL_ARRAY_STEP(**base**)

2.7 FGPIO Driver

2.8 GLIKEY

2.9 GLIKEY

Values:

enumerator kStatus_GLIKEY_LockedError
 GLIKEY status for locked SFR registers (unexpected) .

enumerator kStatus_GLIKEY_NotLocked
 GLIKEY status for unlocked SFR registers.

enumerator kStatus_GLIKEY_Locked
 GLIKEY status for locked SFR registers.

enumerator kStatus_GLIKEY_DisabledError
 GLIKEY status for disabled error.

FSL_GLIKEY_DRIVER_VERSION
 Defines GLIKEY driver version 2.0.1.

Change log:

- Version 2.0.1
 - Implement INIT state recovery from the LOCKED state after a reset when the previous index was locked.

- Version 2.0.0

- Initial version

GLIKEY_CODEWORD_STEP1

GLIKEY_CODEWORD_STEP2

GLIKEY_CODEWORD_STEP3

GLIKEY_CODEWORD_STEP4

GLIKEY_CODEWORD_STEP5

GLIKEY_CODEWORD_STEP6

GLIKEY_CODEWORD_STEP7

GLIKEY_CODEWORD_STEP_EN

GLIKEY_FSM_WR_DIS

GLIKEY_FSM_INIT

GLIKEY_FSM_STEP1

GLIKEY_FSM_STEP2

GLIKEY_FSM_STEP3

GLIKEY_FSM_STEP4

GLIKEY_FSM_LOCKED

GLIKEY_FSM_WR_EN

GLIKEY_FSM_SSR_RESET

uint32_t GLIKEY_GetStatus(GLIKEY_Type *base)

Retreives the current status of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Glikey status information

status_t GLIKEY_IsLocked(GLIKEY_Type *base)

Get if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_Locked – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status

status_t GLIKEY_CheckLock(GLIKEY_Type *base)

Check if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_LockedError – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status kStatus_Success if success

status_t GLIKEY_SyncReset(GLIKEY_Type *base)

Perform a synchronous reset of Glikey.

This function performs a synchronous reset of the Glikey. This results in:

- Glikey will return to the INIT state, unless it is in the LOCK state

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntEnable(GLIKEY_Type *base, uint32_t value)

Set interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[in]** Value to set the interrupt enable flag to, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_GetIntEnable(GLIKEY_Type *base, uint32_t *value)

Get interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[out]** Pointer which will be filled with the interrupt enable status, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success

status_t GLIKEY_ClearIntStatus(GLIKEY_Type *base)

Clear the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntStatus(GLIKEY_Type *base)

Set the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_Lock(GLIKEY_Type *base)

Lock Glikey SFR (Special Function Registers) interface.

This operation locks the Glikey SFR interface if it is not locked yet.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success

status_t GLIKEY_LockIndex(GLIKEY_Type *base)

Lock Glikey index.

This operation is used to lock a Glikey index. It can only be executed from the WR_EN state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_IsIndexLocked(GLIKEY_Type *base, uint32_t index)

Check if Glikey index is locked.

This operation returns the locking status of Glikey index.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns

kStatus_GLIKEY_Locked if locked, kStatus_GLIKEY_NotLocked if unlocked
Possible errors: kStatus_Fail

status_t GLIKEY_StartEnable(GLIKEY_Type *base, uint32_t index)

Start Glikey enable.

This operation is used to set a new index and start a the sequence to enable it. It needs to be started from the INIT state. If the new index is already locked Glikey will go to LOCKED state, otherwise it will go to STEP1 state. If this operation is used when Glikey is in any state other than INIT Glikey will go to WR_DIS state. It can only recover from this state through a reset (synchronous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_Fail

status_t GLIKEY_ContinueEnable(GLIKEY_Type *base, uint32_t codeword)

Continue Glikey enable.

This operation is used to progress through the different states of the state machine, starting from STEP1 until the state WR_EN is reached. Each next state of the state machine can

only be reached by providing the right codeword to this function. If anything goes wrong the state machine will go to WR_DIS state and can only recover from it through a reset (synchronous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- codeword – **[in]** Encoded word for progressing to next FSM state (see GLIKEY_CODEWORD_STEPx/EN)

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_Fail, kStatus_GLIKEY_DisabledError

status_t GLIKEY_EndOperation(GLIKEY_Type *base)

End Glikey operation.

This operation is used to end a Glikey operation. It can only be executed from the WR_EN, LOCKED and RESET states. Executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. After this operation Glikey will go to INIT state or stay in LOCKED state when the index was locked. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

A code-flow protected error code (see nxpCsslFlowProtection)

Returns

Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_ResetIndex(GLIKEY_Type *base, uint32_t index)

Reset Glikey index.

This operation is used to reset a Glikey index. It can only be executed from the INIT state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active or the index is locked this operation will return an error.

Returns

A code-flow protected error code (see nxpCsslFlowProtection)

Returns

Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

2.10 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.

2.11 GPIO Driver

```
void GPIO_PortInit(GPIO_Type *base)
```

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

```
void GPIO_PortDeinit(GPIO_Type *base)
```

Denitalizes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

`void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const gpio_pin_config_t *config)`

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the `GPIO_PinInit()` function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalInput,
    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

2.12 INPUTMUX: Input Multiplexing Driver

enum _inputmux_index_t

Values:

enumerator kINPUTMUXAON_INDEX_QTMR0_TMR0

enumerator kINPUTMUXAON_INDEX_QTMR0_TMR1

enumerator kINPUTMUXAON_INDEX_QTMR0_TMR2

enumerator kINPUTMUXAON_INDEX_QTMR0_TMR3

enumerator kINPUTMUXAON_INDEX_SOC_GLUE_XOR0_IN0

enumerator kINPUTMUXAON_INDEX_SOC_GLUE_XOR0_IN1

enumerator kINPUTMUXAON_INDEX_QTMR1_TMR0

enumerator kINPUTMUXAON_INDEX_QTMR1_TMR1

enumerator kINPUTMUXAON_INDEX_QTMR1_TMR2

enumerator kINPUTMUXAON_INDEX_QTMR1_TMR3

enumerator kINPUTMUXAON_INDEX_ACMP0_TRIG0

enumerator kINPUTMUXAON_INDEX_ACMP0_TRIG1

enumerator kINPUTMUXAON_INDEX_ACMP0_TRIG2

enumerator kINPUTMUXAON_INDEX_ACOMP0_TRIG3
enumerator kINPUTMUXAON_INDEX_LPADC0_TRIG0
enumerator kINPUTMUXAON_INDEX_LPADC0_TRIG1
enumerator kINPUTMUXAON_INDEX_LPADC0_TRIG2
enumerator kINPUTMUXAON_INDEX_LPADC0_TRIG3
enumerator kINPUTMUXAON_INDEX_AON_TRIG_OUT0
enumerator kINPUTMUXAON_INDEX_AON_TRIG_OUT1
enumerator kINPUTMUXAON_INDEX_SOC_GLUE_CMPPADS_PCTRL_XOR_IN0
enumerator kINPUTMUXAON_INDEX_SOC_GLUE_CTRLPADS_PCTRL_XOR_IN0
enumerator kINPUTMUXAON_INDEX_LC_ROT_SOC_LOGIC_IN
enumerator kINPUTMUXAON_INDEX_LCSENSE_SEQ_PTRIG_GLUE_IN
enumerator kINPUTMUXAON_INDEX_LCSENSE_SEQ_TICKS_GLUE_IN
enumerator kINPUTMUXAON_INDEX_CMP0_SAMPLE
enumerator kINPUTMUXAON_INDEX_CMP0_RR_TRIG
enumerator kINPUTMUXAON_INDEX_LPI2C0_TRIG
enumerator kINPUTMUXAON_INDEX_LPUART0

enum _inputmux_connection_t

INPUTMUX connections type.

Values:

enumerator kINPUTMUXAON_AonTrigIn0ToQtmr0Tmrn
QTMRO_TMRn: QTMRO Input Connections aon_trig_in0 input is selected
enumerator kINPUTMUXAON_AonTrigIn1ToQtmr0Tmrn
aon_trig_in1 input is selected
enumerator kINPUTMUXAON_AonTrigIn2ToQtmr0Tmrn
aon_trig_in2 input is selected
enumerator kINPUTMUXAON_AonTrigIn3ToQtmr0Tmrn
aon_trig_in3 input is selected
enumerator kINPUTMUXAON_AonTrigIn4ToQtmr0Tmrn
aon_trig_in4 input is selected
enumerator kINPUTMUXAON_AonTrigIn5ToQtmr0Tmrn
aon_trig_in5 input is selected
enumerator kINPUTMUXAON_AonTrigIn6ToQtmr0Tmrn
aon_trig_in6 input is selected
enumerator kINPUTMUXAON_AonTrigIn7ToQtmr0Tmrn
aon_trig_in7 input is selected
enumerator kINPUTMUXAON_Cm33TeToQtmr0Tmrn
cm33 transmit event is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr0OutputToQtmr0Tmrn
qtmr1_tmr0_output is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr1OutputToQtmr0Tmrn
qtmr1_tmr1_output is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr2OutputToQtmr0Tmrn
qtmr1_tmr2_output is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr3OutputToQtmr0Tmrn
qtmr1_tmr3_output is selected

enumerator kINPUTMUX_AON_Cmp0OutToQtmr0Tmrn
cmp0_out is selected

enumerator kINPUTMUX_AON_Lpi2c0CeopToQtmr0Tmrn
lpi2c0 controller end of packet is selected

enumerator kINPUTMUX_AON_Lpi2c0TeopToQtmr0Tmrn
lpi2c0 target end of packet is selected

enumerator kINPUTMUX_AON_Lpuart0RdwToQtmr0Tmrn
lpuart0 received data word is selected

enumerator kINPUTMUX_AON_Lpuart0TdwToQtmr0Tmrn
lpuart0 transmitted data word is selected

enumerator kINPUTMUX_AON_Lpuart0RliToQtmr0Tmrn
lpuart0 receive line idle is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut0ToQtmr0Tmrn
lpadc_trigger_out[0] is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut1ToQtmr0Tmrn
lpadc_trigger_out[1] is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut2ToQtmr0Tmrn
lpadc_trigger_out[2] is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut3ToQtmr0Tmrn
lpadc_trigger_out[3] is selected

enumerator kINPUTMUX_AON_Lptmr0OToQtmr0Tmrn
lptmr0 output is selected

enumerator kINPUTMUX_AON_LcRotSocLogicOut1ToQtmr0Tmrn
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUX_AON_LcRotSocLogicOut2ToQtmr0Tmrn
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUX_AON_LcRotSocLogicOut3ToQtmr0Tmrn
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUX_AON_LcRotSocLogicOut4ToQtmr0Tmrn
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUX_AON_Qtmr0Tmr0DirToQtmr0Tmrn
qtmr0_tmr0_dir is selected

enumerator kINPUTMUX_AON_Qtmr0Tmr1DirToQtmr0Tmrn
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToQtmr0Tmrn
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToQtmr0Tmrn
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToQtmr0Tmrn
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToQtmr0Tmrn
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToQtmr0Tmrn
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToQtmr0Tmrn
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToQtmr0Tmrn
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToQtmr0Tmrn
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToQtmr0Tmrn
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToQtmr0Tmrn
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToQtmr0Tmrn
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToQtmr0Tmrn
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToQtmr0Tmrn
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToQtmr0Tmrn
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToQtmr0Tmrn
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToQtmr0Tmrn
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToQtmr0Tmrn
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToQtmr0Tmrn
lcsense_sequencer_primary_trigger_glue_out is selected SOC_GLUE_XOR0_INn:
SOC_GLUE_XOR0 trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToSocGlueXor0Inn
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToSocGlueXor0Inn
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToSocGlueXor0Inn
aon_trig_in2 input is selected

enumerator kINPUTMUXAON__AonTrigIn3ToSocGlueXor0Inn
aon_trig_in3 input is selected

enumerator kINPUTMUXAON__AonTrigIn4ToSocGlueXor0Inn
aon_trig_in4 input is selected

enumerator kINPUTMUXAON__AonTrigIn5ToSocGlueXor0Inn
aon_trig_in5 input is selected

enumerator kINPUTMUXAON__AonTrigIn6ToSocGlueXor0Inn
aon_trig_in6 input is selected

enumerator kINPUTMUXAON__AonTrigIn7ToSocGlueXor0Inn
aon_trig_in7 input is selected

enumerator kINPUTMUXAON__Cm33TeToSocGlueXor0Inn
cm33 transmit event is selected

enumerator kINPUTMUXAON__Qtmr1Tmr0OutputToSocGlueXor0Inn
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON__Qtmr1Tmr1OutputToSocGlueXor0Inn
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON__Qtmr1Tmr2OutputToSocGlueXor0Inn
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON__Qtmr1Tmr3OutputToSocGlueXor0Inn
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON__Cmp0OutToSocGlueXor0Inn
cmp0_out is selected

enumerator kINPUTMUXAON__Lpi2c0CeopToSocGlueXor0Inn
lpi2c0 controller end of packet is selected

enumerator kINPUTMUXAON__Lpi2c0TeopToSocGlueXor0Inn
lpi2c0 target end of packet is selected

enumerator kINPUTMUXAON__Lpuart0RdwToSocGlueXor0Inn
lpuart0 received data word is selected

enumerator kINPUTMUXAON__Lpuart0TdwToSocGlueXor0Inn
lpuart0 transmitted data word is selected

enumerator kINPUTMUXAON__Lpuart0RliToSocGlueXor0Inn
lpuart0 receive line idle is selected

enumerator kINPUTMUXAON__LpadcTriggerOut0ToSocGlueXor0Inn
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON__LpadcTriggerOut1ToSocGlueXor0Inn
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON__LpadcTriggerOut2ToSocGlueXor0Inn
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON__LpadcTriggerOut3ToSocGlueXor0Inn
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON__Lptmr0OToSocGlueXor0Inn
lptmr0 output is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut1ToSocGlueXor0Inn
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut2ToSocGlueXor0Inn
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut3ToSocGlueXor0Inn
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut4ToSocGlueXor0Inn
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0DirToSocGlueXor0Inn
qtmr0_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1DirToSocGlueXor0Inn
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToSocGlueXor0Inn
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToSocGlueXor0Inn
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToSocGlueXor0Inn
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToSocGlueXor0Inn
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToSocGlueXor0Inn
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToSocGlueXor0Inn
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToSocGlueXor0Inn
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToSocGlueXor0Inn
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToSocGlueXor0Inn
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToSocGlueXor0Inn
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToSocGlueXor0Inn
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToSocGlueXor0Inn
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToSocGlueXor0Inn
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToSocGlueXor0Inn
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToSocGlueXor0Inn
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToSocGlueXor0Inn
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToSocGlueXor0Inn
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToSocGlueXor0Inn
lcsense_sequencer_primary_trigger_glue_out is selected QTMR1_TMRn: QTMR1 Input Connections

enumerator kINPUTMUXAON_AonTrigIn0ToQtmr1Tmrn
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToQtmr1Tmrn
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToQtmr1Tmrn
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToQtmr1Tmrn
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToQtmr1Tmrn
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToQtmr1Tmrn
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToQtmr1Tmrn
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToQtmr1Tmrn
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToQtmr1Tmrn
cm33 transmit event is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToQtmr1Tmrn
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToQtmr1Tmrn
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToQtmr1Tmrn
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToQtmr1Tmrn
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_Cmp0OutToQtmr1Tmrn
cmp0_out is selected

enumerator kINPUTMUXAON_Lpi2c0CeopToQtmr1Tmrn
lpi2c0 controller end of packet is selected

enumerator kINPUTMUXAON_Lpi2c0TeopToQtmr1Tmrn
lpi2c0 target end of packet is selected

enumerator kINPUTMUXAON_Lpuart0RdwToQtmr1Tmrn
lpuart0 received data word is selected

enumerator kINPUTMUXAON_Lpuart0TdwToQtmr1Tmrn
lpuart0 transmitted data word is selected

enumerator kINPUTMUXAON_Lpuart0RliToQtmr1Tmrn
lpuart0 receive line idle is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToQtmr1Tmrn
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToQtmr1Tmrn
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToQtmr1Tmrn
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToQtmr1Tmrn
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Lptmr0OToQtmr1Tmrn
lptmr0 output is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut1ToQtmr1Tmrn
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut2ToQtmr1Tmrn
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut3ToQtmr1Tmrn
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut4ToQtmr1Tmrn
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0DirToQtmr1Tmrn
qtmr0_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1DirToQtmr1Tmrn
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToQtmr1Tmrn
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToQtmr1Tmrn
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToQtmr1Tmrn
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToQtmr1Tmrn
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToQtmr1Tmrn
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToQtmr1Tmrn
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToQtmr1Tmrn
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToQtmr1Tmrn
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToQtmr1Tmrn
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToQtmr1Tmrn
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToQtmr1Tmrn
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToQtmr1Tmrn
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToQtmr1Tmrn
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToQtmr1Tmrn
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToQtmr1Tmrn
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToQtmr1Tmrn
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToQtmr1Tmrn
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToQtmr1Tmrn
lcsense_sequencer_primary_trigger_glue_out is selected
ACMP0_TRIGn: ACMP0 Input Connections

enumerator kINPUTMUXAON_AonTrigIn0ToAcmp0Trign
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToAcmp0Trign
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToAcmp0Trign
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToAcmp0Trign
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToAcmp0Trign
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToAcmp0Trign
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToAcmp0Trign
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToAcmp0Trign
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToAcmp0Trign
cm33 transmit event is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToAcmp0Trign
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToAcmp0Trign
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToAcmp0Trign
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToAcmp0Trign
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_Lptmr00ToAcmp0Trign
lptmr0 output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToAcmp0Trign
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToAcmp0Trign
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToAcmp0Trign
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToAcmp0Trign
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_WuuToAcmp0Trign
wu is selected

enumerator kINPUTMUXAON_GpioApet0ToAcmp0Trign
gpio (aon) pin event trig 0 is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToAcmp0Trign
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToAcmp0Trign
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToAcmp0Trign
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToAcmp0Trign
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Acmp0RacoToAcmp0Trign
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToAcmp0Trign
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToAcmp0Trign
soc_glue_xor0_out is selected LPADC0_TRIGN: LPADC trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToLpadc0Trign
aon_trig_in0 is selected

enumerator kINPUTMUXAON_AonTrigIn1ToLpadc0Trign
aon_trig_in1 is selected

enumerator kINPUTMUXAON_AonTrigIn2ToLpadc0Trign
aon_trig_in2 is selected

enumerator kINPUTMUXAON_AonTrigIn03ToLpadc0Trign
aon_trig_in03 is selected

enumerator kINPUTMUXAON_AonTrigIn04ToLpadc0Trign
aon_trig_in04 is selected

enumerator kINPUTMUXAON_AonTrigIn5ToLpadc0Trign
aon_trig_in5 is selected

enumerator kINPUTMUXAON_AonTrigIn6ToLpadc0Trign
aon_trig_in6 is selected

enumerator kINPUTMUXAON_AonTrigIn7ToLpadc0Trign
aon_trig_in7 is selected

enumerator kINPUTMUXAON_Cm33TeToLpadc0Trign
cm33 transmit event is selected

enumerator kINPUTMUXAON_LpcmpOutToLpadc0Trign
lpcmp_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToLpadc0Trign
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToLpadc0Trign
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToLpadc0Trign
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToLpadc0Trign
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_Lptmr0OToLpadc0Trign
lptmr0 output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToLpadc0Trign
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToLpadc0Trign
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToLpadc0Trign
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToLpadc0Trign
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_WuuToLpadc0Trign
wuu is selected

enumerator kINPUTMUXAON_GpioApet0ToLpadc0Trign
gpio (aon) pin event trig 0 is selected

enumerator kINPUTMUXAON_Acmp0RacoToLpadc0Trign
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToLpadc0Trign
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToLpadc0Trign
soc_glue_xor0_out is selected

AON_TRIGGER_OUTn: AON Trigger Output Connections

enumerator kINPUTMUXAON_Cm33TeToAonTrigOutn
cm33 transmit event is selected

enumerator kINPUTMUXAON_AonLpuart0IToAonTrigOutn
aon_lpuart0 (ipp_do_lpuart_txd) is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToAonTrigOutn
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToAonTrigOutn
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToAonTrigOutn
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_ParkedOutIntvertedToAonTrigOutn
parked_out_intverted is selected

enumerator kINPUTMUXAON_AonLptmr0ToAonTrigOutn
aon_lptmr0 output is selected

enumerator kINPUTMUXAON_Lpcomp0OutToAonTrigOutn
lpcomp0_out is selected

enumerator kINPUTMUXAON_Acmp0RacoToAonTrigOutn
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToAonTrigOutn
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToAonTrigOutn
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToAonTrigOutn
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToAonTrigOutn
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToAonTrigOutn
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToAonTrigOutn
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToAonTrigOutn
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToAonTrigOutn
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToAonTrigOutn
qtmr1_tmr3_output is selected SOC_GLUE_CMPPADS_PCTRL_XOR_IN0:
SOC_GLUE_CMPPADS_PCTRL_XOR_IN0 trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToSocGlueCmppadsPctrlXorIn0
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToSocGlueCmppadsPctrlXorIn0
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToSocGlueCmppadsPctrlXorIn0
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToSocGlueCmppadsPctrlXorIn0
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToSocGlueCmppadsPctrlXorIn0
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToSocGlueCmppadsPctrlXorIn0
aon_trig_in5 input is selected

enumerator kINPUTMUX_AON_AonTrigIn6ToSocGlueCmppsPctrlXorIn0
aon_trig_in6 input is selected

enumerator kINPUTMUX_AON_AonTrigIn7ToSocGlueCmppsPctrlXorIn0
aon_trig_in7 input is selected

enumerator kINPUTMUX_AON_Cm33TeToSocGlueCmppsPctrlXorIn0
cm33 transmit event is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr0OutputToSocGlueCmppsPctrlXorIn0
qtmr1_tmr0_output is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr1OutputToSocGlueCmppsPctrlXorIn0
qtmr1_tmr1_output is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr2OutputToSocGlueCmppsPctrlXorIn0
qtmr1_tmr2_output is selected

enumerator kINPUTMUX_AON_Qtmr1Tmr3OutputToSocGlueCmppsPctrlXorIn0
qtmr1_tmr3_output is selected

enumerator kINPUTMUX_AON_Cmp0OutToSocGlueCmppsPctrlXorIn0
cmp0_out is selected

enumerator kINPUTMUX_AON_Lpi2c0CeopToSocGlueCmppsPctrlXorIn0
lpi2c0 controller end of packet is selected

enumerator kINPUTMUX_AON_Lpi2c0TeopToSocGlueCmppsPctrlXorIn0
lpi2c0 target end of packet is selected

enumerator kINPUTMUX_AON_Lpuart0RdwToSocGlueCmppsPctrlXorIn0
lpuart0 received data word is selected

enumerator kINPUTMUX_AON_Lpuart0TdwToSocGlueCmppsPctrlXorIn0
lpuart0 transmitted data word is selected

enumerator kINPUTMUX_AON_Lpuart0RliToSocGlueCmppsPctrlXorIn0
lpuart0 receive line idle is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut0ToSocGlueCmppsPctrlXorIn0
lpadc_trigger_out[0] is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut1ToSocGlueCmppsPctrlXorIn0
lpadc_trigger_out[1] is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut2ToSocGlueCmppsPctrlXorIn0
lpadc_trigger_out[2] is selected

enumerator kINPUTMUX_AON_LpadcTriggerOut3ToSocGlueCmppsPctrlXorIn0
lpadc_trigger_out[3] is selected

enumerator kINPUTMUX_AON_Lptmr0ToSocGlueCmppsPctrlXorIn0
lptmr0 output is selected

enumerator kINPUTMUX_AON_LcRotSocLogicOut1ToSocGlueCmppsPctrlXorIn0
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUX_AON_LcRotSocLogicOut2ToSocGlueCmppsPctrlXorIn0
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUX_AON_LcRotSocLogicOut3ToSocGlueCmppsPctrlXorIn0
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut4ToSocGlueCmppadsPctrlXorIn0
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0DirToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1DirToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToSocGlueCmppadsPctrlXorIn0
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToSocGlueCmppadsPctrlXorIn0
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToSocGlueCmppadsPctrlXorIn0
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToSocGlueCmppadsPctrlXorIn0
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToSocGlueCmppadsPctrlXorIn0
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToSocGlueCmppadsPctrlXorIn0
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToSocGlueCmppadsPctrlXorIn0
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToSocGlueCmppadsPctrlXorIn0
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToSocGlueCmppadsPctrlXorIn0
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToSocGlueCmppadsPctrlXorIn0
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToSocGlueCmppadsPctrlXorIn0
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToSocGlueCmppadsPctrlXorIn0
qtmr0_tmr3_output is selected

enumerator

kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToSocGlueCmppsPctrlXorIn0
lcsense_sequencer_primary_trigger_glue_out is selected
SOC_GLUE_CTRLPADS_PCTRL_XOR_IN0: SOC_GLUE_CTRLPADS_PCTRL_XOR_IN0
trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToSocGlueCtrlpadsPctrlXorIn0
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToSocGlueCtrlpadsPctrlXorIn0
cm33 transmit event is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_Cmp0OutToSocGlueCtrlpadsPctrlXorIn0
cmp0_out is selected

enumerator kINPUTMUXAON_Lpi2c0CeopToSocGlueCtrlpadsPctrlXorIn0
lpi2c0 controller end of packet is selected

enumerator kINPUTMUXAON_Lpi2c0TeopToSocGlueCtrlpadsPctrlXorIn0
lpi2c0 target end of packet is selected

enumerator kINPUTMUXAON_Lpuart0RdwToSocGlueCtrlpadsPctrlXorIn0
lpuart0 received data word is selected

enumerator kINPUTMUXAON_Lpuart0TdwToSocGlueCtrlpadsPctrlXorIn0
lpuart0 transmitted data word is selected

enumerator kINPUTMUXAON_Lpuart0RliToSocGlueCtrlpadsPctrlXorIn0
lpuart0 receive line idle is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToSocGlueCtrlpadsPctrlXorIn0
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToSocGlueCtrlpadsPctrlXorIn0
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToSocGlueCtrlpadsPctrlXorIn0
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToSocGlueCtrlpadsPctrlXorIn0
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Lptmr00ToSocGlueCtrlpadsPctrlXorIn0
lptmr0 output is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut1ToSocGlueCtrlpadsPctrlXorIn0
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut2ToSocGlueCtrlpadsPctrlXorIn0
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut3ToSocGlueCtrlpadsPctrlXorIn0
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut4ToSocGlueCtrlpadsPctrlXorIn0
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0DirToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1DirToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToSocGlueCtrlpadsPctrlXorIn0
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToSocGlueCtrlpadsPctrlXorIn0
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToSocGlueCtrlpadsPctrlXorIn0
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToSocGlueCtrlpadsPctrlXorIn0
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToSocGlueCtrlpadsPctrlXorIn0
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToSocGlueCtrlpadsPctrlXorIn0
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToSocGlueCtrlpadsPctrlXorIn0
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToSocGlueCtrlpadsPctrlXorIn0
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToSocGlueCtrlpadsPctrlXorIn0
qtmr0_tmr3_output is selected

enumerator
kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToSocGlueCtrlpadsPctrlXorIn0
lcsense_sequencer_primary_trigger_glue_out is selected LC_ROT_SOC_LOGIC_IN:
LC_ROT_SOC_LOGIC_IN trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToLcRotSocLogicIn
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToLcRotSocLogicIn
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToLcRotSocLogicIn
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToLcRotSocLogicIn
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToLcRotSocLogicIn
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToLcRotSocLogicIn
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToLcRotSocLogicIn
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToLcRotSocLogicIn
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToLcRotSocLogicIn
cm33 transmit event is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToLcRotSocLogicIn
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToLcRotSocLogicIn
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToLcRotSocLogicIn
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToLcRotSocLogicIn
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_Cmp0OutToLcRotSocLogicIn
cmp0_out is selected

enumerator kINPUTMUXAON_Lpi2c0CeopToLcRotSocLogicIn
lpi2c0 controller end of packet is selected

enumerator kINPUTMUXAON_Lpi2c0TeopToLcRotSocLogicIn
lpi2c0 target end of packet is selected

enumerator kINPUTMUXAON_Lpuart0RdwToLcRotSocLogicIn
lpuart0 received data word is selected

enumerator kINPUTMUXAON_Lpuart0TdwToLcRotSocLogicIn
lpuart0 transmitted data word is selected

enumerator kINPUTMUXAON_Lpuart0RliToLcRotSocLogicIn
lpuart0 receive line idle is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToLcRotSocLogicIn
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToLcRotSocLogicIn
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToLcRotSocLogicIn
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToLcRotSocLogicIn
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Lptmr0OToLcRotSocLogicIn
lptmr0 output is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut1ToLcRotSocLogicIn
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut2ToLcRotSocLogicIn
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut3ToLcRotSocLogicIn
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut4ToLcRotSocLogicIn
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0DirToLcRotSocLogicIn
qtmr0_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1DirToLcRotSocLogicIn
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToLcRotSocLogicIn
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToLcRotSocLogicIn
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToLcRotSocLogicIn
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToLcRotSocLogicIn
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToLcRotSocLogicIn
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToLcRotSocLogicIn
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToLcRotSocLogicIn
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToLcRotSocLogicIn
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToLcRotSocLogicIn
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToLcRotSocLogicIn
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToLcRotSocLogicIn
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToLcRotSocLogicIn
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToLcRotSocLogicIn
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToLcRotSocLogicIn
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToLcRotSocLogicIn
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToLcRotSocLogicIn
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToLcRotSocLogicIn
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToLcRotSocLogicIn
lcsense_sequencer_primary_trigger_glue_out is selected LC-
SENSE_SEQ_PTRIG_GLUE_IN: LcsenseSeqPtrigGlueIn trigger input con-
nections

enumerator kINPUTMUXAON_AonTrigIn0ToLcsenseSeqPtrigGlueIn
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToLcsenseSeqPtrigGlueIn
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToLcsenseSeqPtrigGlueIn
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToLcsenseSeqPtrigGlueIn
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToLcsenseSeqPtrigGlueIn
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToLcsenseSeqPtrigGlueIn
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToLcsenseSeqPtrigGlueIn
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToLcsenseSeqPtrigGlueIn
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToLcsenseSeqPtrigGlueIn
cm33 transmit event is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToLcsenseSeqPtrigGlueIn
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToLcsenseSeqPtrigGlueIn
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToLcsenseSeqPtrigGlueIn
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToLcsenseSeqPtrigGlueIn
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_Cmp0OutToLcsenseSeqPtrigGlueIn
cmp0_out is selected

enumerator kINPUTMUXAON_Lpi2c0CeopToLcsenseSeqPtrigGlueIn
lpi2c0 controller end of packet is selected

enumerator kINPUTMUXAON_Lpi2c0TeopToLcsenseSeqPtrigGlueIn
lpi2c0 target end of packet is selected

enumerator kINPUTMUXAON_Lpuart0RdwToLcsenseSeqPtrigGlueIn
lpuart0 received data word is selected

enumerator kINPUTMUXAON_Lpuart0TdwToLcsenseSeqPtrigGlueIn
lpuart0 transmitted data word is selected

enumerator kINPUTMUXAON_Lpuart0RliToLcsenseSeqPtrigGlueIn
lpuart0 receive line idle is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToLcsenseSeqPtrigGlueIn
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToLcsenseSeqPtrigGlueIn
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToLcsenseSeqPtrigGlueIn
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToLcsenseSeqPtrigGlueIn
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Lptmr0OToLcsenseSeqPtrigGlueIn
lptmr0 output is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut1ToLcsenseSeqPtrigGlueIn
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut2ToLcsenseSeqPtrigGlueIn
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut3ToLcsenseSeqPtrigGlueIn
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut4ToLcsenseSeqPtrigGlueIn
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0DirToLcsenseSeqPtrigGlueIn
qtmr0_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1DirToLcsenseSeqPtrigGlueIn
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToLcsenseSeqPtrigGlueIn
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToLcsenseSeqPtrigGlueIn
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToLcsenseSeqPtrigGlueIn
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToLcsenseSeqPtrigGlueIn
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToLcsenseSeqPtrigGlueIn
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToLcsenseSeqPtrigGlueIn
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToLcsenseSeqPtrigGlueIn
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToLcsenseSeqPtrigGlueIn
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToLcsenseSeqPtrigGlueIn
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToLcsenseSeqPtrigGlueIn
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToLcsenseSeqPtrigGlueIn
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToLcsenseSeqPtrigGlueIn
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToLcsenseSeqPtrigGlueIn
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToLcsenseSeqPtrigGlueIn
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToLcsenseSeqPtrigGlueIn
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToLcsenseSeqPtrigGlueIn
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToLcsenseSeqPtrigGlueIn
qtmr0_tmr3_output is selected

enumerator

kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToLcsenseSeqPtrigGlueIn
lcsense_sequencer_primary_trigger_glue_out is selected LC-
SENSE_SEQ_TICKS_GLUE_IN: LCSenseSeqTicksGlueIn trigger input con-
nections

enumerator kINPUTMUXAON_AonTrigIn0ToLcsenseSeqTicksGlueIn
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToLcsenseSeqTicksGlueIn
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToLcsenseSeqTicksGlueIn
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToLcsenseSeqTicksGlueIn
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToLcsenseSeqTicksGlueIn
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToLcsenseSeqTicksGlueIn
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToLcsenseSeqTicksGlueIn
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToLcsenseSeqTicksGlueIn
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToLcsenseSeqTicksGlueIn
cm33 transmit event is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToLcsenseSeqTicksGlueIn
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToLcsenseSeqTicksGlueIn
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToLcsenseSeqTicksGlueIn
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToLcsenseSeqTicksGlueIn
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_Cmp0OutToLcsenseSeqTicksGlueIn
cmp0_out is selected

enumerator kINPUTMUXAON_Lpi2c0CeopToLcsenseSeqTicksGlueIn
lpi2c0 controller end of packet is selected

enumerator kINPUTMUXAON_Lpi2c0TeopToLcsenseSeqTicksGlueIn
lpi2c0 target end of packet is selected

enumerator kINPUTMUXAON_Lpuart0RdwToLcsenseSeqTicksGlueIn
lpuart0 received data word is selected

enumerator kINPUTMUXAON_Lpuart0TdwToLcsenseSeqTicksGlueIn
lpuart0 transmitted data word is selected

enumerator kINPUTMUXAON_Lpuart0RliToLcsenseSeqTicksGlueIn
lpuart0 receive line idle is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToLcsenseSeqTicksGlueIn
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToLcsenseSeqTicksGlueIn
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToLcsenseSeqTicksGlueIn
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToLcsenseSeqTicksGlueIn
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Lptmr00ToLcsenseSeqTicksGlueIn
lptmr0 output is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut1ToLcsenseSeqTicksGlueIn
lc_rot_soc_logic_out1 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut2ToLcsenseSeqTicksGlueIn
lc_rot_soc_logic_out2 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut3ToLcsenseSeqTicksGlueIn
lc_rot_soc_logic_out3 is selected

enumerator kINPUTMUXAON_LcRotSocLogicOut4ToLcsenseSeqTicksGlueIn
lc_rot_soc_logic_out4 is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0DirToLcsenseSeqTicksGlueIn
qtmr0_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1DirToLcsenseSeqTicksGlueIn
qtmr0_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2DirToLcsenseSeqTicksGlueIn
qtmr0_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3DirToLcsenseSeqTicksGlueIn
qtmr0_tmr3_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0DirToLcsenseSeqTicksGlueIn
qtmr1_tmr0_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1DirToLcsenseSeqTicksGlueIn
qtmr1_tmr1_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2DirToLcsenseSeqTicksGlueIn
qtmr1_tmr2_dir is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3DirToLcsenseSeqTicksGlueIn
qtmr1_tmr3_dir is selected

enumerator kINPUTMUXAON_Acmp0RacoToLcsenseSeqTicksGlueIn
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToLcsenseSeqTicksGlueIn
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_Logic0ToLcsenseSeqTicksGlueIn
logic_0 is selected

enumerator kINPUTMUXAON_Logic1ToLcsenseSeqTicksGlueIn
logic_1 is selected

enumerator kINPUTMUXAON_WuuOToLcsenseSeqTicksGlueIn
wuu output is selected

enumerator kINPUTMUXAON_GpioApet0ToLcsenseSeqTicksGlueIn
gpio (aon) pin event trig 0 input is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToLcsenseSeqTicksGlueIn
soc_glue_xor0_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToLcsenseSeqTicksGlueIn
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToLcsenseSeqTicksGlueIn
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToLcsenseSeqTicksGlueIn
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToLcsenseSeqTicksGlueIn
qtmr0_tmr3_output is selected

enumerator
kINPUTMUXAON_LcsenseSequencerPrimaryTriggerGlueOutToLcsenseSeqTicksGlueIn
lcsense_sequencer_primary_trigger_glue_out is selected CMP0_SAMPLE: ACMP0 SAM-
PLE trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToCmp0Sample
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToCmp0Sample
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToCmp0Sample
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToCmp0Sample
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToCmp0Sample
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToCmp0Sample
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToCmp0Sample
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToCmp0Sample
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToCmp0Sample
cm33 transmit event is selected

enumerator kINPUTMUXAON_LpcmpOutToCmp0Sample
lpcmp_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToCmp0Sample
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToCmp0Sample
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToCmp0Sample
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToCmp0Sample
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_Lptmr0ToCmp0Sample
lptmr0 output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToCmp0Sample
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToCmp0Sample
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToCmp0Sample
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToCmp0Sample
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_WuuToCmp0Sample
wuu is selected

enumerator kINPUTMUXAON_GpioApet0ToCmp0Sample
gpio (aon) pin event trig 0 is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToCmp0Sample
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToCmp0Sample
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToCmp0Sample
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToCmp0Sample
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToCmp0Sample
soc_glue_xor0_out is selected CMP0_RR_TRIG: ACMP0 RR trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToCmp0RrTrig
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToCmp0RrTrig
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToCmp0RrTrig
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToCmp0RrTrig
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToCmp0RrTrig
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToCmp0RrTrig
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToCmp0RrTrig
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToCmp0RrTrig
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToCmp0RrTrig
cm33 transmit event is selected

enumerator kINPUTMUXAON_LpcmpOutToCmp0RrTrig
lpcmp_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToCmp0RrTrig
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToCmp0RrTrig
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToCmp0RrTrig
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToCmp0RrTrig
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_Lptmr0OToCmp0RrTrig
lptmr0 output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToCmp0RrTrig
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToCmp0RrTrig
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToCmp0RrTrig
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToCmp0RrTrig
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_WuuToCmp0RrTrig
wuu is selected

enumerator kINPUTMUXAON_GpioApet0ToCmp0RrTrig
gpio (aon) pin event trig 0 is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToCmp0RrTrig
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToCmp0RrTrig
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToCmp0RrTrig
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToCmp0RrTrig
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToCmp0RrTrig
soc_glue_xor0_out is selected LPI2C0_TRIG: LPI2C0 trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToLpi2c0Trig
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToLpi2c0Trig
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToLpi2c0Trig
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToLpi2c0Trig
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToLpi2c0Trig
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToLpi2c0Trig
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToLpi2c0Trig
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToLpi2c0Trig
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToLpi2c0Trig
cm33 transmit event is selected

enumerator kINPUTMUXAON_LpcmpOutToLpi2c0Trig
lpcmp_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToLpi2c0Trig
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToLpi2c0Trig
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToLpi2c0Trig
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToLpi2c0Trig
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_Lptmr0OToLpi2c0Trig
lptmr0 output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToLpi2c0Trig
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToLpi2c0Trig
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToLpi2c0Trig
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToLpi2c0Trig
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_WuuToLpi2c0Trig
wuu is selected

enumerator kINPUTMUXAON_GpioApet0ToLpi2c0Trig
gpio (aon) pin event trig 0 is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToLpi2c0Trig
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToLpi2c0Trig
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToLpi2c0Trig
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToLpi2c0Trig
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Acmp0RacoToLpi2c0Trig
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToLpi2c0Trig
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToLpi2c0Trig
soc_glue_xor0_out is selected LPUART0: LPUART0 trigger input connections

enumerator kINPUTMUXAON_AonTrigIn0ToLPUART0
aon_trig_in0 input is selected

enumerator kINPUTMUXAON_AonTrigIn1ToLPUART0
aon_trig_in1 input is selected

enumerator kINPUTMUXAON_AonTrigIn2ToLPUART0
aon_trig_in2 input is selected

enumerator kINPUTMUXAON_AonTrigIn3ToLPUART0
aon_trig_in3 input is selected

enumerator kINPUTMUXAON_AonTrigIn4ToLPUART0
aon_trig_in4 input is selected

enumerator kINPUTMUXAON_AonTrigIn5ToLPUART0
aon_trig_in5 input is selected

enumerator kINPUTMUXAON_AonTrigIn6ToLPUART0
aon_trig_in6 input is selected

enumerator kINPUTMUXAON_AonTrigIn7ToLPUART0
aon_trig_in7 input is selected

enumerator kINPUTMUXAON_Cm33TeToLPUART0
cm33 transmit event is selected

enumerator kINPUTMUXAON_LpcmpOutToLPUART0
lpcmp_out is selected

enumerator kINPUTMUXAON_Qtmr0Tmr0OutputToLPUART0
qtmr0_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr1OutputToLPUART0
qtmr0_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr2OutputToLPUART0
qtmr0_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr0Tmr3OutputToLPUART0
qtmr0_tmr3_output is selected

enumerator kINPUTMUXAON_Lptmr0ToLPUART0
lptmr0 output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr0OutputToLPUART0
qtmr1_tmr0_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr1OutputToLPUART0
qtmr1_tmr1_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr2OutputToLPUART0
qtmr1_tmr2_output is selected

enumerator kINPUTMUXAON_Qtmr1Tmr3OutputToLPUART0
qtmr1_tmr3_output is selected

enumerator kINPUTMUXAON_WuuToLPUART0
wu is selected

enumerator kINPUTMUXAON_GpioApet0ToLPUART0
gpio (aon) pin event trig 0 is selected

enumerator kINPUTMUXAON_LpadcTriggerOut0ToLPUART0
lpadc_trigger_out[0] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut1ToLPUART0
lpadc_trigger_out[1] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut2ToLPUART0
lpadc_trigger_out[2] is selected

enumerator kINPUTMUXAON_LpadcTriggerOut3ToLPUART0
lpadc_trigger_out[3] is selected

enumerator kINPUTMUXAON_Acmp0RacoToLPUART0
acmp0 raw analog comparator output

enumerator kINPUTMUXAON_Acmp0AonCoutToLPUART0
acmp0_aon_cout is selected

enumerator kINPUTMUXAON_SocGlueXor0OutToLPUART0
soc_glue_xor0_out is selected

typedef enum *_inputmux_index_t* inputmux_index_t

typedef enum *_inputmux_connection_t* inputmux_connection_t
INPUTMUX connections type.

QTMR0_TMRn_REG

Periphinmux IDs.

SOC_GLUE_XOR0_INn_REG

QTMR1_TMRn_REG

ACMP0_TRIGn_REG

LPADC0_TRIGn_REG

AON_TRIG_OUTn_REG

SOC_GLUE_CMPPADS_PCTRL_XOR_IN0_REG

SOC_GLUE_CTRLPADS_PCTRL_XOR_IN0_REG

LC_ROT_SOC_LOGIC_IN_REG

LCSENSE_SEQ_PTRIG_GLUE_IN_REG

LCSENSE_SEQ_TICKS_GLUE_IN_REG

CMP0_SAMPLE_REG

CMP0_RR_TRIG_REG

LPI2C0_TRIG_REG

LPUART0_REG

PMUX_SHIFT

FSL_INPUTMUX_DRIVER_VERSION

Group interrupt driver version for SDK.

void INPUTMUX_Init(void *base)

Initialize INPUTMUX peripheral.

This function enables the INPUTMUX clock.

Parameters

- `base` – Base address of the INPUTMUX peripheral.

Return values

None. –

void INPUTMUX_AttachSignal(void *base, uint32_t index, *inputmux_connection_t* connection)

Attaches a signal.

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
```

In this example, INTMUX has 8 registers for PINT, PINT_SEL0~PINT_SEL7. With parameter `index` specified as 2, this function configures register PINT_SEL2.

Parameters

- `base` – Base address of the INPUTMUX peripheral.
- `index` – The serial number of destination register in the group of INPUTMUX registers with same name.
- `connection` – Applies signal from source signals collection to target signal.

Return values

None. –

void INPUTMUX_Deinit(void *base)

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

Parameters

- `base` – Base address of the INPUTMUX peripheral.

Return values

None. –

2.13 IPMQ Driver

enum `_ipmq_ret_value`

Defines the enumeration type for IPMQ return values.

This enumeration type defines various return values for IPMQ message queue operations, indicating the status of these operations.

Values:

enumerator `kIPMQ_Ret_Error`

Operation error

enumerator `kIPMQ_Ret_ReceiveUnkownMsg`

Received unknown message

enumerator `kIPMQ_Ret_ReceiveOpcodeMismatch`

Received a message with mismatched opcode

enumerator `kIPMQ_Ret_ReceiveUserDataMismatch`

Received a message with mismatched user data

enumerator `kIPMQ_Ret_ReceiveAck`

Successfully received an acknowledgement message

enumerator `kIPMQ_Ret_ReceiveNACK`

Successfully received a negative acknowledgement message

enum `_ipmq_msg_type`

Defines an enumeration for IPMQ message types.

This enumeration defines various message types within the message queue, used to distinguish different categories of messages.

Values:

enumerator `kIPMQ_MsgType_Message`

Regular message type

enumerator `kIPMQ_MsgType_ACK`

Acknowledgement message type

enumerator `kIPMQ_MsgType_NACK`

Negative acknowledgement message type

enumerator `kIPMQ_MsgType_Reserved`

Reserved message type for future use

enum `_ipmq_msg_opcode`

Enumeration of IPMQ Message Opcodes.

This enumeration lists the different message opcodes supported by the IPMQ messaging system, which define the specific actions or operations to be performed on the target device.

Values:

enumerator `kIPMQ_MsgOpcode_BootReason`

Opcode for querying the reason of boot

enumerator `kIPMQ_MsgOpcode_DataLoad`

Opcode for loading data

enumerator `kIPMQ_MsgOpcode_AdvcEnable`

Opcode to enable ADVC functionality

enumerator `kIPMQ_MsgOpcode_RegisterWrite`

Opcode to write to a register

enumerator kIPMQ_MsgOpcode_RegisterRead
Opcode to read from a register

enumerator kIPMQ_MsgOpcode_GpioInit
Opcode to initialize GPIO

enumerator kIPMQ_MsgOpcode_GpioSet
Opcode to set GPIO state

enumerator kIPMQ_MsgOpcode_GpioGet
Opcode to get GPIO state

enumerator kIPMQ_MsgOpcode_GpioDeinit
Opcode to deinitialize GPIO

enumerator kIPMQ_MsgOpcode_UartInit
Opcode to initialize UART

enumerator kIPMQ_MsgOpcode_UartPinsSelect
Opcode to select UART pins

enumerator kIPMQ_MsgOpcode_UartRbInit
Opcode to initialize UART ring buffer

enumerator kIPMQ_MsgOpcode_UartRead
Opcode to read from UART

enumerator kIPMQ_MsgOpcode_UartRbDeinit
Opcode to deinitialize UART ring buffer

enumerator kIPMQ_MsgOpcode_UartDeinit
Opcode to deinitialize UART

enumerator kIPMQ_MsgOpcode_I2cPinsSelect
Opcode to select I2C pins

enumerator kIPMQ_MsgOpcode_I2cMasterInit
Opcode to initialize I2C master

enumerator kIPMQ_MsgOpcode_I2cMasterCreatehandle
Opcode to create an I2C master handle

enumerator kIPMQ_MsgOpcode_I2cMasterTransfernonblocking
Opcode for non-blocking I2C master transfer

enumerator kIPMQ_MsgOpcode_I2cMasterDeinit
Opcode to deinitialize I2C master

enumerator kIPMQ_MsgOpcode_I2cSlaveInit
Opcode to initialize I2C slave

enumerator kIPMQ_MsgOpcode_I2cSlaveCreatehandle
Opcode to create an I2C slave handle

enumerator kIPMQ_MsgOpcode_I2cSlaveTransfernonblocking
Opcode for non-blocking I2C slave transfer

enumerator kIPMQ_MsgOpcode_I2cRead
Opcode to read from I2C

enumerator kIPMQ_MsgOpcode_I2cSlaveDeinit
Opcode to deinitialize I2C slave

enumerator kIPMQ_MsgOpcode_Ping
Opcode for ping operation

enumerator kIPMQ_MsgOpcode_ApiDeinit
Opcode to deinitialize API

enumerator kIPMQ_MsgOpcode_RegisterReadModifyWrite
Opcode for reading, modifying, and writing to a register

enumerator kIPMQ_MsgOpcode_RtcIsAlive
Opcode to check if RTC is alive

enumerator kIPMQ_MsgOpcode_RtcSetAlarm
Opcode to set RTC alarm

enumerator kIPMQ_MsgOpcode_RtcClearAlarm
Opcode to clear RTC alarm

enumerator kIPMQ_MsgOpcode_RtcGetCounter
Opcode to get RTC counter value

enumerator kIPMQ_MsgOpcode_RtcLiveDetectorEnable
Opcode to enable RTC live detector

enumerator kIPMQ_MsgOpcode_RtcAnalogDone
Opcode for RTC analog done event

enumerator kIPMQ_MsgOpcode_SecuritySet
Opcode to set security settings

enumerator kIPMQ_MsgOpcode_ComparatorInit
Opcode to initialize comparator

enumerator kIPMQ_MsgOpcode_ComparatorDeinit
Opcode to deinitialize comparator

enumerator kIPMQ_MsgOpcode_ComparatorInterruptEn
Opcode to enable comparator interrupt

enumerator kIPMQ_MsgOpcode_MemoryRead
Opcode to read memory

enumerator kIPMQ_MsgOpcode_MemoryWrite
Opcode to write to memory

enumerator kIPMQ_MsgOpcode_MemoryLock
Opcode to lock memory

enumerator kIPMQ_MsgOpcode_DebugEn
Opcode to enable debug mode

enumerator kIPMQ_MsgOpcode_AdvcEn
Opcode to enable ADV feature

enumerator kIPMQ_MsgOpcode_AdvcPreVoltageRequest
Opcode for ADV pre-voltage request

enumerator kIPMQ_MsgOpcode_AdvcPostChangeFrequencyVoltageRequest
Opcode for ADV post-change frequency voltage request

enumerator kIPMQ_MsgOpcode_UpdateTemperature
Opcode to update temperature

enumerator kIPMQ_MsgOpcode_DeepPowerDown

Opcode to enter deep power down mode

enumerator kIPMQ_MsgOpcode_DeepPowerDownAbort

Opcode to abort deep power down mode

enumerator kIPMQ_MsgOpcode_RomDone

Opcode for ROM done event

enumerator kIPMQ_MsgOpcode_UartMode

Opcode to configure UART mode

enumerator kIPMQ_MsgOpcode_LogReset

Opcode to reset logs

enumerator kIPMQ_MsgOpcode_LogDump

Opcode to dump logs

enumerator kIPMQ_MsgOpcode_WakeupSource

Opcode related to wakeup source

enumerator kIPMQ_MsgOpcode_CommandsStub

Opcode for commands stub

enumerator kIPMQ_MsgOpcode_UartWrite

Opcode to write to UART

enum _ipmq_msg_general_purpose_interrupt

Enumeration of IPMQ General Purpose Interrupts.

This enumeration defines the general-purpose interrupts used in the IPMQ (Inter-Process Messaging Queue). These interrupts are triggered when specific events occur within the AON domain.

Values:

enumerator kIPMQ_MsgGpi_NewMsg

Interrupt for new message arrival

enumerator kIPMQ_MsgGpi_Error

Interrupt for error occurrence

typedef enum _ipmq_ret_value ipmq_ret_value_t

Defines the enumeration type for IPMQ return values.

This enumeration type defines various return values for IPMQ message queue operations, indicating the status of these operations.

typedef enum _ipmq_msg_type ipmq_msg_type_t

Defines an enumeration for IPMQ message types.

This enumeration defines various message types within the message queue, used to distinguish different categories of messages.

typedef enum _ipmq_msg_opcode ipmq_msg_opcode_t

Enumeration of IPMQ Message Opcodes.

This enumeration lists the different message opcodes supported by the IPMQ messaging system, which define the specific actions or operations to be performed on the target device.

typedef enum _ipmq_msg_general_purpose_interrupt ipmq_msg_general_purpose_interrupt_t

Enumeration of IPMQ General Purpose Interrupts.

This enumeration defines the general-purpose interrupts used in the IPMQ (Inter-Process Messaging Queue). These interrupts are triggered when specific events occur within the AON domain.

```
typedef struct ipmq_interrupt_cb ipmq_interrupt_cb_t
```

Structure definition for IPMQ Interrupt Callbacks.

This structure holds function pointers for callback functions to be invoked when specific IPMQ interrupt events occur.

- `error()`: A pointer to a function that will be called when an error interrupt occurs within the AON domain.

```
typedef struct ipmq_config ipmq_config_t
```

IP Message Queue Configuration Structure.

This structure is used to configure the behavior and parameters for the inter-processor message queue.

```
ipmq_interrupt_cb_t g_ipmCabllbacks
```

```
FSL_IPMQ_DRIVER_VERSION
```

ipmq driver version 2.0.0

```
IPMQ_MSG_HEADER_SYNC_CODE
```

```
IPMQ_MSG_PHY_CHANNEL_COUNT
```

```
IPMQ_USELESS_VALUE
```

```
IPMQ_MSG_GPI_COUNT
```

```
void IPMQ_Init(const ipmq_config_t *ptrConfig)
```

Initialize inter-processor message queue.

Note: Once initialized, please do not invoke this API repeatedly.

Note: If **recoveryQueueBeforeInit** is set as **true**, this function execution might involve a considerable duration due to the presence of an inlined delay function.

Parameters

- `ptrConfig` – **[in]** Pointer to `ipmq_config_t`, used to set basic configuration of Message queue.

```
void IPMQ_Deinit(void)
```

De-initialize inter-processor message queue.

Note: This function should be invoked only after the IPMQ has been successfully initialized.

```
void IPMQ_RecoveryQueue(uint32_t coreClock_Hz)
```

Establish synchronization between the CM33 and AON domains through the recovery message queue mechanism.

Used when an executable file is loaded directly into RAM, in which case the RAM executable resets local pointers while the AON ROM remains unaware

Note: Please note that this function execution might involve a considerable duration due to the presence of an inlined delay function.

Parameters

- `coreClock_Hz` – **[in]** The core clock frequency to delay servial micro seconds.

`void IPMQ_EnableMsgGeneralPurposeInterrupts(uint32_t mask)`

Enable message general purpose interrupt.

Parameters

- `mask` – **[in]** The mask of interrupts to enable, should be the OR'ed value of `ipmq_msg_general_purpose_interrupt_t`.

`void IPMQ_DisableMsgGeneralPurposeInterrupts(uint32_t mask)`

Disable message general purpose interrupt.

Parameters

- `mask` – **[in]** The mask of interrupts to disable, should be the OR'ed value of `ipmq_msg_general_purpose_interrupt_t`.

`void IPMQ_SendMessage(ipmq_msg_opcode_t opcode, uint8_t userData, uint8_t payloadSize, uint32_t *ptrPayload)`

Send message to AON with payload from single memory location.

Parameters

- `opcode` – **[in]** Message opcode, please refer to `ipmq_msg_opcode_t`.
- `userData` – **[in]** User data send with the message, return with the ACK/NACK feedback, can be used by application to match feedback to the message.
- `payloadSize` – **[in]** Size of payload to send.
- `ptrPayload` – **[in]** Pointer to the buffer of payload to send.

`void IPMQ_SendMessage2Payload(ipmq_msg_opcode_t opcode, uint8_t userData, uint8_t payload1Size, uint32_t *ptr1stPayload, uint8_t payload2Size, uint32_t *ptr2ndPayload)`

Send message to AON with two different payload from different memory location, for example the first payload may from RAM, and the second payload may from flash.

Parameters

- `opcode` – **[in]** Message opcode, please refer to `ipmq_msg_opcode_t`.
- `userData` – **[in]** User data send with the message, return with the ACK/NACK feedback, can be used by application to match feedback to the message.
- `payload1Size` – **[in]** Size of first payload to send.
- `ptr1stPayload` – **[in]** Pointer to the buffer of the first payload to send.
- `payload2Size` – **[in]** Size of second payload to send.
- `ptr2ndPayload` – **[in]** Pointer to the buffer of the second payload to send.

`ipmq_ret_value_t IPMQ_ReceiveMessage(ipmq_msg_opcode_t opcode, uint8_t userData, uint32_t *ptrResponsePayload, uint8_t *ptrResponsePayloadSize)`

Block receiving message until error detected or message finished.

Note: In case the return value of this function is anything other than `kIPMQ_Ret_ReceiveAck`, it is recommended for the user to examine the sent message according to the reported error and resend the message accordingly

Parameters

- opcode – **[in]** Message opcode, please refer to ipmq_msg_opcode_t.
- userData – **[in]** User data send with the message, return with the ACK/NACK feedback, can be used by application to match feedback to the message.
- ptrResponsePayload – **[out]** Pointer to the buffer to store response message's payload.
- ptrResponsePayloadSize – **[out]** Pointer to the variable to store the size of response message's payload.

Return values

- kIPMQ_Ret_ReceiveUnkownMsg – An unknown message was received
- kIPMQ_Ret_ReceiveOpcodeMismatch – The received message does not match the expected message header
- kIPMQ_Ret_ReceiveUserDataMismatch – The received user data does not match the input value.
- kIPMQ_Ret_ReceiveAck – The message was received successfully and an acknowledgment (ACK) is returned
- kIPMQ_Ret_ReceiveNACK – The message was received but there was a failure, a negative acknowledgment (NACK) is returned, and the NACK reason may contain in payload.

struct _ipmq_interrupt_cb

#include <fsl_ipmq.h> Structure definition for IPMQ Interrupt Callbacks.

This structure holds function pointers for callback functions to be invoked when specific IPMQ interrupt events occur.

- error(): A pointer to a function that will be called when an error interrupt occurs within the AON domain.

Public Members

void (*error)(void)

Callback function for error interrupts

struct _ipmq_config

#include <fsl_ipmq.h> IP Message Queue Configuration Structure.

This structure is used to configure the behavior and parameters for the inter-processor message queue.

Public Members

bool receiveByPolling

Whether to receive messages through polling, if set as **false** the kIPMQ_MsgGpi_NewMsg will be enabled, if set as **true** the code will polling the MU general purpose status.

bool enableErrorInterrupt

Whether to enable/disable error interrupts, if enabled please note as set error callback function in ptrCb.

`bool recoveryQueueBeforeInit`

Whether to recover the message queue before initialization. This option is used when an executable file is loaded directly into RAM, in which case the RAM executable re-sets local pointers while the AON ROM remains unaware. Set this to **true** to enable recovery.

`uint32_t coreClock_Hz`

The core clock frequency required to recover the queue before initialization, if `recoveryQueueBeforeInit` is set as false, this member will not be used.

`ipmq_interrupt_cb_t *ptrCb`

Pointer to the interrupt callback functions.

2.14 KPP: KeyPad Port Driver

`void KPP_Init(KPP_Type *base, kpp_config_t *configure)`

KPP initialize. This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

- `base` – KPP peripheral base address.
- `configure` – The KPP configuration structure pointer.

`void KPP_Deinit(KPP_Type *base)`

Deinitializes the KPP module and gates the clock. This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

- `base` – KPP peripheral base address.

`static inline void KPP_EnableInterrupts(KPP_Type *base, uint16_t mask)`

Enable the interrupt.

Parameters

- `base` – KPP peripheral base address.
- `mask` – KPP interrupts to enable. This is a logical OR of the enumeration :: `kpp_interrupt_enable_t`.

`static inline void KPP_DisableInterrupts(KPP_Type *base, uint16_t mask)`

Disable the interrupt.

Parameters

- `base` – KPP peripheral base address.
- `mask` – KPP interrupts to disable. This is a logical OR of the enumeration :: `kpp_interrupt_enable_t`.

`static inline uint16_t KPP_GetStatusFlag(KPP_Type *base)`

Gets the KPP interrupt event status.

Parameters

- `base` – KPP peripheral base address.

Returns

The status of the KPP. Application can use the enum type in the "`kpp_interrupt_enable_t`" to get the right status of the related event.

```
static inline void KPP_ClearStatusFlag(KPP_Type *base, uint16_t mask)
```

Clears KPP status flag.

Parameters

- base – KPP peripheral base address.
- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: `kpp_interrupt_enable_t`.

```
static inline void KPP_SetSynchronizeChain(KPP_Type *base, uint16_t mask)
```

Set KPP synchronization chain.

Parameters

- base – KPP peripheral base address.
- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: `kpp_sync_operation_t`.

```
void KPP_keyPressScanning(KPP_Type *base, uint8_t *data, uint32_t clockSrc_Hz)
```

Keypad press scanning.

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

Parameters

- base – KPP peripheral base address.
- data – KPP key press scanning data. The data buffer should be prepared with length at least equal to `KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX`. the data pointer is recommended to be a array like `uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]`. for example the `data[2] = 4`, that means in column 1 row 2 has a key press event.
- clockSrc_Hz – Source clock.

```
FSL_KPP_DRIVER_VERSION
```

KPP driver version.

```
enum _kpp_interrupt_enable
```

List of interrupts supported by the peripheral. This enumeration uses one-hot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Values:

```
enumerator kKPP_keyDepressInterrupt
```

Keypad depress interrupt source

```
enumerator kKPP_keyReleaseInterrupt
```

Keypad release interrupt source

```
enum _kpp_sync_operation
```

Lists of KPP synchronize chain operation.

Values:

```
enumerator kKPP_ClearKeyDepressSyncChain
```

Keypad depress interrupt status.

```
enumerator kKPP_SetKeyReleasesSyncChain
```

Keypad release interrupt status.

`typedef enum _kpp_interrupt_enable kpp_interrupt_enable_t`

List of interrupts supported by the peripheral. This enumeration uses one-hot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

`typedef enum _kpp_sync_operation kpp_sync_operation_t`

Lists of KPP synchronize chain operation.

`typedef struct _kpp_config kpp_config_t`

Lists of KPP status.

`KPP_KEYPAD_COLUMNNUM_MAX`

`KPP_KEYPAD_ROWNUM_MAX`

`struct _kpp_config`

`#include <fsl_kpp.h>` Lists of KPP status.

Public Members

`uint8_t activeRow`

The row number: bit 7 ~ 0 represents the row 7 ~ 0.

`uint8_t activeColumn`

The column number: bit 7 ~ 0 represents the column 7 ~ 0.

`uint16_t interrupt`

KPP interrupt source. A logical OR of “kpp_interrupt_enable_t”.

2.15 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_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*.

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI

Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART

Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C

Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C

Group number for LPI2C status codes.

enumerator kStatusGroup_UART

Group number for UART status codes.

enumerator kStatusGroup_I2C

Group number for I2C status codes.

enumerator kStatusGroup_LPSCI

Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART

Group number for LPUART status codes.

enumerator kStatusGroup_SPI

Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPIDSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator `kStatusGroup_SMARTCARD`
Group number for SMARTCARD status codes.

enumerator `kStatusGroup_LMEM`
Group number for LMEM status codes.

enumerator `kStatusGroup_QSPI`
Group number for QSPI status codes.

enumerator `kStatusGroup_DMA`
Group number for DMA status codes.

enumerator `kStatusGroup_EDMA`
Group number for EDMA status codes.

enumerator `kStatusGroup_DMAMGR`
Group number for DMAMGR status codes.

enumerator `kStatusGroup_FLEXCAN`
Group number for FlexCAN status codes.

enumerator `kStatusGroup_LTC`
Group number for LTC status codes.

enumerator `kStatusGroup_FLEXIO_CAMERA`
Group number for FLEXIO CAMERA status codes.

enumerator `kStatusGroup_LPC_SPI`
Group number for LPC_SPI status codes.

enumerator `kStatusGroup_LPC_USART`
Group number for LPC_USART status codes.

enumerator `kStatusGroup_DMIC`
Group number for DMIC status codes.

enumerator `kStatusGroup_SDIF`
Group number for SDIF status codes.

enumerator `kStatusGroup_SPIFI`
Group number for SPIFI status codes.

enumerator `kStatusGroup_OTP`
Group number for OTP status codes.

enumerator `kStatusGroup_MCAN`
Group number for MCAN status codes.

enumerator `kStatusGroup_CAAM`
Group number for CAAM status codes.

enumerator `kStatusGroup_ECSPI`
Group number for ECSPI status codes.

enumerator `kStatusGroup_USDHC`
Group number for USDHC status codes.

enumerator `kStatusGroup_LPC_I2C`
Group number for LPC_I2C status codes.

enumerator `kStatusGroup_DCP`
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_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_CODECD
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_SDIOISLV
Group number for SDIOISLV 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 will be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, “SUPPRESS_FALL_THROUGH_WARNING();” need to be added at the end of each case section which misses “break;”statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.16 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-0xFFFFFFFFUL.

```
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 hysteresis 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.

2.17 LPI2C: Low Power Inter-Integrated Circuit Driver

void LPI2C_DriverIRQHandler(uint32_t instance)

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

Parameters

- instance – LPI2C instance.

FSL_LPI2C_DRIVER_VERSION

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy

The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle

The slave driver is idle.

enumerator kStatus_LPI2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail

DMA request failed.

enumerator kStatus_LPI2C_Timeout

Timeout polling status flags.

IRQn_Type const kLpi2cIrqs[]

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

lpi2c_master_isr_t s_lpi2cMasterIsr

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

void *s_lpi2cMasterHandle[]

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

uint32_t LPI2C_GetInstance(LPI2C_Type *base)

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- base – The LPI2C peripheral base address.

Returns

LPI2C instance number starting from 0.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.18 LPI2C Master Driver

`void LPI2C_MasterGetDefaultConfig(lpi2c_master_config_t *masterConfig)`

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig          = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns = 0;
masterConfig->pinLowTimeout_ns  = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable = false;
masterConfig->hostRequest.source  = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `lpi2c_master_config_t`.

`void LPI2C_MasterInit(LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t sourceClock_Hz)`

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `LPI2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`void LPI2C_MasterDeinit(LPI2C_Type *base)`

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

`void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const lpi2c_data_match_config_t *matchConfig)`

Configures LPI2C master data match feature.

Parameters

- `base` – The LPI2C peripheral base address.
- `matchConfig` – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

Convert provided flags to status code, and clear any errors if present.

Parameters

- base – The LPI2C peripheral base address.
- status – Current status flags value that will be checked.

Return values

- kStatus_Success –
- kStatus_LPI2C_PinLowTimeout –
- kStatus_LPI2C_ArbitrationLost –
- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_master_flags`

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

`_lpi2c_master_flags`.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_MasterGetStatusFlags()`.

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)
Enables or disables LPI2C master DMA requests.

Parameters

- `base` – The LPI2C peripheral base address.
- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)
Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)
Gets LPI2C master receive data register address for DMA transfer.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)
Sets the watermarks for LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txWords` – Transmit FIFO watermark value in words. The `kLPI2C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO is equal or less than `txWords`. Writing a value equal or greater than the FIFO size is truncated.
- `rxWords` – Receive FIFO watermark value in words. The `kLPI2C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO is greater than `rxWords`. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)
Gets the current number of words in the LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t
                             baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

```
static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)
```

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

```
static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address,
                                                  lpi2c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or over run.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.

- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterStop(LPI2C_Type *base)`

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, lpi2c_master_transfer_t *transfer)`

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The LPI2C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, lpi2c_master_handle_t *handle, lpi2c_master_transfer_callback_t callback, void *userData)`

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *size_t* *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags

Errors to check for.

enum _lpi2c_direction

Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write

Master transmit.

enumerator kLPI2C_Read

Master receive.

enum _lpi2c_master_pin_config

LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source

LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin

Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger

Select the input trigger as the host request input

enum _lpi2c_host_request_polarity

LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow

Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh

Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode

LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled

LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1

LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1

LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1

LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1

LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1

LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1

LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Values:

enumerator kLPI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag

Don't send a stop condition.


```
typedef enum _lpi2c_direction lpi2c_direction_t
```

Direction of master and slave transfers.

```
typedef enum _lpi2c_master_pin_config lpi2c_master_pin_config_t
```

LPI2C pin configuration.

```
typedef enum _lpi2c_host_request_source lpi2c_host_request_source_t
```

LPI2C master host request selection.

```
typedef enum _lpi2c_host_request_polarity lpi2c_host_request_polarity_t
```

LPI2C master host request pin polarity configuration.

```
typedef struct _lpi2c_master_config lpi2c_master_config_t
```

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_data_match_config_mode lpi2c_data_match_config_mode_t
```

LPI2C master data match configuration modes.

```
typedef struct _lpi2c_match_config lpi2c_data_match_config_t
```

LPI2C master data match configuration structure.

```
typedef struct _lpi2c_master_transfer lpi2c_master_transfer_t
```

LPI2C master descriptor of the transfer.

```
typedef struct _lpi2c_master_handle lpi2c_master_handle_t
```

LPI2C master handle of the transfer.

```
typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)
```

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to `LPI2C_MasterTransferCreateHandle()`.

Param base

The LPI2C peripheral base address.

Param handle

Pointer to the LPI2C master driver handle.

Param completionStatus

Either `kStatus_Success` or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest

Host request options.

struct *_lpi2c_match_config*

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32_t match0

Match value 0.

uint32_t match1

Match value 1.

struct *_lpi2c_master_transfer*

#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration `_lpi2c_master_transfer_flags` for available options. Set to 0 or `kLPI2C_TransferDefaultFlag` for normal transfers.

uint16_t slaveAddress

The 7-bit slave address.

lpi2c_direction_t direction

Either `kLPI2C_Read` or `kLPI2C_Write`.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

struct `_lpi2c_master_handle`

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint16_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: `Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]`

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

struct hostRequest

Public Members

`bool enable`

Enable host request.

`lpi2c_host_request_source_t source`

Host request source.

`lpi2c_host_request_polarity_t polarity`

Host request pin polarity.

2.19 LPI2C Master DMA Driver

```
void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   edma_handle_t *rxDmaHandle, edma_handle_t
                                   *txDmaHandle, lpi2c_master_edma_transfer_callback_t
                                   callback, void *userData)
```

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbortEDMA()` API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – **[out]** Pointer to the LPI2C master driver handle.
- *rxDmaHandle* – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_LPI2C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                           *handle, size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

```
status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                         *handle)
```

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

```
typedef struct _lpi2c_master_edma_handle lpi2c_master_edma_handle_t
LPI2C master EDMA handle of the transfer.
```

```
typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)
```

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Param base

The LPI2C peripheral base address.

Param handle

Handle associated with the completed transfer.

Param completionStatus

Either kStatus_Success or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_master_edma_handle
    #include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base
LPI2C base pointer.

bool isBusy
Transfer state machine current state.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[20]
LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer
Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback
Callback function pointer.

void *userData
Application data passed to callback.

edma_handle_t *rx
Handle for receive DMA channel.

edma_handle_t *tx
Handle for transmit DMA channel.

edma_tcd_t tcds[3]
Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.20 LPI2C Slave Driver

```
void LPI2C_SlaveGetDefaultConfig(lpi2c_slave_config_t *slaveConfig)
```

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave      = true;
slaveConfig->address0         = 0U;
slaveConfig->address1         = 0U;
slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable = true;
slaveConfig->filterEnable     = true;
slaveConfig->enableGeneralCall = false;
slaveConfig->sclStall.enableAck = false;
slaveConfig->sclStall.enableTx  = true;
slaveConfig->sclStall.enableRx  = true;
```

(continues on next page)

(continued from previous page)

```
slaveConfig->sclStall.enableAddress    = true;
slaveConfig->ignoreAck                 = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns   = 0;
slaveConfig->sclGlitchFilterWidth_ns   = 0;
slaveConfig->dataValidDelay_ns         = 0;
slaveConfig->clockHoldTime_ns          = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with `LPI2C_SlaveInit()`. Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure that is set to default values. Refer to `lpi2c_slave_config_t`.

```
void LPI2C_SlaveInit(LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t
                    sourceClock_Hz)
```

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `LPI2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

```
void LPI2C_SlaveDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveReset(LPI2C_Type *base)
```

Performs a software reset of the LPI2C slave peripheral.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)
```

Enables or disables the LPI2C module as slave.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as slave.

```
static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)
```

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_slave_flags`

Parameters

- `base` – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)
```

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- `kLPI2C_SlaveRepeatedStartDetectFlag`
- `kLPI2C_SlaveStopDetectFlag`
- `kLPI2C_SlaveBitErrFlag`
- `kLPI2C_SlaveFifoErrFlag`

Attempts to clear other flags has no effect.

See also:

`_lpi2c_slave_flags`.

Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_SlaveGetStatusFlags()`.

```
static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Enables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Disables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.


```
static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)
```

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)
```

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

```
static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

```
static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)
```

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.
- ackOrNack – Pass true for an ACK or false for a NAK.

```
static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)
```

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL, false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns

Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns

Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, lpi2c_slave_handle_t *handle, lpi2c_slave_transfer_callback_t callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

```
status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, lpi2c_slave_handle_t *handle,
                                         uint32_t eventMask)
```

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kLPI2C_SlaveTransmitEvent* and *kLPI2C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to *lpi2c_slave_handle_t* structure which stores the transfer state.
- *eventMask* – Bit mask formed by OR'ing together *lpi2c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kLPI2C_SlaveAllEvents* to enable all events.

Return values

- *kStatus_Success* – Slave transfers were successfully started.
- *kStatus_LPI2C_Busy* – Slave transfers have already been started on this handle.

```
status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, lpi2c_slave_handle_t *handle, size_t
                                     *count)
```

Gets the slave transfer status during a non-blocking transfer.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to *i2c_slave_handle_t* structure.
- *count* – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- *kStatus_Success* –
- *kStatus_NoTransferInProgress* –

```
void LPI2C_SlaveTransferAbort(LPI2C_Type *base, lpi2c_slave_handle_t *handle)
```

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- *base* – The LPI2C peripheral base address.

- `handle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, lpi2c_slave_handle_t *handle)`

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`enum _lpi2c_slave_flags`

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- `kLPI2C_SlaveRepeatedStartDetectFlag`
- `kLPI2C_SlaveStopDetectFlag`
- `kLPI2C_SlaveBitErrFlag`
- `kLPI2C_SlaveFifoErrFlag`

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator `kLPI2C_SlaveTxReadyFlag`

Transmit data flag

enumerator `kLPI2C_SlaveRxReadyFlag`

Receive data flag

enumerator `kLPI2C_SlaveAddressValidFlag`

Address valid flag

enumerator `kLPI2C_SlaveTransmitAckFlag`

Transmit ACK flag

enumerator `kLPI2C_SlaveRepeatedStartDetectFlag`

Repeated start detect flag

enumerator `kLPI2C_SlaveStopDetectFlag`

Stop detect flag

enumerator `kLPI2C_SlaveBitErrFlag`

Bit error flag

enumerator `kLPI2C_SlaveFifoErrFlag`

FIFO error flag

enumerator `kLPI2C_SlaveAddressMatch0Flag`

Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag

Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag

General call flag

enumerator kLPI2C_SlaveBusyFlag

Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag

Bus busy flag

enumerator kLPI2C_SlaveClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags

Errors to check for.

enum _lpi2c_slave_address_match

LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0

Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

LPI2C slave handle structure.

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base

Base address for the LPI2C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

struct *_lpi2c_slave_config*

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

`bool enableSlave`
Enable slave mode.

`uint8_t address0`
Slave's 7-bit address.

`uint8_t address1`
Alternate slave 7-bit address.

`lpi2c_slave_address_match_t addressMatchMode`
Address matching options.

`bool filterDozeEnable`
Enable digital glitch filter in doze mode.

`bool filterEnable`
Enable digital glitch filter.

`bool enableGeneralCall`
Enable general call address matching.

`struct _lpi2c_slave_config sclStall`
SCL stall enable options.

`bool ignoreAck`
Continue transfers after a NACK is detected.

`bool enableReceivedAddressRead`
Enable reading the address received address as the first byte of data.

`uint32_t sdaGlitchFilterWidth_ns`
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

`uint32_t sclGlitchFilterWidth_ns`
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

`uint32_t dataValidDelay_ns`
Width in nanoseconds of the data valid delay.

`uint32_t clockHoldTime_ns`
Width in nanoseconds of the clock hold time.

`struct _lpi2c_slave_transfer`
#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

`lpi2c_slave_transfer_event_t event`
Reason the callback is being invoked.

`uint8_t receivedAddress`
Matching address send by master.

`uint8_t *data`
Transfer buffer

`size_t dataSize`
Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct *_lpi2c_slave_handle*

#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer

LPI2C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck

Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDatasCLStall or enableAddressSCLStall.

bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

2.21 LPSPI: Low Power Serial Peripheral Interface

2.22 LPSPI Peripheral driver

```
void LPSPI_MasterInit(LPSPI_Type *base, const lpspi_master_config_t *masterConfig, uint32_t srcClock_Hz)
```

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure *lpspi_master_config_t*.
- srcClock_Hz – Module source input clock in Hertz

```
void LPSPI_MasterGetDefaultConfig(lpspi_master_config_t *masterConfig)
```

Sets the *lpspi_master_config_t* structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;  
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to *lpspi_master_config_t* structure

```
void LPSPI_SlaveInit(LPSPI_Type *base, const lpspi_slave_config_t *slaveConfig)
```

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure *lpspi_slave_config_t*.

```
void LPSPI_SlaveGetDefaultConfig(lpspi_slave_config_t *slaveConfig)
```

Sets the *lpspi_slave_config_t* structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;  
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to *lpspi_slave_config_t* structure.

```
void LPSPI_Deinit(LPSPI_Type *base)
```

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

```
void LPSPI_Reset(LPSPI_Type *base)
```

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag | kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type `_lpspi_flags`.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

```
static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

```
static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

```
bool LPSPI_CheckTransferArgument(LPSPI_Type *base, lpspi_transfer_t *transfer, bool isEdma)
```

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

```
static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, lpspi_master_slave_mode_t mode)
```

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type `lpspi_master_slave_mode_t`.

```
static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, lpspi_which_pcs_t select)
```

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.

- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI_SetPCSContinuous(LPSPI_Type *base, bool IsContinuous)

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinuous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum `_lpspi_pcs_polarity`.

```
static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)
```

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPi in slave mode, although the command register does not update until after the LPSPi is enabled. After it is enabled, the transmit command register should only be changed if the LPSPi is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

```
uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t  
srcClock_Hz, uint32_t *tcrPrescaleValue)
```

Sets the LPSPi baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale `tcrPrescaleValue` parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPi module must first be disabled before configuring this. Note that the LPSPi module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPi is not configured for master mode or if the LPSPi module is not disabled.

```
void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, lpspi_delay_type_t
                                whichDelay)
```

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).
- whichDelay – The desired delay to configure, must be of type `lpspi_delay_type_t`.

```
uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec,
                                    lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)
```

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the $\text{delayTime} = \text{LPSPI\_clockSource} / (\text{PRESCALE} * \text{Delay\_scaler})$.

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nano-seconds.
- whichDelay – The desired delay to configuration, which must be of type `lpspi_delay_type_t`.
- srcClock_Hz – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

```
static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)
```

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result

in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.
- data – The data word to be sent.

```
static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)
```

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The data read from the data buffer.

```
void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)
```

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

```
void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, lpspi_master_handle_t *handle,  
                                     lpspi_master_transfer_callback_t callback, void  
                                     *userData)
```

Initializes the LPSPI master handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to `lpspi_master_handle_t`.
- callback – DSPI callback.
- userData – callback function parameter.

```
status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, lpspi_transfer_t *transfer)
```

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferNonBlocking(*LPSPI_Type* *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.
- transfer – pointer to *lpspi_transfer_t* structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferGetCount(*LPSPI_Type* *base, *lpspi_master_handle_t* *handle, *size_t* *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

void LPSPI_MasterTransferAbort(*LPSPI_Type* *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(*LPSPI_Type* *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.

```
void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, lpspi_slave_handle_t *handle,  
                                     lpspi_slave_transfer_callback_t callback, void *userData)
```

Initializes the LPSPI slave handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to *lpspi_slave_handle_t*.
- callback – DSPI callback.
- userData – callback function parameter.

```
status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, lpspi_slave_handle_t *handle,  
                                         lpspi_transfer_t *transfer)
```

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.
- transfer – pointer to *lpspi_transfer_t* structure.

Returns

status of *status_t*.

```
status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, lpspi_slave_handle_t *handle, size_t  
                                     *count)
```

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of *status_t*.

```
void LPSPI_SlaveTransferAbort(LPSPI_Type *base, lpspi_slave_handle_t *handle)
```

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns

true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum *_lpspi_flags*

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag

Data Match flag

enumerator kLPSPI_ModuleBusyFlag

Module Busy flag

enumerator kLPSPI_AllStatusFlag

Used for clearing all w1c status flags

enum _lpspi_interrupt_enable

LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable

Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable

Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable

Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable

Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable

Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable

Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable

Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable

Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable

All above interrupts enable.

enum _lpspi_dma_enable

LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable

Receive data DMA enable

enum _lpspi_master_slave_mode

LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master

LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave

LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0

PCS[0]

enumerator kLPSPI_Pcs1

PCS[1]

enumerator kLPSPI_Pcs2

PCS[2]

enumerator kLPSPI_Pcs3

PCS[3]

enum _lpspi_pcs_polarity_config

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh

PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow

PCS Active Low (idles high)

enum _lpspi_pcs_polarity

LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow

Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow

Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow

Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow

Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow

Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity

LPSPI clock polarity configuration.

Values:

enumerator kLPSPIClockPolarityActiveHigh

CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPIClockPolarityActiveLow

CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase

LPSPI clock phase configuration.

Values:

enumerator kLPSPIClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPIClockPhaseSecondEdge

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 kLPSPIMsbFirst

Data transfers start with most significant bit.

enumerator kLPSPILsbFirst

Data transfers start with least significant bit.

enum _lpspi_host_request_select

LPSPI Host Request select configuration.

Values:

enumerator kLPSPICHostReqExtPin

Host Request is an ext pin.

enumerator kLPSPICHostReqInternalTrigger

Host Request is an internal trigger.

enum _lpspi_match_config

LPSPI Match configuration options.

Values:

enumerator kLPSPIMatchDisabled

LPSPI Match Disabled.

enumerator kLPSPIC1stWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSPICAnyWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSPIC1stWordEqualsM0and2ndWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSPICAnyWordEqualsM0andNxtWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordAndM1EqualsM0andM1
LPSPi Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1
LPSPi Match Enabled.

enum _lpspi_pin_config
LPSPi pin (SDO and SDI) configuration.

Values:

enumerator kLPSPI_SdiInSdoOut
LPSPi SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut
LPSPi SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut
LPSPi SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut
LPSPi SDO input, SDI output.

enum _lpspi_data_out_config
LPSPi data output configuration.

Values:

enumerator kLpspiDataOutRetained
Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate
Data out is tristated when chip select is de-asserted

enum _lpspi_transfer_width
LPSPi transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPi delay type selection.

Values:

enumerator kLPSPI_PcsToSck
PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
Delay between transfers.

enum _lpspi_transfer_config_flag_for_master

Use this enumeration for LPSPi master transfer configFlags.

Values:

enumerator kLPSPi_MasterPcs0

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS0 signal

enumerator kLPSPi_MasterPcs1

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS1 signal

enumerator kLPSPi_MasterPcs2

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS2 signal

enumerator kLPSPi_MasterPcs3

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS3 signal

enumerator kLPSPi_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPi_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi_MasterByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPi slave transfer configFlags.

Values:

enumerator kLPSPi_SlavePcs0

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS0 signal

enumerator kLPSPi_SlavePcs1

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS1 signal

enumerator kLPSPi_SlavePcs2

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS2 signal

enumerator kLPSPi_SlavePcs3

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS3 signal

enumerator kLPSPi_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.

enum `_lpspi_transfer_state`

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator `kLPSPi_Idle`

Nothing in the transmitter/receiver.

enumerator `kLPSPi_Busy`

Transfer queue is not finished.

enumerator `kLPSPi_Error`

Transfer error.

typedef enum `_lpspi_master_slave_mode` `lpspi_master_slave_mode_t`

LPSPI master or slave mode configuration.

typedef enum `_lpspi_which_pcs_config` `lpspi_which_pcs_t`

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum `_lpspi_pcs_polarity_config` `lpspi_pcs_polarity_config_t`

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum `_lpspi_clock_polarity` `lpspi_clock_polarity_t`

LPSPI clock polarity configuration.

typedef enum `_lpspi_clock_phase` `lpspi_clock_phase_t`

LPSPI clock phase configuration.

typedef enum `_lpspi_shift_direction` `lpspi_shift_direction_t`

LPSPI data shifter direction options.

typedef enum `_lpspi_host_request_select` `lpspi_host_request_select_t`

LPSPI Host Request select configuration.

typedef enum `_lpspi_match_config` `lpspi_match_config_t`

LPSPI Match configuration options.

typedef enum `_lpspi_pin_config` `lpspi_pin_config_t`

LPSPI pin (SDO and SDI) configuration.

typedef enum `_lpspi_data_out_config` `lpspi_data_out_config_t`

LPSPI data output configuration.

typedef enum `_lpspi_transfer_width` `lpspi_transfer_width_t`

LPSPI transfer width configuration.

typedef enum `_lpspi_delay_type` `lpspi_delay_type_t`

LPSPI delay type selection.

typedef struct `_lpspi_master_config` `lpspi_master_config_t`

LPSPI master configuration structure.

typedef struct `_lpspi_slave_config` `lpspi_slave_config_t`

LPSPI slave configuration structure.

typedef struct `_lpspi_master_handle` `lpspi_master_handle_t`

Forward declaration of the `_lpspi_master_handle` typedefs.

typedef struct `_lpspi_slave_handle` `lpspi_slave_handle_t`

Forward declaration of the `_lpspi_slave_handle` typedefs.

```
typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t
*handle, status_t status, void *userData)
```

Master completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI master.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle,
status_t status, void *userData)
```

Slave completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI slave.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef struct _lpspi_transfer lpspi_transfer_t
```

LPSPI master/slave transfer structure.

```
volatile uint8_t g_lpspiDummyData[]
```

Global variable for dummy data value setting.

```
LPSPI_DUMMY_DATA
```

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
LPSPI_MASTER_PCS_SHIFT
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_PCS_MASK
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_SHIFT
```

LPSPI slave PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_MASK
```

LPSPI slave PCS shift macro , internal used.

```
struct _lpspi_master_config
```

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

`uint32_t baudRate`

Baud Rate for LPSPI.

`uint32_t bitsPerFrame`

Bits per frame, minimum 8, maximum 4096.

`lpspi_clock_polarity_t cpol`

Clock polarity.

`lpspi_clock_phase_t cpha`

Clock phase.

`lpspi_shift_direction_t direction`

MSB or LSB data shift direction.

`uint32_t pcsToSckDelayInNanoSec`

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`uint32_t lastSckToPcsDelayInNanoSec`

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`uint32_t betweenTransferDelayInNanoSec`

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`lpspi_which_pcs_t whichPcs`

Desired Peripheral Chip Select (PCS).

`lpspi_pcs_polarity_config_t pcsActiveHighOrLow`

Desired PCS active high or low

`lpspi_pin_config_t pinCfg`

Configures which pins are used for input and output data during single bit transfers.

`lpspi_data_out_config_t dataOutConfig`

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

`bool enableInputDelay`

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

`struct __lpspi_slave_config`

`#include <fsl_lpspi.h>` LPSPI slave configuration structure.

Public Members

`uint32_t bitsPerFrame`

Bits per frame, minimum 8, maximum 4096.

`lpspi_clock_polarity_t cpol`

Clock polarity.

`lpspi_clock_phase_t cpha`

Clock phase.

`lpspi_shift_direction_t direction`

MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct *_lpspi_transfer*

#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize

Transfer bytes.

uint32_t configFlags

Transfer transfer configuration flags. Set from *_lpspi_transfer_config_flag_for_master* if the transfer is used for master or *_lpspi_transfer_config_flag_for_slave* enumeration if the transfer is used for slave.

struct *_lpspi_master_handle*

#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame

Number of bytes in each frame

volatile uint16_t frameSize

Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

`volatile uint8_t bytesEachWrite`
Bytes for each write TDR.

`volatile uint8_t bytesEachRead`
Bytes for each read RDR.

`const uint8_t *volatile txData`
Send buffer.

`uint8_t *volatile rxData`
Receive buffer.

`volatile size_t txRemainingByteCount`
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`uint32_t txBuffIfNull`
Used if the txData is NULL.

`volatile uint8_t state`
LPSPI transfer state , `_lpspi_transfer_state`.

`lpspi_master_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

`struct _lpspi_slave_handle`
#include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.

Public Members

`volatile bool isByteSwap`
A flag that whether should byte swap.

`volatile uint8_t fifoSize`
FIFO dataSize.

`volatile uint8_t rxWatermark`
Rx watermark.

`volatile uint8_t bytesEachWrite`
Bytes for each write TDR.

`volatile uint8_t bytesEachRead`
Bytes for each read RDR.

`const uint8_t *volatile txData`
Send buffer.

`uint8_t *volatile rxData`
Receive buffer.

`volatile size_t txRemainingByteCount`
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`volatile uint8_t state`
LPSPI transfer state , `_lpspi_transfer_state`.

`volatile uint32_t errorCount`
Error count for slave transfer.

`lpspi_slave_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

2.23 LPSPI eDMA Driver

`FSL_LPSPi_EDMA_DRIVER_VERSION`

LPSPI EDMA driver version.

`DMA_MAX_TRANSFER_COUNT`

DMA max transfer size.

`typedef struct _lpspi_master_edma_handle lpspi_master_edma_handle_t`

Forward declaration of the `_lpspi_master_edma_handle` typedefs.

`typedef struct _lpspi_slave_edma_handle lpspi_slave_edma_handle_t`

Forward declaration of the `_lpspi_slave_edma_handle` typedefs.

`typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base,
lpspi_master_edma_handle_t *handle, status_t status, void *userData)`

Completion callback function pointer type.

Param base

LPSPI peripheral base address.

Param handle

Pointer to the handle for the LPSPI master.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPi_Type *base,  
lpspi_slave_edma_handle_t *handle, status_t status, void *userData)
```

Completion callback function pointer type.

Param base

LPSPi peripheral base address.

Param handle

Pointer to the handle for the LPSPi slave.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
void LPSPi_MasterTransferCreateHandleEDMA(LPSPi_Type *base, lpspi_master_edma_handle_t  
*handle, lpspi_master_edma_transfer_callback_t  
callback, void *userData, edma_handle_t  
*edmaRxRegToRxDataHandle, edma_handle_t  
*edmaTxDataToTxRegHandle)
```

Initializes the LPSPi master eDMA handle.

This function initializes the LPSPi eDMA handle which can be used for other LPSPi transactional APIs. Usually, for a specified LPSPi instance, call this API once to get the initialized handle.

Note that the LPSPi eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPi peripheral base address.
- handle – LPSPi handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPi callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

```
status_t LPSPi_MasterTransferEDMA(LPSPi_Type *base, lpspi_master_edma_handle_t *handle,  
lpspi_transfer_t *transfer)
```

LPSPi master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPi peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- configFlags – transfer configuration flags. `_lpspi_transfer_config_flag_for_master`.

Return values

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.

Returns

Indicates whether LPSPI master transfer was successful or not.

`status_t` LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure, config field is not used.

Return values

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.
- `kStatus_InvalidArgument` – The transfer structure is invalid.

Returns

Indicates whether LPSPI master transfer was successful or not.

`void` LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *size_t* *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns

status of *status_t*.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, *void* *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for `edmaRxRegToRxDataHandle`.

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to `lpspi_slave_edma_handle_t`.
- callback – LPSPI callback.
- userData – callback function parameter.
- `edmaRxRegToRxDataHandle` – `edmaRxRegToRxDataHandle` pointer to `edma_handle_t`.
- `edmaTxDataToTxRegHandle` – `edmaTxDataToTxRegHandle` pointer to `edma_handle_t`.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size

should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)`

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.

`status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, size_t *count)`

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns

status of `status_t`.

`struct _lpspi_master_edma_handle`

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

`volatile bool isPcsContinuous`

Is PCS continuous in transfer.

`volatile bool isByteSwap`

A flag that whether should byte swap.

`volatile uint8_t fifoSize`

FIFO dataSize.

`volatile uint8_t rxWatermark`

Rx watermark.

`volatile uint8_t bytesEachWrite`

Bytes for each write TDR.

`volatile uint8_t bytesEachRead`
Bytes for each read RDR.

`volatile uint8_t bytesLastRead`
Bytes for last read RDR.

`volatile bool isThereExtraRxBytes`
Is there extra RX byte.

`const uint8_t *volatile txData`
Send buffer.

`uint8_t *volatile rxData`
Receive buffer.

`volatile size_t txRemainingByteCount`
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`edma_tcd_t *lastTimeTCD`
Pointer to the lastTime TCD

`bool isMultiDMATransmit`
Is there multi DMA transmit

`volatile uint8_t dmaTransmitTime`
DMA Transfer times.

`uint32_t lastTimeDataBytes`
DMA transmit last Time data Bytes

`uint32_t dataBytesEveryTime`
Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

`edma_transfer_config_t transferConfigRx`
Config of DMA rx channel.

`edma_transfer_config_t transferConfigTx`
Config of DMA tx channel.

`uint32_t txBuffIfNull`
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`
Used if there is not rxData for DMA purpose.

`uint32_t transmitCommand`
Used to write TCR for DMA purpose.

`volatile uint8_t state`
LPSPI transfer state , `_lpspi_transfer_state`.

```
uint8_t nbytes
    eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
    Completion callback.

void *userData
    Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
    edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
    edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
    SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
    #include <fsl_lpspi_edma.h> LPSPI slave eDMA transfer handle structure used for transac-
    tional API.
```

Public Members

```
volatile bool isByteSwap
    A flag that whether should byte swap.

volatile uint8_t fifoSize
    FIFO dataSize.

volatile uint8_t rxWatermark
    Rx watermark.

volatile uint8_t bytesEachWrite
    Bytes for each write TDR.

volatile uint8_t bytesEachRead
    Bytes for each read RDR.

volatile uint8_t bytesLastRead
    Bytes for last read RDR.

volatile bool isThereExtraRxBytes
    Is there extra RX byte.

uint8_t nbytes
    eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData
    Send buffer.

uint8_t *volatile rxData
    Receive buffer.

volatile size_t txRemainingByteCount
    Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
    Number of bytes remaining to receive.
```

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`uint32_t txBuffIfNull`
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`
Used if there is not rxData for DMA purpose.

`volatile uint8_t state`
LPSPI transfer state.

`uint32_t errorCount`
Error count for slave transfer.

`lpspi_slave_edma_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

`edma_handle_t *edmaRxRegToRxDataHandle`
edma_handle_t handle point used for RxReg to RxData buff

`edma_handle_t *edmaTxDataToTxRegHandle`
edma_handle_t handle point used for TxData to TxReg

`edma_tcd_t lpspiSoftwareTCD[2]`
SoftwareTCD, internal used

2.24 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, which should be equal or greater than 1.

```
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

2.25 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.26 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, false to disable.

```
static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t
address2)
```

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer; otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.

- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART__EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART__SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART__SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART__TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in *lpuart_handle_t*.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART__GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators `_lpuart_flags`. To check for a specific status, compare the return value with enumerators in the `_lpuart_flags`. For example, to check whether the TX is empty:

```
if (kLPUART__TxDataRegEmptyFlag & LPUART__GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the `_lpuart_flags`.

`status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)`

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: `kLPUART_TxDataRegEmptyFlag`, `kLPUART_TransmissionCompleteFlag`, `kLPUART_RxDataRegFullFlag`, `kLPUART_RxActiveFlag`, `kLPUART_NoiseErrorFlag`, `kLPUART_ParityErrorFlag`, `kLPUART_TxFifoEmptyFlag`, `kLPUART_RxFifoEmptyFlag`. Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- `base` – LPUART peripheral base address.
- `mask` – the status flags to be cleared. The user can use the enumerators in the `_lpuart_status_flag_t` to do the OR operation and get the mask.

Return values

- `kStatus_LPUART_FlagCannotClearManually` – The flag can't be cleared by this function but it is cleared automatically by hardware.
- `kStatus_Success` – Status in the mask are cleared.

Returns

0 succeed, others failed.

`void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)`

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the `_lpuart_interrupt_enable`. This example shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- `base` – LPUART peripheral base address.
- `mask` – The interrupts to enable. Logical OR of `_lpuart_interrupt_enable`.

`void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)`

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpuart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- `base` – LPUART peripheral base address.
- `mask` – The interrupts to disable. Logical OR of `_lpuart_interrupt_enable`.

`uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)`

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_lpuart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in

`_lpuart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);  
  
if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)  
{  
    ...  
}
```

Parameters

- `base` – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in `_lpuart_interrupt_enable`.

`static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)`

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- `base` – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

`static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)`

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, `STAT[TDRE]`, to generate DMA requests.

Parameters

- `base` – LPUART peripheral base address.
- `enable` – True to enable, false to disable.

`static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)`

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, `STAT[RDRF]`, to generate DMA requests.

Parameters

- `base` – LPUART peripheral base address.
- `enable` – True to enable, false to disable.

`uint32_t LPUART_GetInstance(LPUART_Type *base)`

Get the LPUART instance from peripheral base address.

Parameters

- `base` – LPUART peripheral base address.

Returns

LPUART instance.

```
static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)
```

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

```
static inline uint8_t LPUART_ReadByte(LPUART_Type *base)
```

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

```
static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)
```

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

```
static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)
```

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

`void LPUART_SendAddress(LPUART_Type *base, uint8_t address)`

Transmit an address frame in 9-bit data mode.

Parameters

- `base` – LPUART peripheral base address.
- `address` – LPUART slave address.

`status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)`

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the data to write.
- `length` – Size of the data to write.

Return values

- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully wrote all data.

`status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)`

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the data to write.
- `length` – Size of the data to write.

Return values

- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully wrote all data.

`status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)`

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the RX register.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the buffer to store the received data.
- `length` – Size of the buffer.

Return values

- `kStatus_LPUART_RxHardwareOverrun` – Receiver overrun happened while receiving data.
- `kStatus_LPUART_NoiseError` – Noise error happened while receiving data.

- `kStatus_LPUART_FramingError` – Framing error happened while receiving data.
- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)`

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- `length` – Size of the buffer.

Return values

- `kStatus_LPUART_RxHardwareOverrun` – Receiver overrun happened while receiving data.
- `kStatus_LPUART_NoiseError` – Noise error happened while receiving data.
- `kStatus_LPUART_FramingError` – Framing error happened while receiving data.
- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`void LPUART_TransferCreateHandle(LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_callback_t callback, void *userData)`

Initializes the LPUART handle.

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

The LPUART driver supports the “background” receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the `LPUART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `callback` – Callback function.
- `userData` – User data.

```
status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, lpuart_handle_t *handle,  
                                         lpuart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

```
void LPUART_TransferStartRingBuffer(LPUART_Type *base, lpuart_handle_t *handle, uint8_t  
                                     *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

```
void LPUART_TransferStopRingBuffer(LPUART_Type *base, lpuart_handle_t *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.

- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, lpuart_handle_t *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, lpuart_handle_t *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter `kStatus_UART_RxIdle`. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to `xfer->data`, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from `xfer->data[5]`. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see `uart_transfer_t`.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_LPUART_RxBusy` – Previous receive request is not finished.
- `kStatus_InvalidArgument` – Invalid argument.

`void LPUART_TransferAbortReceive(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

`status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

`void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)`

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

`void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)`

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

`void LPUART_DriverIRQHandler(uint32_t instance)`

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- `instance` – LPUART instance.

`FSL_LPUART_DRIVER_VERSION`

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator `kStatus_LPUART_TxBusy`

TX busy

enumerator `kStatus_LPUART_RxBusy`

RX busy

enumerator `kStatus_LPUART_TxIdle`

LPUART transmitter is idle.

enumerator `kStatus_LPUART_RxIdle`

LPUART receiver is idle.

enumerator `kStatus_LPUART_TxWatermarkTooLarge`

TX FIFO watermark too large

enumerator `kStatus_LPUART_RxWatermarkTooLarge`

RX FIFO watermark too large

enumerator `kStatus_LPUART_FlagCannotClearManually`

Some flag can't manually clear

enumerator `kStatus_LPUART_Error`

Error happens on LPUART.

enumerator `kStatus_LPUART_RxRingBufferOverrun`

LPUART RX software ring buffer overrun.

enumerator `kStatus_LPUART_RxHardwareOverrun`

LPUART RX receiver overrun.

enumerator `kStatus_LPUART_NoiseError`

LPUART noise error.

enumerator `kStatus_LPUART_FramingError`

LPUART framing error.

enumerator `kStatus_LPUART_ParityError`

LPUART parity error.

enumerator `kStatus_LPUART_BaudrateNotSupport`

Baudrate is not support in current clock source

enumerator `kStatus_LPUART_IdleLineDetected`

IDLE flag.

enumerator `kStatus_LPUART_Timeout`

LPUART times out.

enum `_lpuart_parity_mode`

LPUART parity mode.

Values:

enumerator `kLPUART_ParityDisabled`

Parity disabled

enumerator `kLPUART_ParityEven`

Parity enabled, type even, bit setting: PE|PT = 10

enumerator `kLPUART_ParityOdd`

Parity enabled, type odd, bit setting: PE|PT = 11

enum `_lpuart_data_bits`

LPUART data bits count.

Values:

enumerator `kLPUART_EightDataBits`

Eight data bit

enumerator `kLPUART_SevenDataBits`

Seven data bit

enum `_lpuart_stop_bit_count`

LPUART stop bit count.

Values:

enumerator `kLPUART_OneStopBit`

One stop bit

enumerator `kLPUART_TwoStopBit`

Two stop bits

enum `_lpuart_transmit_cts_source`

LPUART transmit CTS source.

Values:

enumerator `kLPUART_CtsSourcePin`

CTS resource is the LPUART_CTS pin.

enumerator `kLPUART_CtsSourceMatchResult`

CTS resource is the match result.

enum `_lpuart_transmit_cts_config`

LPUART transmit CTS configure.

Values:

enumerator `kLPUART_CtsSampleAtStart`

CTS input is sampled at the start of each character.

enumerator `kLPUART_CtsSampleAtIdle`

CTS input is sampled when the transmitter is idle

enum `_lpuart_idle_type_select`

LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator `kLPUART_IdleTypeStartBit`

Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit

Start counting after a stop bit.

enum __lpuart_idle_config

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1

the number of idle characters.

enumerator kLPUART_IdleCharacter2

the number of idle characters.

enumerator kLPUART_IdleCharacter4

the number of idle characters.

enumerator kLPUART_IdleCharacter8

the number of idle characters.

enumerator kLPUART_IdleCharacter16

the number of idle characters.

enumerator kLPUART_IdleCharacter32

the number of idle characters.

enumerator kLPUART_IdleCharacter64

the number of idle characters.

enumerator kLPUART_IdleCharacter128

the number of idle characters.

enum __lpuart_interrupt_enable

LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable

LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable

Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable

Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable

Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable

Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable

Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable

Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable

Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable

Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable

Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable

Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable

Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable

Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable

Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags

LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag

Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag

The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag

The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag

RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t

LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t

LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t

LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t

LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t

LPUART transmit CTS configure.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t

LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *_lpuart_config* lpuart_config_t

LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t

LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES

Retry times for waiting flag.

struct `_lpuart_config`

#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps

LPUART baud rate

lpuart_parity_mode_t parityMode

Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount

Data bits count, eight (default), seven

bool isMsb

Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount

Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark

TX FIFO watermark

uint8_t rxFifoWatermark

RX FIFO watermark

bool enableRxRTS

RX RTS enable

bool enableTxCTS

TX CTS enable

lpuart_transmit_cts_source_t txCtsSource

TX CTS source

lpuart_transmit_cts_config_t txCtsConfig

TX CTS configure

lpuart_idle_type_select_t rxIdleType

RX IDLE type.

lpuart_idle_config_t rxIdleConfig

RX IDLE configuration.

bool enableTx

Enable TX

bool enableRx

Enable RX

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 __unnamed22__`

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 __unnamed24__
```

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 __unnamed26__
```

Public Members

```
uint8_t *volatile rxData
    Address of remaining data to receive.
uint16_t *volatile rxData16
    Address of remaining data to receive.
union __unnamed28__
```

Public Members

```
uint8_t *rxRingBuffer
    Start address of the receiver ring buffer.
uint16_t *rxRingBuffer16
    Start address of the receiver ring buffer.
```

2.27 LPUART eDMA Driver

```
void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,
    lpuart_edma_transfer_callback_t callback, void
    *userData, edma_handle_t *txEdmaHandle,
    edma_handle_t *rxEdmaHandle)
```

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

```
status_t LPUART_SendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                        lpuart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

```
status_t LPUART_ReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                           lpuart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

```
void LPUART_TransferAbortSendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)  
Aborts the sent data using eDMA.
```

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

```
void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)  
Aborts the received data using eDMA.
```

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

```
status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, lpuart_edma_handle_t
                                         *handle, uint32_t *count)
```

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

```
status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, lpuart_edma_handle_t
                                             *handle, uint32_t *count)
```

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

```
void LPUART_TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)
```

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- base – LPUART peripheral base address.
- lpuartEdmaHandle – LPUART handle pointer.

```
FSL_LPUART_EDMA_DRIVER_VERSION
```

LPUART EDMA driver version.

```
typedef struct _lpuart_edma_handle lpuart_edma_handle_t
```

```
typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t
*handle, status_t status, void *userData)
```

LPUART transfer callback function.

```
struct _lpuart_edma_handle
    #include <fsl_lpuart_edma.h> LPUART eDMA handle.
```

Public Members

lpuart_edma_transfer_callback_t callback
Callback function.

void *userData
LPUART callback function parameter.

size_t rxDataSizeAll
Size of the data to receive.

size_t txDataSizeAll
Size of the data to send out.

edma_handle_t *txEdmaHandle
The eDMA TX channel used.

edma_handle_t *rxEdmaHandle
The eDMA RX channel used.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.28 MU: Messaging Unit

void MU_Init(MU_Type *base)
Initializes the MU module.
This function enables the MU clock only.

Parameters

- base – MU peripheral base address.

void MU_Deinit(MU_Type *base)
De-initializes the MU module.
This function disables the MU clock only.

Parameters

- base – MU peripheral base address.

static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)
Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } Wait for TX0 register empty.
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL); Write message to the TX0 register.
```


Parameters

- base – MU peripheral base address.
- regIndex – TX register index, see `mu_msg_reg_index_t`.
- msg – Message to send.

```
void MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)
```

Blocks to send a message.

This function waits until the TX register is empty and sends the message.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see `mu_msg_reg_index_t`.
- msg – Message to send.

```
static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)
```

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
    } Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0); Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see `mu_msg_reg_index_t`.

Returns

The received message.

```
uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)
```

Blocks to receive a message.

This function waits until the RX register is full and receives the message.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see `mu_msg_reg_index_t`

Returns

The received message.

```
static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)
```

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag `kMU_FlagsUpdatingFlag` asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag `kMU_FlagsUpdatingFlag` is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag `kMU_FlagsUpdatingFlag` is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
} Wait for previous MU flags updating.

MU_SetFlagsNonBlocking(base, 0U); Set the mU flags.
```

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

void MU_SetFlags(MU_Type *base, uint32_t flags)

Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

static inline uint32_t MU_GetFlags(MU_Type *base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

static inline uint32_t MU_GetStatusFlags(MU_Type *base)

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See `_mu_status_flags`.

```
uint32_t flags;
flags = MU_GetStatusFlags(base); Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU status flags, see `_mu_status_flags`.

```
static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)
```

Return the RX status flags.

This function return the RX status flags. Note: RFn bits of SR[27-24](mu status register) are mapped in reverse numerical order: RF0 -> SR[27] RF1 -> SR[26] RF2 -> SR[25] RF3 -> SR[24]

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns

MU RX status

```
static inline uint32_t MU_GetInterruptsPending(MU_Type *base)
```

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. kMU_Tx0EmptyFlag kMU_Tx1EmptyFlag kMU_Tx2EmptyFlag kMU_Tx3EmptyFlag kMU_Rx0FullFlag kMU_Rx1FullFlag kMU_Rx2FullFlag kMU_Rx3FullFlag kMU_GenInt0Flag kMU_GenInt1Flag kMU_GenInt2Flag kMU_GenInt3Flag

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

```
static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t mask)
```

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See `_mu_status_flags`.

```
Clear general interrupt 0 and general interrupt 1 pending flags.  
MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU status flags. See `_mu_status_flags`. The following flags are cleared by hardware, this function could not clear them.
 - kMU_Tx0EmptyFlag
 - kMU_Tx1EmptyFlag
 - kMU_Tx2EmptyFlag
 - kMU_Tx3EmptyFlag
 - kMU_Rx0FullFlag
 - kMU_Rx1FullFlag
 - kMU_Rx2FullFlag
 - kMU_Rx3FullFlag
 - kMU_EventPendingFlag
 - kMU_FlagsUpdatingFlag
 - kMU_OtherSideInResetFlag

```
static inline void MU_EnableInterrupts(MU_Type *base, uint32_t mask)
```

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See `_mu_interrupt_enable`.

```
Enable general interrupt 0 and TX0 empty interrupt.
MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

```
static inline void MU_DisableInterrupts(MU_Type *base, uint32_t mask)
```

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See `_mu_interrupt_enable`.

```
Disable general interrupt 0 and TX0 empty interrupt.
MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

```
status_t MU_TriggerInterrupts(MU_Type *base, uint32_t mask)
```

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `_mu_interrupt_trigger`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the interrupts to trigger. See `_mu_interrupt_trigger`.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

```
void MU_BootCoreB(MU_Type *base, mu_core_boot_mode_t mode)
```

Boots the core at B side.

This function sets the B side core's boot configuration and releases the core from reset.

Note: Only MU side A can use this function.

Parameters

- base – MU peripheral base address.
- mode – Core B boot mode.

```
static inline void MU__HoldCoreBReset(MU_Type *base)
```

Holds the core reset of B side.

This function causes the core of B side to be held in reset following any reset event.

Note: Only A side could call this function.

Parameters

- base – MU peripheral base address.

```
void MU__BootOtherCore(MU_Type *base, mu_core_boot_mode_t mode)
```

Boots the other core.

This function boots the other core with a boot configuration.

Parameters

- base – MU peripheral base address.
- mode – The other core boot mode.

```
static inline void MU__HoldOtherCoreReset(MU_Type *base)
```

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

Parameters

- base – MU peripheral base address.

```
static inline void MU__ResetBothSides(MU_Type *base)
```

Resets the MU for both A side and B side.

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

Note: For some platforms, only MU side A could use this function, check reference manual for details.

Parameters

- base – MU peripheral base address.

```
void MU__HardwareResetOtherCore(MU_Type *base, bool waitReset, bool holdReset,  
                                mu_core_boot_mode_t bootMode)
```

Hardware reset the other core.

This function resets the other core, the other core could mask the hardware reset by calling MU_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU_BootOtherCore to control the other core reset workflow.

Example 1: Reset the other core, and no hold reset

```
MU__HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later.

```
Here the other core enters reset, and the reset is hold
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
Current core boot the other core when necessary.
MU_BootOtherCore(MU_A, bootMode);
```

Parameters

- base – MU peripheral base address.
- waitReset – Wait the other core enters reset.
 - true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
 - false: Don't wait the reset.
- holdReset – Hold the other core reset or not.
 - true: Hold the other core in reset, this function returns directly when the other core enters reset.
 - false: Don't hold the other core in reset, this function waits until the other core out of reset.
- bootMode – Boot mode of the other core, if holdReset is true, this parameter is useless.

```
static inline mu_power_mode_t MU_GetOtherCorePowerMode(MU_Type *base)
```

Gets the power mode of the other core.

This function gets the power mode of the other core.

Parameters

- base – MU peripheral base address.

Returns

Power mode of the other core.

```
FSL_MU_DRIVER_VERSION
```

MU driver version.

```
enum __mu_status_flags
```

MU status flags.

Values:

```
enumerator kMU_Tx0EmptyFlag
```

TX0 empty.

```
enumerator kMU_Tx1EmptyFlag
```

TX1 empty.

```
enumerator kMU_Tx2EmptyFlag
```

TX2 empty.

```
enumerator kMU_Tx3EmptyFlag
```

TX3 empty.

```
enumerator kMU_Rx0FullFlag
```

RX0 full.

enumerator kMU_Rx1FullFlag
RX1 full.

enumerator kMU_Rx2FullFlag
RX2 full.

enumerator kMU_Rx3FullFlag
RX3 full.

enumerator kMU_GenInt0Flag
General purpose interrupt 0 pending.

enumerator kMU_GenInt1Flag
General purpose interrupt 1 pending.

enumerator kMU_GenInt2Flag
General purpose interrupt 2 pending.

enumerator kMU_GenInt3Flag
General purpose interrupt 3 pending.

enumerator kMU_EventPendingFlag
MU event pending.

enumerator kMU_FlagsUpdatingFlag
MU flags update is on-going.

enumerator kMU_ResetAssertInterruptFlag
The other core reset assert interrupt pending.

enumerator kMU_ResetDeassertInterruptFlag
The other core reset de-assert interrupt pending.

enumerator kMU_MuResetInterruptFlag
The other side initializes MU reset.

enumerator kMU_HardwareResetInterruptFlag
Current side has been hardware reset by the other side.

enum _mu_interrupt_enable
MU interrupt source to enable.

Values:

enumerator kMU_Tx0EmptyInterruptEnable
TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable
TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable
TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable
TX3 empty.

enumerator kMU_Rx0FullInterruptEnable
RX0 full.

enumerator kMU_Rx1FullInterruptEnable
RX1 full.

enumerator kMU_Rx2FullInterruptEnable
RX2 full.

enumerator kMU_Rx3FullInterruptEnable
RX3 full.

enumerator kMU_GenInt0InterruptEnable
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable
General purpose interrupt 3.

enumerator kMU_ResetAssertInterruptEnable
The other core reset assert interrupt.

enumerator kMU_ResetDeassertInterruptEnable
The other core reset de-assert interrupt.

enumerator kMU_MuResetInterruptEnable
The other side initializes MU reset. The interrupt is ORed with the general purpose interrupt 3. The general purpose interrupt 3 is issued when the other side set the MU reset and this interrupt is enabled.

enumerator kMU_HardwareResetInterruptEnable
Current side has been hardware reset by the other side.

enum __mu_interrupt_trigger
MU interrupt that could be triggered to the other core.

Values:

enumerator kMU_GenInt0InterruptTrigger
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptTrigger
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptTrigger
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptTrigger
General purpose interrupt 3.

enum __mu_msg_reg_index
MU message register.

Values:

enumerator kMU_MsgReg0

enumerator kMU_MsgReg1

enumerator kMU_MsgReg2

enumerator kMU_MsgReg3

typedef enum __mu_msg_reg_index mu_msg_reg_index_t
MU message register.

MU_CR_NMI_MASK

MU_GET_CORE_FLAG(flags)

MU_GET_STAT_FLAG(flags)

MU_GET_TX_FLAG(flags)

MU_GET_RX_FLAG(flags)

MU_GET_GI_FLAG(flags)

2.29 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 is gray-code, if decimal data was desired, please using OSTIMER_SetMatchValue().

Parameters

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value is 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.
- kStatus_Fail -- Set match raw value fail.

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 internally.)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

- kStatus_Success -- Set match value and enable interrupt Successfully.
- kStatus_Fail -- Set match value fail.

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 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 a gray code type timer count value from OS timer register. The raw value of timer count is gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, 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. The RAW value of timer count is gray code format, will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formatted to decimal value.

static inline uint64_t OSTIMER_GetCaptureRawValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a captured gray-code value from OSTIMER. The Raw value of timer capture is gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of capture register, data format is 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. The RAW value of timer capture is gray code format, 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.

2.30 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.

2.31 POWER Driver

Values:

enumerator kStatus_POWER_MuTransferError
Fail due to Mu transfer error.

enumerator kStatus_POWER_RequestNotAllowed
Request not allowed by another core.

enumerator kStatus_Power_HandleDuplicated
Handle already be created.

enumerator kStatus_Power_NotInTargetMode

Not in target low power mode.

enum _power_low_power_mode

The enumeration of low power modes.

Values:

enumerator kPower_Sleep

Sleep Mode.

enumerator kPower_DeepSleep

Deep Sleep Mode.

enumerator kPower_PowerDown1

Power Down Mode 1.

enumerator kPower_PowerDown2

Power Down Mode 2.

enumerator kPower_DeepPowerDown1

Deep Power Down Mode 1.

enumerator kPower_DeepPowerDown2

Deep Power Down Mode 2.

enumerator kPower_DeepPowerDown3

Deep Power Down Mode 3.

enumerator kPower_ShutDown

ShutDown Mode

enumerator kPower_Active

Active Mode.

enum _power_wakeup_source

The enumeration of wakeup sources for different low power modes.

Values:

enumerator kPower_WS_Main_RtcAlarm0

RTC Alarm0 as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_RtcAlarm0

RTC Alarm0 as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_RtcAlarm0

RTC Alarm0 as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_RtcAlarm1

RTC Alarm1 as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_RtcAlarm1

RTC Alarm1 as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_RtcAlarm1

RTC Alarm1 as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_QTimerIrq

QTimer IRQ as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_QTimerIrq

QTimer IRQ as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_QTimerIrq
QTimer IRQ as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_RtcWDT
RTC watch dog as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_RtcWDT
RTC watch dog as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_RtcWDT
RTC watch dog as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_RtcXtalFail
RTC XTAL fail as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_RtcXtalFail
RTC XTAL fail as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_RtcXtalFail
RTC XTAL fail as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_ExternalINTFallEdge
External INT falling edge as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_ExternalINTFallEdge
External INT falling edge as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_ExternalINTFallEdge
External INT falling edge as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_ExternalINTRiseEdge
External INT rising edge as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_ExternalINTRiseEdge
External INT rising edge as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_ExternalINTRiseEdge
External INT rising edge as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_SMMTimer
Deep sleep counter as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_SMMTimer
Deep sleep counter as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_SMMTimer
Deep sleep counter as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_Comparator
Voltage comparator as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_Comparator
Voltage comparator as wakeup source, only wakeup AON Domain

enumerator kPower_WS_Both_Comparator
Voltage comparator as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_AonHsGpioWakeup0
HS_GPIO wakeup0 as wakeup source, only wakeup Main Domain

enumerator kPower_WS_Aon_AonHsGpioWakeup0
HS_GPIO wakeup0 as wakeup source, only wakeup AON Domain

```

enumerator kPower_WS_Both_AonHsGpioWakeup0
    HS_GPIO wakeup0 as wakeup source, wakeup both Main and AON Domains
enumerator kPower_WS_Main_Lpuart0
    LPUART0 Interrupt as wakeup source, only wakeup Main Domain
enumerator kPower_WS_Aon_Lpuart0
    LPUART0 Interrupt as wakeup source, only wakeup AON Domain
enumerator kPower_WS_Both_Lpuart0
    LPUART0 Interrupt as wakeup source, wakeup both Main and AON Domains
enumerator kPower_WS_Main_AdvcOrACMP
    ADVC or ACMP as wakeup source, only wakeup Main Domain
enumerator kPower_WS_Aon_AdvcOrACMP
    ADVC or ACMP as wakeup source, only wakeup AON Domain
enumerator kPower_WS_Both_AdvcOrACMP
    ADVC or ACMP as wakeup source, wakeup both Main and AON Domains
enumerator kPower_WS_Main_Lpi2cInt
    AON Lpi2c interrupt as wakeup source, only wakeup Main Domain
enumerator kPower_WS_Aon_Lpi2cInt
    AON Lpi2c interrupt as wakeup source, only wakeup AON Domain
enumerator kPower_WS_Both_Lpi2cInt
    AON Lpi2c interrupt as wakeup source, wakeup both Main and AON Domains
enumerator kPower_WS_Main_AdcInt
    ADC interrupt as wakeup source, only wakeup Main Domain
enumerator kPower_WS_Aon_AdcInt
    ADC interrupt as wakeup source, only wakeup AON Domain
enumerator kPower_WS_Both_AdcInt
    ADC interrupt as wakeup source, wakeup both Main and AON Domains
enumerator kPower_WS_Main_LptmrInt
    LPTMR interrupt as wakeup source, only wakeup Main Domain
enumerator kPower_WS_Aon_LptmrInt
    LPTMR interrupt as wakeup source, only wakeup AON Domain
enumerator kPower_WS_Both_LptmrInt
    LPTMR interrupt as wakeup source, wakeup both Main and AON Domains
enumerator kPower_WS_Main_TamperDetect
    Tamper detect as wakeup source, only wakeup Main Domain
enumerator kPower_WS_Aon_TamperDetect
    Tamper detect as wakeup source, only wakeup AON Domain
enumerator kPower_WS_Both_TamperDetect
    Tamper detect as wakeup source, wakeup both Main and AON Domains
enumerator kPower_WS_Main_LcdInt
    LCD interrupt as wakeup source, only wakeup Main Domain
enumerator kPower_WS_Aon_LcdInt
    LCD interrupt as wakeup source, only wakeup AON Domain

```

enumerator kPower_WS_Both_LcdInt

LCD interrupt as wakeup source, wakeup both Main and AON Domains

enumerator kPower_WS_Main_P0_0RiseEdgeInt

Pin P0_0 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0RiseEdgeDma

Pin P0_0 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0RiseEdgeTrig

Pin P0_0 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0FallEdgeInt

Pin P0_0 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0FallEdgeDma

Pin P0_0 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0FallEdgeTrig

Pin P0_0 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0AnyEdgeInt

Pin P0_0 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0AnyEdgeDma

Pin P0_0 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_0AnyEdgeTrig

Pin P0_0 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15RiseEdgeInt

Pin P0_15 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15RiseEdgeDma

Pin P0_15 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15RiseEdgeTrig

Pin P0_15 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15FallEdgeInt

Pin P0_15 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15FallEdgeDma

Pin P0_15 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15FallEdgeTrig

Pin P0_15 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15AnyEdgeInt

Pin P0_15 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15AnyEdgeDma

Pin P0_15 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_15AnyEdgeTrig

Pin P0_15 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16RiseEdgeInt

Pin P1_16 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16RiseEdgeDma

Pin P1_16 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16RiseEdgeTrig

Pin P1_16 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16FallEdgeInt

Pin P1_16 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16FallEdgeDma

Pin P1_16 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16FallEdgeTrig

Pin P1_16 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16AnyEdgeInt

Pin P1_16 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16AnyEdgeDma

Pin P1_16 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_16AnyEdgeTrig

Pin P1_16 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17RiseEdgeInt

Pin P1_17 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17RiseEdgeDma

Pin P1_17 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17RiseEdgeTrig

Pin P1_17 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17FallEdgeInt

Pin P1_17 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17FallEdgeDma

Pin P1_17 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17FallEdgeTrig

Pin P1_17 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17AnyEdgeInt

Pin P1_17 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17AnyEdgeDma

Pin P1_17 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_17AnyEdgeTrig

Pin P1_17 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20RiseEdgeInt

Pin P0_20 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20RiseEdgeDma

Pin P0_20 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20RiseEdgeTrig

Pin P0_20 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20FallEdgeInt

Pin P0_20 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20FallEdgeDma

Pin P0_20 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20FallEdgeTrig

Pin P0_20 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20AnyEdgeInt

Pin P0_20 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20AnyEdgeDma

Pin P0_20 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_20AnyEdgeTrig

Pin P0_20 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23RiseEdgeInt

Pin P0_23 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23RiseEdgeDma

Pin P0_23 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23RiseEdgeTrig

Pin P0_23 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23FallEdgeInt

Pin P0_23 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23FallEdgeDma

Pin P0_23 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23FallEdgeTrig

Pin P0_23 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23AnyEdgeInt

Pin P0_23 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23AnyEdgeDma

Pin P0_23 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_23AnyEdgeTrig

Pin P0_23 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3RiseEdgeInt

Pin P0_3 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3RiseEdgeDma

Pin P0_3 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3RiseEdgeTrig

Pin P0_3 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3FallEdgeInt

Pin P0_3 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3FallEdgeDma

Pin P0_3 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3FallEdgeTrig

Pin P0_3 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3AnyEdgeInt

Pin P0_3 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3AnyEdgeDma

Pin P0_3 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_3AnyEdgeTrig

Pin P0_3 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4RiseEdgeInt

Pin P0_4 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4RiseEdgeDma

Pin P0_4 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4RiseEdgeTrig

Pin P0_4 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4FallEdgeInt

Pin P0_4 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4FallEdgeDma

Pin P0_4 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4FallEdgeTrig

Pin P0_4 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4AnyEdgeInt

Pin P0_4 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4AnyEdgeDma

Pin P0_4 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_4AnyEdgeTrig

Pin P0_4 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13RiseEdgeInt

Pin P0_13 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13RiseEdgeDma

Pin P0_13 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13RiseEdgeTrig

Pin P0_13 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13FallEdgeInt

Pin P0_13 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13FallEdgeDma

Pin P0_13 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13FallEdgeTrig

Pin P0_13 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13AnyEdgeInt

Pin P0_13 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13AnyEdgeDma

Pin P0_13 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P0_13AnyEdgeTrig

Pin P0_13 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13RiseEdgeInt

Pin P1_13 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13RiseEdgeDma

Pin P1_13 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13RiseEdgeTrig

Pin P1_13 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13FallEdgeInt

Pin P1_13 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13FallEdgeDma

Pin P1_13 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13FallEdgeTrig

Pin P1_13 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13AnyEdgeInt

Pin P1_13 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13AnyEdgeDma

Pin P1_13 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_13AnyEdgeTrig

Pin P1_13 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1RiseEdgeInt

Pin P1_1 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1RiseEdgeDma

Pin P1_1 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1RiseEdgeTrig

Pin P1_1 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1FallEdgeInt

Pin P1_1 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1FallEdgeDma

Pin P1_1 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1FallEdgeTrig

Pin P1_1 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1AnyEdgeInt

Pin P1_1 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1AnyEdgeDma

Pin P1_1 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_1AnyEdgeTrig

Pin P1_1 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2RiseEdgeInt

Pin P1_2 rising edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2RiseEdgeDma

Pin P1_2 rising edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2RiseEdgeTrig

Pin P1_2 rising edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2FallEdgeInt

Pin P1_2 falling edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2FallEdgeDma

Pin P1_2 falling edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2FallEdgeTrig

Pin P1_2 falling edge trigger as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2AnyEdgeInt

Pin P1_2 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2AnyEdgeDma

Pin P1_2 any edge DMA request as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_2AnyEdgeTrig

Pin P1_2 any edge interrupt as wakeup source, can be used to wakeup from DS mode.

enumerator kPower_WS_Main_P1_5RiseEdgeInt

P1_5 pin rising edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5RiseEdgeDma

P1_5 pin rising edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5RiseEdgeTrig

P1_5 pin rising edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5FallEdgeInt

P1_5 pin falling edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5FallEdgeDma

P1_5 pin falling edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5FallEdgeTrig

P1_5 pin falling edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5AnyEdgeInt

P1_5 pin any edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5AnyEdgeDma

P1_5 pin any edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_5AnyEdgeTrig

P1_5 pin any edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14RiseEdgeInt

P1_14 pin rising edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14RiseEdgeDma

P1_14 pin rising edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14RiseEdgeTrig

P1_14 pin rising edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14FallEdgeInt

P1_14 pin falling edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14FallEdgeDma

P1_14 pin falling edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14FallEdgeTrig

P1_14 pin falling edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14AnyEdgeInt

P1_14 pin any edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14AnyEdgeDma

P1_14 pin any edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_14AnyEdgeTrig

P1_14 pin any edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19RiseEdgeInt

P1_19 pin rising edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19RiseEdgeDma
P1_19 pin rising edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19RiseEdgeTrig
P1_19 pin rising edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19FallEdgeInt
P1_19 pin falling edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19FallEdgeDma
P1_19 pin falling edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19FallEdgeTrig
P1_19 pin falling edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19AnyEdgeInt
P1_19 pin any edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19AnyEdgeDma
P1_19 pin any edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_19AnyEdgeTrig
P1_19 pin any edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23RiseEdgeInt
P1_23 pin rising edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23RiseEdgeDma
P1_23 pin rising edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23RiseEdgeTrig
P1_23 pin rising edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23FallEdgeInt
P1_23 pin falling edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23FallEdgeDma
P1_23 pin falling edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23FallEdgeTrig
P1_23 pin falling edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23AnyEdgeInt
P1_23 pin any edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23AnyEdgeDma
P1_23 pin any edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_23AnyEdgeTrig
P1_23 pin any edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22RiseEdgeInt
P1_22 pin rising edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22RiseEdgeDma
P1_22 pin rising edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22RiseEdgeTrig
P1_22 pin rising edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22FallEdgeInt
P1_22 pin falling edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22FallEdgeDma
P1_22 pin falling edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22FallEdgeTrig
P1_22 pin falling edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22AnyEdgeInt
P1_22 pin any edge interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22AnyEdgeDma
P1_22 pin any edge DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_P1_22AnyEdgeTrig
P1_22 pin any edge trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_Lptmr0Int
LPTMR0 interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_Cmp0Int
CMP0 interrupt as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_Lptmr0Dma
LPTMR0 DMA as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_Lptmr0Trig
LPTMR0 trigger as wakeup source, wakeup Main Domain

enumerator kPower_WS_Main_Cmp0DMA
CMP0 DMA as wakeup source, wakeup Main Domain

enum _power_main_domain_sram_array

Values:

enumerator kPower_MainDomainNoneRams

enumerator kPower_MainDomainRamX0
Main Domain RAM X0, bitmask representation for power control

enumerator kPower_MainDomainRamX1
Main Domain RAM X1, bitmask representation for power control

enumerator kPower_MainDomainRamA0
Main Domain RAM A0, bitmask representation for power control

enumerator kPower_MainDomainRamA1
Main Domain RAM A1, bitmask representation for power control

enumerator kPower_MainDomainRamA2
Main Domain RAM A2, bitmask representation for power control

enumerator kPower_MainDomainRamA3
Main Domain RAM A3, bitmask representation for power control

enumerator kPower_MainDomainRamB0
Main Domain RAM B0, bitmask representation for power control

enumerator kPower_MainDomainRamB1
Main Domain RAM B1, bitmask representation for power control

enumerator kPower_MainDomainRamB2ToB4
Main Domain RAM B2 to B4, bitmask representation for power control

enumerator kPower__MainDomainAllRams

Represents all RAMs in the Main Domain, bitmask for power control

enum __power__aon__domain__sram__array

Values:

enumerator kPower__AonDomainNoneRams

No AON Domain RAMs.

enumerator kPower__AonDomainRam1stHalf16kB

First half (16kB) of AON Domain RAM, bitmask for power control

enumerator kPower__AonDomainRam2nd8kB

Second 8kB of AON Domain RAM, bitmask for power control

enumerator kPower__AonDomainRamLower8kB

Lower 8kB of AON Domain RAM, bitmask for power control

enumerator kPower__AonDomainAllRams

Represents all RAMs in the AON Domain, bitmask for power control

enum __power__mu__message__direction

Enumeration of power MU message directions.

Values:

enumerator kPower__MsgDirMainToAon

Message direction from Main to AON domain

enumerator kPower__MsgDirAonToMain

Message direction from AON to Main domain

enum __power__mu__message__type

Enumeration of power MU message types.

Values:

enumerator kPower__MsgTypeRequest

Message type is a request

enumerator kPower__MsgTypeACK

Message type is an ACKnowledgment

enumerator kPower__MsgTypeNACK

Message type is a Negative ACKnowledgment

enum __power__vdd__core__output__voltage

Output voltage options for VDDCore in low power modes.

Values:

enumerator kPower__VddCoreOutput1P0V

enum __power__dpd1__transition

Enumeration of follow power transition.

Values:

enumerator kPower__Dpd1ToActive

Transition from DPD1 to Active mode

enumerator kPower__Dpd1ToDpd2WakeToDpd1

Transition from DPD1 to DPD2 wakeup to DPD1 mode

enumerator `kPower_Dpd1ToDpd2WakeToActive`

Transition from DPD1 to DPD2 wakeup to Active mode

typedef enum `_power_low_power_mode` `power_low_power_mode_t`

The enumeration of low power modes.

typedef bool (`*power_user_callback_t`)(`power_low_power_mode_t` targetPowerMode, void `*ptrPowerConfig`, void `*userData`)

typedef enum `_power_wakeup_source` `power_wakeup_source_t`

The enumeration of wakeup sources for different low power modes.

typedef struct `_power_wakeup_source_info` `power_wakeup_source_info_t`

The structure of dumped wakeup source information.

typedef enum `_power_mu_message_direction` `power_mu_message_direction_t`

Enumeration of power MU message directions.

typedef enum `_power_mu_message_type` `power_mu_message_type_t`

Enumeration of power MU message types.

typedef union `_power_mu_message` `power_mu_message_t`

Union representing a power MU message.

typedef enum `_power_vdd_core_output_voltage` `power_vdd_core_output_voltage_t`

Output voltage options for VDDCore in low power modes.

typedef struct `_power_ds_config` `power_ds_config_t`

typedef struct `_power_pd1_config` `power_pd1_config_t`

Configuration structure for power down mode.

typedef struct `_power_pd2_config` `power_pd2_config_t`

typedef enum `_power_dpd1_transition` `power_dpd1_transition_t`

Enumeration of follow power transition.

typedef struct `_power_dpd1_config` `power_dpd1_config_t`

Configuration structure for deep power down mode 1.

typedef struct `_power_dpd2_config` `power_dpd2_config_t`

Configuration structure for deep power down mode 2.

typedef struct `_power_dpd3_config` `power_dpd3_config_t`

Configuration structure for deep power down mode 3.

typedef struct `_power_sd_config` `power_sd_config_t`

Configuration structure for shut down.

typedef struct `_power_handle` `power_handle_t`

Structure to handle power management operations.

`status_t` `Power_CreateHandle(power_handle_t *handle, uint32_t muChannelId)`

Create shared power handle.

Parameters

- `handle` – **[in]** Pointer to a handle in type of `power_handle_t`, must be in shared RAM.
- `muChannelId` – **[in]** MU channel ID used by power driver.

Return values

- kStatus_POWER_MuTransferError – Something error occurs during MU transfer.
- kStatus_Power_HandleDuplicated – Shared power handle already be created.
- kStatus_Success – Created handle successfully.

void Power__DumpHandleValue(*power_handle_t* *ptrDumpBuffer)

uint32_t Power__GetHandleOffset(void)

Get the offset of shared handle in shared RAM.

Returns

Offset of shared handle in shared RAM, in bytes.

void Power__RestoreHandleOffset(uint32_t offset)

Restore the shared handle from offset in shared RAM.

Parameters

- offset – **[in]** Offset of shared handle in shared RAM, in bytes.

FSL_POWER_DRIVER_VERSION

power driver version 2.0.0.

POWER_SHARED_RAM_BASE_ADDR

POWER_ENCODE_WS(cm33Ws, wuuExtPin, wakeupDomain, pinEdge, wuuEvent, wuuIndex, aonIndex)

Encode wakeup source code.

Note: bit 7 - 0: aon control bit index; bit 15 - 8: wuu control bit index; bit 19 - 16: wuu event(0: interrupt, 1: DMA request, 2: trigger event); bit 23 - 20: external pin edge(1: rising edge, 2: falling edge, 3: any edge); bit 27 - 24: Wakeup domain(0: only Main, 1: only Aon, 2: Both Main and Aon), in some low power mode(such as PD2) it is possible only wakeup Aon Domain. bit 30: external pin or internal module(0: internal module, 1: external pin); bit 31: wakeup source in aon domain or cm33 domain(0: In AON domain, 1: In CM33 domain).

POWER_DECODE_WS(wsCode)

POWER_DPD2_WS_BIT_MASK

POWER_DPD3_WS_BIT_MASK

POWER_SHUTDOWN_WS_BIT_MASK

void Power__EnableWakeupSource(*power_wakeup_source_t* ws)

Enable input wakeup source, once enabled it will be effective until disabled.

Parameters

- ws – **[in]** Specify the coded wakeup source, please refer to *power_wakeup_source_t* for details.

void Power__DisableWakeupSource(*power_wakeup_source_t* ws)

Disable input wakeup source.

Parameters

- ws – **[in]** Specify the coded wakeup source, please refer to *power_wakeup_source_t* for details.

void Power__DumpEnabledWakeSource(*power_wakeup_source_info_t* *ptrWsInfo)

Dump information of all configured wakeup sources, in type of *power_wakeup_source_info_t*.

Parameters

- ptrWsInfo – **[out]** Pointer to the variable to store dumped wakeup source information.

void Power__GetWakeupSource(uint32_t *ptrWakeupSourceMask)

Get latest mask of wakeup sources which cause the wake-up to main CPU.

Parameters

- ptrWakeupSourceMask – **[out]** Pointer to the variable to store mask of wakeup sources.

void Power__RegisterUserCallback(*power_user_callback_t* callback, void *userData)

Register user callback.

Parameters

- callback – **[in]** Pointer to callback in type of *power_user_callback_t*.
- userData – **[in]** Pointer to user data.

void Power__UnRegisterUserCallback(void)

Unregister user callback.

status_t Power__GetCurrentPowerMode(*power_low_power_mode_t* *ptrCurLpMode)

Parameters

- ptrCurLpMode – **[out]** Pointer to store current low power mode

Return values

kStatus__Success – Successfully retrieved current low power mode.

power_low_power_mode_t Power__GetTargetPowerMode(void)

Get the target low power mode.

Returns

Requested low power mode, in type of *power_low_power_mode_t*.

void Power__ClearTargatePowerMode(void)

Clear the target low power mode.

void Power__ClearLpPowerSettings(void)

Clear all low power settings.

status_t Power__EnterLowPowerMode(*power_low_power_mode_t* lowpowerMode, void *config)

Enter selected low power mode.

Parameters

- lowpowerMode – **[in]** Indicate specific low power mode.
- config – Point to low power configurations.

Return values

- kStatus__POWER__MuTransferError – Something error occurs during MU transfer.
- kStatus__POWER__RequestNotAllowed – Request not allowed by another core.

status_t Power_EnterSleep(void)

Enter the sleep mode.

This function is used to put the system into sleep mode.

Return values

- kStatus_Success – Successfully entered sleep mode.
- kStatus_POWER_MuTransferError – Something error occurs during MU transfer.
- kStatus_POWER_RequestNotAllowed – Request not allowed by another core.

status_t Power_EnterDeepSleep(*power_ds_config_t* *config)

Enter Deep Sleep mode.

This function attempts to put the system into Deep Sleep mode with the provided configuration.

Parameters

- config – **[in]** Pointer to the Deep Sleep mode configuration.

Return values

- kStatus_Success – Successfully entered Deep Sleep mode.
- kStatus_POWER_MuTransferError – Something error occurs during MU transfer.
- kStatus_POWER_RequestNotAllowed – Request not allowed by another core.

status_t Power_EnterPowerDown1(*power_pd1_config_t* *config)

Enter Power Down 1 mode.

This function attempts to put the system into Power Down 1 mode with the provided configuration.

Parameters

- config – **[in]** Pointer to the Power Down 1 mode configuration.

Return values

- kStatus_Success – Successfully entered Power Down 1 mode.
- kStatus_POWER_MuTransferError – Something error occurs during MU transfer.
- kStatus_POWER_RequestNotAllowed – Request not allowed by another core.

status_t Power_EnterPowerDown2(*power_pd2_config_t* *config)

Enter Power Down 2 mode.

This function attempts to put the system into Power Down 2 mode with the provided configuration.

Parameters

- config – **[in]** Pointer to the Power Down 2 mode configuration.

Return values

- kStatus_Success – Successfully entered Power Down 2 mode.
- kStatus_POWER_MuTransferError – Something error occurs during MU transfer.

- `kStatus_POWER_RequestNotAllowed` – Request not allowed by another core.

status_t Power_EnterDeepPowerDown1(*power_dpd1_config_t* *config)

Enter Deep Power Down 1 mode.

This function attempts to put the system into Deep Power Down 1 mode with the provided configuration.

Parameters

- config – **[in]** Pointer to the Deep Power Down 1 mode configuration.

Return values

- `kStatus_Success` – Successfully entered Deep Power Down 1 mode.
- `kStatus_POWER_MuTransferError` – Something error occurs during MU transfer.
- `kStatus_POWER_RequestNotAllowed` – Request not allowed by another core.

power_dpd1_transition_t Power_GetDeepPowerDown1NextTransition(void)

Get the next transition after Deep Power Down 1 mode.

Returns

Next transition after Deep Power Down 1 mode, in type of *power_dpd1_transition_t*.

status_t Power_EnterDeepPowerDown2(*power_dpd2_config_t* *config)

Enter Deep Power Down 2 mode.

This function attempts to put the system into Deep Power Down 2 mode with the provided configuration.

Parameters

- config – **[in]** Pointer to the Deep Power Down 2 mode configuration.

Return values

- `kStatus_Success` – Successfully entered Deep Power Down 2 mode.
- `kStatus_POWER_MuTransferError` – Something error occurs during MU transfer.
- `kStatus_POWER_RequestNotAllowed` – Request not allowed by another core.

status_t Power_EnterDeepPowerDown3(*power_dpd3_config_t* *config)

Enter Deep Power Down 3 mode.

This function attempts to put the system into Deep Power Down 3 mode with the provided configuration.

Parameters

- config – **[in]** Pointer to the Deep Power Down 3 mode configuration.

Return values

- `kStatus_Success` – Successfully entered Deep Power Down 3 mode.
- `kStatus_POWER_MuTransferError` – Something error occurs during MU transfer.
- `kStatus_POWER_RequestNotAllowed` – Request not allowed by another core.

status_t Power_EnterShutDown(*power_sd_config_t* *config)

Enter Shutdown mode.

This function attempts to put the system into Shutdown mode with the provided configuration.

Parameters

- config – **[in]** Pointer to the Shutdown mode configuration.

Return values

- kStatus_Success – Successfully entered Shutdown mode.
- kStatus_POWER_MuTransferError – Something error occurs during MU transfer.
- kStatus_POWER_RequestNotAllowed – Request not allowed by another core.

void Power_MuMessageCallback(*uint32_t* message, *uint32_t* channelId)

Callback function for handling power MU messages.

This function is called when a power MU message is received. It processes the message based on the given message content and the channel ID.

Parameters

- message – **[in]** The received power MU message.
- channelId – **[in]** The ID of the channel on which the message was received.

Return values

None – This function does not return a value.

struct __power_wakeup_source_info

#include <fsl_power.h> The structure of dumped wakeup source information.

Public Members

uint32_t aonWakeupSourceMask

The mask of wakeup sources in AON domain.

uint32_t wuuPinIntEnable[2]

Information of enabled Pin interrupts.

uint32_t wuuModuleIntEnable

Information of enabled internal module interrupts.

uint32_t wuuModuleDmaTrigEnable

Information of enabled internal module DMA request/trigger.

uint32_t wuuPinDmaTrigConfig[2]

Information of enabled pin DMA request/trigger.

union __power_mu_message

#include <fsl_power.h> Union representing a power MU message.

Public Members

struct __power_mu_message strcutFormat

uint32_t wordFormat

Message in word format

```
struct __power_ds_config
    #include <fsl_power.h>
struct __power_pd1_config
    #include <fsl_power.h> Configuration structure for power down mode.
```

Public Members

```
uint32_t mainRamArraysToRetain
    Bitmask representing the main domain RAM arrays to retain during power down
bool disableBandgap
    Flag to indicate whether to disable the bandgap during power down
bool enableIVSMode
    Enable/disable IVS mode for the Main domain SRAM retention.
struct __power_pd2_config
    #include <fsl_power.h>
```

Public Members

```
uint32_t mainRamArraysToRetain
    Bitmask representing the main domain RAM arrays to retain during DPD2 mode
uint32_t aonRamArraysToRetain
    Bitmask representing the AON domain RAM arrays to retain during DPD2 mode
bool enableIVSMode
    Enable/disable IVS mode for the Main domain SRAM retention.
bool disableBandgap
    Flag to indicate whether to disable the bandgap during DPD2 mode
bool switchToX32K
    Flag to indicate whether to switch to X32K clock source during DPD2 mode
bool disableFRO10M
    Flag to indicate whether to disable the FRO10M clock during DPD2 mode
struct __power_dpd1_config
    #include <fsl_power.h> Configuration structure for deep power down mode 1.
```

Public Members

```
uint32_t mainRamArraysToRetain
    Bitmask representing the main domain RAM arrays to retain during power down
bool disableBandgap
    Flag to indicate whether to disable the bandgap during power down
bool enableIVSMode
    Enable/disable IVS mode for the Main domain SRAM retention.
power_dpd1_transition_t nextTrans
    Next transition after DPD1 mode, refer to power_dpd1_transition_t
struct __power_dpd2_config
    #include <fsl_power.h> Configuration structure for deep power down mode 2.
```


Public Members

`uint32_t` `mainRamArraysToRetain`
Bitmask representing the main domain RAM arrays to retain during DPD2 mode

`uint32_t` `aonRamArraysToRetain`
Bitmask representing the AON domain RAM arrays to retain during DPD2 mode

`bool` `enableIVSMode`
Enable/disable IVS mode for the Main domain SRAM retention.

`bool` `disableBandgap`
Flag to indicate whether to disable the bandgap during DPD2 mode

`bool` `switchToX32K`
Flag to indicate whether to switch to X32K clock source during DPD2 mode

`bool` `disableFRO10M`
Flag to indicate whether to disable the FRO10M clock during DPD2 mode

`bool` `wakeToDpd1`
Flag to indicate whether to wake up to DPD1 mode after DPD2 mode

`struct __power_dpd3_config`
#include <fsl_power.h> Configuration structure for deep power down mode 3.

`struct __power_sd_config`
#include <fsl_power.h> Configuration structure for shut down.

`struct __power_handle`
#include <fsl_power.h> Structure to handle power management operations.

Public Members

`uint32_t` `muChannelId`
ID of the Message Unit (MU) channel used for power communication

`power_low_power_mode_t` `targetPowerMode`
Current low power mode of the system

`uint32_t` `lpConfig[2U]`
Array of two 32-bit values for low power configuration

`bool` `dualCoreSynced`
Flag indicating whether dual cores are synchronized

`bool` `requestCM33Start`
Flag indicating whether a request to start CM33 core is made

`power_user_callback_t` `cm33Callback`
Callback function for CM33 core operations, in type of `power_user_callback_t`

`void *``cm33UserData`
User data pointer for CM33 core operations

`power_user_callback_t` `cm0pCallback`
Callback function for CM0+ core operations, in type of `power_user_callback_t`

`void *``cm0pUserData`
User data pointer for CM0+ core operations

`struct` `strcutFormat`

Public Members

uint32_t syncCode

Synchronization code for the message

power_mu_message_type_t type

Type of the message, refer to power_mu_message_type_t

power_mu_message_direction_t direction

Direction of the message, refer to power_mu_message_direction_t

power_low_power_mode_t requestLowPowerMode

Requested low power mode

bool init

Initialization flag

uint32_t sharedHandleAddrOff

Offset from shared ram base address

2.32 QTMR: Quad Timer Driver

```
void QTMR_Init(TMR_Type *base, qtmr_channel_selection_t channel, const qtmr_config_t *config)
```

Ungates the Quad Timer clock and configures the peripheral for basic operation.

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

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- config – Pointer to user's Quad Timer config structure

```
void QTMR_Deinit(TMR_Type *base, qtmr_channel_selection_t channel)
```

Stops the counter and gates the Quad Timer clock.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

```
void QTMR_GetDefaultConfig(qtmr_config_t *config)
```

Fill in the Quad Timer config struct with the default settings.

The default values are:

```
config->debugMode = kQTMR_RunNormalInDebug;
config->enableExternalForce = false;
config->enableMasterMode = false;
config->faultFilterCount = 0;
config->faultFilterPeriod = 0;
config->primarySource = kQTMR_ClockDivide_2;
config->secondarySource = kQTMR_Counter0InputPin;
```

Parameters

- config – Pointer to user's Quad Timer config structure.

```
void QTMR_EnableInterrupts(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Enables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `qtmr_interrupt_enable_t`

```
void QTMR_DisableInterrupts(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Disables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `qtmr_interrupt_enable_t`

```
uint32_t QTMR_GetEnabledInterrupts(TMR_Type *base, qtmr_channel_selection_t channel)
```

Gets the enabled Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `qtmr_interrupt_enable_t`

```
uint32_t QTMR_GetStatus(TMR_Type *base, qtmr_channel_selection_t channel)
```

Gets the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

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

```
void QTMR_ClearStatusFlags(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Clears the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration `qtmr_status_flags_t`

`void QTMR_SetTimerPeriod(TMR_Type *base, qtmr_channel_selection_t channel, uint16_t ticks)`
Sets the timer period in ticks.

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note:

- a. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
 - b. User can call the utility macros provided in `fsl_common.h` to convert to ticks
 - c. This function supports cases, providing only primary source clock without secondary source clock.
-

Parameters

- `base` – Quad Timer peripheral base address
- `channel` – Quad Timer channel number
- `ticks` – Timer period in units of ticks

`void QTMR_SetCompareValue(TMR_Type *base, qtmr_channel_selection_t channel, uint16_t ticks)`

Set compare value.

This function sets the value used for comparison with the counter value.

Parameters

- `base` – Quad Timer peripheral base address
- `channel` – Quad Timer channel number
- `ticks` – Timer period in units of ticks.

`static inline void QTMR_SetLoadValue(TMR_Type *base, qtmr_channel_selection_t channel, uint16_t value)`

Set load value.

This function sets the value used to initialize the counter after a counter comparison.

Parameters

- `base` – Quad Timer peripheral base address
- `channel` – Quad Timer channel number
- `value` – Load register initialization value.

`static inline uint16_t QTMR_GetCurrentTimerCount(TMR_Type *base, qtmr_channel_selection_t channel)`

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec

Parameters

- `base` – Quad Timer peripheral base address

- channel – Quad Timer channel number

Returns

Current counter value in ticks

```
static inline void QTMR_StartTimer(TMR_Type *base, qtmr_channel_selection_t channel,  
                                  qtmr_counting_mode_t clockSource)
```

Starts the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- clockSource – Quad Timer clock source

```
static inline void QTMR_StopTimer(TMR_Type *base, qtmr_channel_selection_t channel)
```

Stops the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

```
void QTMR_EnableDma(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Enable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration *qtmr_dma_enable_t*

```
void QTMR_DisableDma(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Disable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration *qtmr_dma_enable_t*

```
void QTMR_SetPwmOutputToIdle(TMR_Type *base, qtmr_channel_selection_t channel, bool  
                             idleStatus)
```

Set PWM output in idle status (high or low).

Note: When the PWM is set again, the counting needs to be restarted.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- idleStatus – True: PWM output is high in idle status; false: PWM output is low in idle status.

```
static inline qtmr_pwm_out_state_t QTMR_GetPwmOutputStatus(TMR_Type *base,
                                                         qtmr_channel_selection_t
                                                         channel)
```

Get the channel output status.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel output status.

```
uint8_t QTMR_GetPwmChannelStatus(TMR_Type *base, qtmr_channel_selection_t channel)
```

Get the PWM channel duty cycle value.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel duty cycle value.

```
void QTMR_SetPwmClockMode(TMR_Type *base, qtmr_channel_selection_t channel,
                          qtmr_primary_count_source_t prescaler)
```

This function set the value of the prescaler on QTimer channels.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- prescaler – Set prescaler value

```
FSL_QTMR_DRIVER_VERSION
```

Version

```
enum __qtmr_primary_count_source
```

Quad Timer primary clock source selection.

Values:

```
enumerator kQTMR_ClockCounter0InputPin
```

Use counter 0 input pin

```
enumerator kQTMR_ClockCounter1InputPin
```

Use counter 1 input pin

```
enumerator kQTMR_ClockCounter2InputPin
```

Use counter 2 input pin

```
enumerator kQTMR_ClockCounter3InputPin
```

Use counter 3 input pin

```
enumerator kQTMR_ClockCounter0Output
```

Use counter 0 output

```
enumerator kQTMR_ClockCounter1Output
```

Use counter 1 output

```
enumerator kQTMR_ClockCounter2Output
```

Use counter 2 output

enumerator kQTMR_ClockCounter3Output

Use counter 3 output

enumerator kQTMR_ClockDivide_1

IP bus clock divide by 1 prescaler

enumerator kQTMR_ClockDivide_2

IP bus clock divide by 2 prescaler

enumerator kQTMR_ClockDivide_4

IP bus clock divide by 4 prescaler

enumerator kQTMR_ClockDivide_8

IP bus clock divide by 8 prescaler

enumerator kQTMR_ClockDivide_16

IP bus clock divide by 16 prescaler

enumerator kQTMR_ClockDivide_32

IP bus clock divide by 32 prescaler

enumerator kQTMR_ClockDivide_64

IP bus clock divide by 64 prescaler

enumerator kQTMR_ClockDivide_128

IP bus clock divide by 128 prescaler

enum _qtmr_input_source

Quad Timer input sources selection.

Values:

enumerator kQTMR_Counter0InputPin

Use counter 0 input pin

enumerator kQTMR_Counter1InputPin

Use counter 1 input pin

enumerator kQTMR_Counter2InputPin

Use counter 2 input pin

enumerator kQTMR_Counter3InputPin

Use counter 3 input pin

enum _qtmr_counting_mode

Quad Timer counting mode selection.

Values:

enumerator kQTMR_NoOperation

No operation

enumerator kQTMR_PriSrcRiseEdge

Count rising edges of primary source

enumerator kQTMR_PriSrcRiseAndFallEdge

Count rising and falling edges of primary source

enumerator kQTMR_PriSrcRiseEdgeSecInpHigh

Count rise edges of pri SRC while sec inp high active

enumerator kQTMR_QuadCountMode

Quadrature count mode, uses pri and sec sources

enumerator kQTMR_PriSrcRiseEdgeSecDir
Count rising edges of pri SRC; sec SRC specifies dir

enumerator kQTMR_SecSrcTrigPriCnt
Edge of sec SRC trigger primary count until compare

enumerator kQTMR_CascadeCount
Cascaded count mode (up/down)

enum __qtmr_pwm_out_state
Quad Timer PWM output state.
Values:

enumerator kQTMR_PwmLow
The output state of PWM channel is low

enumerator kQTMR_PwmHigh
The output state of PWM channel is low

enum __qtmr_output_mode
Quad Timer output mode selection.
Values:

enumerator kQTMR_AssertWhenCountActive
Assert OFLAG while counter is active

enumerator kQTMR_ClearOnCompare
Clear OFLAG on successful compare

enumerator kQTMR_SetOnCompare
Set OFLAG on successful compare

enumerator kQTMR_ToggleOnCompare
Toggle OFLAG on successful compare

enumerator kQTMR_ToggleOnAltCompareReg
Toggle OFLAG using alternating compare registers

enumerator kQTMR_SetOnCompareClearOnSecSrcInp
Set OFLAG on compare, clear on sec SRC input edge

enumerator kQTMR_SetOnCompareClearOnCountRoll
Set OFLAG on compare, clear on counter rollover

enumerator kQTMR_EnableGateClock
Enable gated clock output while count is active

enum __qtmr_input_capture_edge
Quad Timer input capture edge mode, rising edge, or falling edge.
Values:

enumerator kQTMR_NoCapture
Capture is disabled

enumerator kQTMR_RisingEdge
Capture on rising edge (IPS=0) or falling edge (IPS=1)

enumerator kQTMR_FallingEdge
Capture on falling edge (IPS=0) or rising edge (IPS=1)

enumerator kQTMR__RisingAndFallingEdge

Capture on both edges

enum __qtmr__preload__control

Quad Timer input capture edge mode, rising edge, or falling edge.

Values:

enumerator kQTMR__NoPreload

Never preload

enumerator kQTMR__LoadOnComp1

Load upon successful compare with value in COMP1

enumerator kQTMR__LoadOnComp2

Load upon successful compare with value in COMP2

enum __qtmr__debug__action

List of Quad Timer run options when in Debug mode.

Values:

enumerator kQTMR__RunNormalInDebug

Continue with normal operation

enumerator kQTMR__HaltCounter

Halt counter

enumerator kQTMR__ForceOutToZero

Force output to logic 0

enumerator kQTMR__HaltCountForceOutZero

Halt counter and force output to logic 0

enum __qtmr__interrupt__enable

List of Quad Timer interrupts.

Values:

enumerator kQTMR__CompareInterruptEnable

Compare interrupt.

enumerator kQTMR__Compare1InterruptEnable

Compare 1 interrupt.

enumerator kQTMR__Compare2InterruptEnable

Compare 2 interrupt.

enumerator kQTMR__OverflowInterruptEnable

Timer overflow interrupt.

enumerator kQTMR__EdgeInterruptEnable

Input edge interrupt.

enum __qtmr__status__flags

List of Quad Timer flags.

Values:

enumerator kQTMR__CompareFlag

Compare flag

enumerator kQTMR__Compare1Flag

Compare 1 flag

enumerator kQTMR_Compare2Flag
Compare 2 flag

enumerator kQTMR_OverflowFlag
Timer overflow flag

enumerator kQTMR_EdgeFlag
Input edge flag

enum __qtmr_channel_selection
List of channel selection.
Values:

enumerator kQTMR_Channel_0
TMR Channel 0

enumerator kQTMR_Channel_1
TMR Channel 1

enumerator kQTMR_Channel_2
TMR Channel 2

enumerator kQTMR_Channel_3
TMR Channel 3

enum __qtmr_dma_enable
List of Quad Timer DMA enable.
Values:

enumerator kQTMR_InputEdgeFlagDmaEnable
Input Edge Flag DMA Enable.

enumerator kQTMR_ComparatorPreload1DmaEnable
Comparator Preload Register 1 DMA Enable.

enumerator kQTMR_ComparatorPreload2DmaEnable
Comparator Preload Register 2 DMA Enable.

typedef enum __qtmr_primary_count_source qtmr_primary_count_source_t
Quad Timer primary clock source selection.

typedef enum __qtmr_input_source qtmr_input_source_t
Quad Timer input sources selection.

typedef enum __qtmr_counting_mode qtmr_counting_mode_t
Quad Timer counting mode selection.

typedef enum __qtmr_pwm_out_state qtmr_pwm_out_state_t
Quad Timer PWM output state.

typedef enum __qtmr_output_mode qtmr_output_mode_t
Quad Timer output mode selection.

typedef enum __qtmr_input_capture_edge qtmr_input_capture_edge_t
Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum __qtmr_preload_control qtmr_preload_control_t
Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum __qtmr_debug_action qtmr_debug_action_t
List of Quad Timer run options when in Debug mode.

```
typedef enum _qtmr_interrupt_enable qtmr_interrupt_enable_t
```

List of Quad Timer interrupts.

```
typedef enum _qtmr_status_flags qtmr_status_flags_t
```

List of Quad Timer flags.

```
typedef enum _qtmr_channel_selection qtmr_channel_selection_t
```

List of channel selection.

```
typedef enum _qtmr_dma_enable qtmr_dma_enable_t
```

List of Quad Timer DMA enable.

```
typedef struct _qtmr_config qtmr_config_t
```

Quad Timer config structure.

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

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

```
status_t QTMR_SetupPwm(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t  
pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity,  
uint32_t srcClock_Hz)
```

Sets up Quad timer module for PWM signal output.

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- pwmFreqHz – PWM signal frequency in Hz
- dutyCyclePercent – PWM pulse width, value should be between 0 to 100
0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)
- outputPolarity – true: invert polarity of the output signal, false: no inversion
- srcClock_Hz – Main counter clock in Hz.

Returns

Returns an error if there was error setting up the signal.

```
void QTMR_SetupInputCapture(TMR_Type *base, qtmr_channel_selection_t channel,  
qtmr_input_source_t capturePin, bool inputPolarity, bool  
reloadOnCapture, qtmr_input_capture_edge_t captureMode)
```

Allows the user to count the source clock cycles until a capture event arrives.

The count is stored in the capture register.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- capturePin – Pin through which we receive the input signal to trigger the capture
- inputPolarity – true: invert polarity of the input signal, false: no inversion

- reloadOnCapture – true: reload the counter when an input capture occurs, false: no reload
- captureMode – Specifies which edge of the input signal triggers a capture

TMR_CSCTRL_OFLAG_MASK

TMR_CSCTRL_OFLAG_SHIFT

struct __qtmr_config

#include <fsl_qtmr.h> Quad Timer config structure.

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

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

Public Members

qtmr_primary_count_source_t primarySource

Specify the primary count source

qtmr_input_source_t secondarySource

Specify the secondary count source

bool enableMasterMode

true: Broadcast compare function output to other counters; false no broadcast

bool enableExternalForce

true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter

uint8_t faultFilterCount

Fault filter count

uint8_t faultFilterPeriod

Fault filter period; value of 0 will bypass the filter

qtmr_debug_action_t debugMode

Operation in Debug mode

2.33 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 kAonUART_RST_SHIFT_RSTn

AON UART reset control

enumerator kAonI2C_RST_SHIFT_RSTn

AON I2C reset control

enumerator kAonCAL_RST_SHIFT_RSTn

AON CAL reset control

enumerator kAonQTMR0_RST_SHIFT_RSTn
AON QTMR0 reset control

enumerator kAonLPTMR_RST_SHIFT_RSTn
AON LPTMR reset control

enumerator kAonKPP_RST_SHIFT_RSTn
AON KPP reset control

enumerator kAonLPADC_RST_SHIFT_RSTn
AON LPADC reset control

enumerator kAonLCD_RST_SHIFT_RSTn
AON LCD reset control

enumerator kAonSecurity_RST_SHIFT_RSTn
AON Security reset control

enumerator kAonACMP0_RST_SHIFT_RSTn
AON ACMP reset control

enumerator kAonADVC2P0_RST_SHIFT_RSTn
AON ADVC2P0 reset control

enumerator NotAvail_RSTn
No reset control

typedef enum _SYSCON_RSTn SYSCON_RSTn_t
Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

typedef SYSCON_RSTn_t reset_ip_name_t

void RESET_SetPeripheralReset(*reset_ip_name_t* peripheral)

Assert reset to peripheral.

Asserts reset signal to specified peripheral module.

Parameters

- peripheral – Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_ClearPeripheralReset(*reset_ip_name_t* peripheral)

Clear reset to peripheral.

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

- peripheral – Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_PeripheralReset(*reset_ip_name_t* peripheral)

Reset peripheral module.

Reset peripheral module.

Parameters

- peripheral – Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void RESET_ReleasePeripheralReset(*reset_ip_name_t* peripheral)

Release peripheral module.

Release peripheral module.

Parameters

- peripheral – Peripheral to release. The enum argument contains encoding of reset register and reset bit position in the reset register.

FSL_RESET_DRIVER_VERSION

reset driver version 2.5.0

LPUART_RSTS

LPI2C_RSTS

CTIMER_RSTS

PORT_RSTS_N

PORT_RSTS_N

GPIO_RSTS_N

GPIO_RSTS_N

LPCMP_RSTS

AON_LPADC_RSTS

2.34 ROMAPI Driver

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 property.

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_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.

typedef enum *_flash_property_tag* flash_property_tag_t
Enumeration for various flash properties.

typedef struct *_flash_ffr_config* flash_ffr_config_t
Flash controller paramter config.

typedef struct *_flash_config* flash_config_t
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.

typedef struct *_flash_driver_interface* flash_driver_interface_t
Interface for the flash driver.

typedef struct *_bootloader_tree* bootloader_tree_t
Root of the bootloader API tree.

An instance of this struct resides in read-only memory in the bootloader. It provides a user application access to APIs exported by the bootloader.

FSL_ROMAPI_DRIVER_VERSION
romapi driver version 2.0.0.

FOUR_CHAR_CODE(a, b, c, d)
Constructs the four character code for the Flash driver API key.

ROM_API_BASE
ROM API base address

ROM_API
ROM API base pointer

FLASH_API
FLASH API base pointer

uint32_t ffrBlockBase

uint32_t ffrTotalSize

uint32_t ffrPageSize

uint32_t sectorSize

uint32_t cfpaPageVersion

uint32_t cfpaPageOffset

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.

flash_ffr_config_t ffrConfig

status_t (*flash_init)(*flash_config_t* *config)

status_t (*flash_erase_sector)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

status_t (*flash_program_phrase)(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

status_t (*flash_program_page)(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

status_t (*flash_verify_program)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, uint32_t *failedAddress, uint32_t *failedData)

status_t (*flash_verify_erase_phrase)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

status_t (*flash_verify_erase_page)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

status_t (*flash_verify_erase_sector)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

status_t (*flash_get_property)(*flash_config_t* *config, *flash_property_tag_t* whichProperty, uint32_t *value)

status_t (*ifr_verify_erase_phrase)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

status_t (*ifr_verify_erase_page)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

status_t (*ifr_verify_erase_sector)(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

status_t (*flash_read)(*flash_config_t* *config, uint32_t start, uint8_t *dest, uint32_t lengthInBytes)

uint32_t version

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

void (*run_bootloader)(void *arg)

Function to start the bootloader executing.

const *flash_driver_interface_t* *flash_driver

Internal Flash driver API.

void (*jump)(void *arg)

FSL_COMPONENT_ID

struct *_flash_ffr_config*

#include <fsl_romapi.h> Flash controller paramter config.

struct *_flash_config*

#include <fsl_romapi.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.

struct *_flash_driver_interface*

#include <fsl_romapi.h> Interface for the flash driver.

struct *user_app_boot_invoke_option_t*

#include <fsl_romapi.h>

struct *_bootloader_tree*

#include <fsl_romapi.h> Root of the bootloader API tree.

An instance of this struct resides in read-only memory in the bootloader. It provides a user application access to APIs exported by the bootloader.

union option

Public Members

struct *user_app_boot_invoke_option_t* B

uint32_t U

struct B

2.35 SLCD: Segment LCD Driver

void SLCD_Init(LCD_Type *base, *slcd_config_t* *configure)

Initializes the SLCD, ungates the module clock, initializes the power setting, enables all used plane pins, and sets with interrupt and work mode with the configuration.

Parameters

- base – SLCD peripheral base address.

- `configure` – SLCD configuration pointer. For the configuration structure, many parameters have the default setting and the `SLCD_Getdefaultconfig()` is provided to get them. Use it verified for their applications. The others have no default settings, such as “clk-Config”, and must be provided by the application before calling the `SLCD_Init()` API.

`void SLCD_Deinit(LCD_Type *base)`

Deinitializes the SLCD module, gates the module clock, disables an interrupt, and displays the SLCD.

Parameters

- `base` – SLCD peripheral base address.

`void SLCD_GetDefaultConfig(slcd_config_t *configure)`

Gets the SLCD default configuration structure. The purpose of this API is to get default parameters of the configuration structure for the `SLCD_Init()`. Use these initialized parameters unchanged in `SLCD_Init()` or modify fields of the structure before the calling `SLCD_Init()`. All default parameters of the configure structuration are listed.

```
config.displayMode      = kSLCD_NormalMode;
config.powerSupply      = kSLCD_InternalV13UseChargePump;
config.voltageTrim      = kSLCD_RegulatedVoltageTrim00;
config.lowPowerBehavior = kSLCD_EnabledInWaitStop;
config.interruptSrc     = 0;
config.faultConfig      = NULL;
config.frameFreqIntEnable = false;
```

Parameters

- `configure` – The SLCD configuration structure pointer.

`static inline void SLCD_StartDisplay(LCD_Type *base)`

Enables the SLCD controller, starts generation, and displays the front plane and back plane waveform.

Parameters

- `base` – SLCD peripheral base address.

`static inline void SLCD_StopDisplay(LCD_Type *base)`

Stops the SLCD controller. There is no waveform generator and all enabled pins only output a low value.

Parameters

- `base` – SLCD peripheral base address.

`void SLCD_StartBlinkMode(LCD_Type *base, slcd_blink_mode_t mode, slcd_blink_rate_t rate)`

Starts the SLCD blink mode.

Parameters

- `base` – SLCD peripheral base address.
- `mode` – SLCD blink mode.
- `rate` – SLCD blink rate.

`static inline void SLCD_StopBlinkMode(LCD_Type *base)`

Stops the SLCD blink mode.

Parameters

- base – SLCD peripheral base address.

```
static inline void SLCD_SetBackPlanePhase(LCD_Type *base, uint32_t pinIndx,  
                                         slcd_phase_type_t phase)
```

Sets the SLCD back plane pin phase.

This function sets the SLCD back plane pin phase. “kSLCD_PhaseXActivate” setting means the phase X is active for the back plane pin. “kSLCD_NoPhaseActivate” setting means there is no phase active for the back plane pin. For example, set the back plane pin 20 for phase A.

```
SLCD_SetBackPlanePhase(LCD, 20, kSLCD_PhaseAActivate);
```

Parameters

- base – SLCD peripheral base address.
- pinIndx – SLCD back plane pin index. Range from 0 to 63.
- phase – The phase activates for the back plane pin.

```
static inline void SLCD_SetFrontPlaneSegments(LCD_Type *base, uint32_t pinIndx, uint8_t  
                                              operation)
```

Sets the SLCD front plane segment operation for a front plane pin.

This function sets the SLCD front plane segment on or off operation. Each bit turns on or off the segments associated with the front plane pin in the following pattern: HGFEDCBA (most significant bit controls segment H and least significant bit controls segment A). For example, turn on the front plane pin 20 for phase B and phase C.

```
SLCD_SetFrontPlaneSegments(LCD, 20, (kSLCD_PhaseBActivate | kSLCD_PhaseCAActivate));
```

Parameters

- base – SLCD peripheral base address.
- pinIndx – SLCD back plane pin index. Range from 0 to 63.
- operation – The operation for the segment on the front plane pin. This is a logical OR of the enumeration :: slcd_phase_type_t.

```
static inline void SLCD_SetFrontPlaneOnePhase(LCD_Type *base, uint32_t pinIndx,  
                                              slcd_phase_index_t phaseIndx, bool enable)
```

Sets one SLCD front plane pin for one phase.

This function can be used to set one phase on or off for the front plane pin. It can be call many times to set the plane pin for different phase indexes. For example, turn on the front plane pin 20 for phase B and phase C.

```
SLCD_SetFrontPlaneOnePhase(LCD, 20, kSLCD_PhaseBIndex, true);  
SLCD_SetFrontPlaneOnePhase(LCD, 20, kSLCD_PhaseCIndex, true);
```

Parameters

- base – SLCD peripheral base address.
- pinIndx – SLCD back plane pin index. Range from 0 to 63.
- phaseIndx – The phase bit index slcd_phase_index_t.
- enable – True to turn on the segment for phaseIndx phase false to turn off the segment for phaseIndx phase.

```
static inline uint32_t SLCD_GetFaultDetectCounter(LCD_Type *base)
```

Gets the SLCD fault detect counter.

This function gets the number of samples inside the fault detection sample window.

Parameters

- base – SLCD peripheral base address.

Returns

The fault detect counter. The maximum return value is 255. If the maximum 255 returns, the overflow may happen. Reconfigure the fault detect sample window and fault detect clock prescaler for proper sampling.

```
void SLCD_EnableInterrupts(LCD_Type *base, uint32_t mask)
```

Enables the SLCD interrupt. For example, to enable fault detect complete interrupt and frame frequency interrupt, for FSL_FEATURE_SLCD_HAS_FRAME_FREQUENCY_INTERRUPT enabled case, do the following.

```
SLCD_EnableInterrupts(LCD, kSLCD_FaultDetectCompleteInterrupt | kSLCD_FrameFreqInterrupt);
```

Parameters

- base – SLCD peripheral base address.
- mask – SLCD interrupts to enable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t.

```
void SLCD_DisableInterrupts(LCD_Type *base, uint32_t mask)
```

Disables the SLCD interrupt. For example, to disable fault detect complete interrupt and frame frequency interrupt, for FSL_FEATURE_SLCD_HAS_FRAME_FREQUENCY_INTERRUPT enabled case, do the following.

```
SLCD_DisableInterrupts(LCD, kSLCD_FaultDetectCompleteInterrupt | kSLCD_FrameFreqInterrupt);
```

Parameters

- base – SLCD peripheral base address.
- mask – SLCD interrupts to disable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t.

```
uint32_t SLCD_GetInterruptStatus(LCD_Type *base)
```

Gets the SLCD interrupt status flag.

Parameters

- base – SLCD peripheral base address.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: slcd_interrupt_enable_t.

```
void SLCD_ClearInterruptStatus(LCD_Type *base, uint32_t mask)
```

Clears the SLCD interrupt events status flag.

Parameters

- base – SLCD peripheral base address.
- mask – SLCD interrupt source to be cleared. This is the logical OR of members of the enumeration :: slcd_interrupt_enable_t.

FSL_SLCD_DRIVER_VERSION

SLCD driver version.

enum _slcd_clock_prescaler

SLCD clock prescaler to generate frame frequency.

Values:

enumerator kSLCD_ClkPrescaler00
Prescaler 0.

enumerator kSLCD_ClkPrescaler01
Prescaler 1.

enumerator kSLCD_ClkPrescaler02
Prescaler 2.

enumerator kSLCD_ClkPrescaler03
Prescaler 3.

enumerator kSLCD_ClkPrescaler04
Prescaler 4.

enumerator kSLCD_ClkPrescaler05
Prescaler 5.

enumerator kSLCD_ClkPrescaler06
Prescaler 6.

enumerator kSLCD_ClkPrescaler07
Prescaler 7.

enum _slcd_blink_rate

SLCD blink rate.

Values:

enumerator kSLCD_BlinkRate00
SLCD blink rate is LCD clock/((2¹²)).

enumerator kSLCD_BlinkRate01
SLCD blink rate is LCD clock/((2¹³)).

enumerator kSLCD_BlinkRate02
SLCD blink rate is LCD clock/((2¹⁴)).

enumerator kSLCD_BlinkRate03
SLCD blink rate is LCD clock/((2¹⁵)).

enumerator kSLCD_BlinkRate04
SLCD blink rate is LCD clock/((2¹⁶)).

enumerator kSLCD_BlinkRate05
SLCD blink rate is LCD clock/((2¹⁷)).

enumerator kSLCD_BlinkRate06
SLCD blink rate is LCD clock/((2¹⁸)).

enumerator kSLCD_BlinkRate07
SLCD blink rate is LCD clock/((2¹⁹)).

enum _slcd_power_supply_option

SLCD power supply option.

Values:

enumerator kSLCD_InternalVll3UseChargePump

VLL3 connected to VDD internally, charge pump is used to generate VLL1 and VLL2.

enumerator kSLCD_ExternalVll3UseResistorBiasNetwork

VLL3 is driven externally and resistor bias network is used to generate VLL1 and VLL2.

enumerator kSLCD_ExtetanlVll3UseChargePump

VLL3 is driven externally and charge pump is used to generate VLL1 and VLL2.

enumerator kSLCD_InternalVll1UseChargePump

VIREG is connected to VLL1 internally and charge pump is used to generate VLL2 and VLL3.

enum _slcd_regulated_voltage_trim

SLCD regulated voltage trim parameter, be used to meet the desired contrast.

Values:

enumerator kSLCD_RegulatedVolatgeTrim00

Increase the voltage to 0.91 V.

enumerator kSLCD_RegulatedVolatgeTrim01

Increase the voltage to 1.01 V.

enumerator kSLCD_RegulatedVolatgeTrim02

Increase the voltage to 0.96 V.

enumerator kSLCD_RegulatedVolatgeTrim03

Increase the voltage to 1.06 V.

enumerator kSLCD_RegulatedVolatgeTrim04

Increase the voltage to 0.93 V.

enumerator kSLCD_RegulatedVolatgeTrim05

Increase the voltage to 1.03 V.

enumerator kSLCD_RegulatedVolatgeTrim06

Increase the voltage to 0.98 V.

enumerator kSLCD_RegulatedVolatgeTrim07

Increase the voltage to 1.07 V.

enumerator kSLCD_RegulatedVolatgeTrim08

Increase the voltage to 0.92 V.

enumerator kSLCD_RegulatedVolatgeTrim09

Increase the voltage to 1.02 V.

enumerator kSLCD_RegulatedVolatgeTrim10

Increase the voltage to 0.97 V.

enumerator kSLCD_RegulatedVolatgeTrim11

Increase the voltage to 1.08 V.

enumerator kSLCD_RegulatedVolatgeTrim12

Increase the voltage to 0.94 V.

enumerator kSLCD_RegulatedVolatgeTrim13

Increase the voltage to 1.05 V.

enumerator kSLCD_RegulatedVolatgeTrim14

Increase the voltage to 0.99 V.

enumerator kSLCD_RegulatedVoltageTrim15

Increase the voltage to 1.09 V.

enum _sld_load_adjust

SLCD load adjust to handle different LCD glass capacitance or configure the LCD charge pump clock source. Adjust the LCD glass capacitance if resistor bias network is enabled: kSLCD_LowLoadOrFastestClkSrc - Low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) kSLCD_LowLoadOrIntermediateClkSrc - low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) kSLCD_HighLoadOrIntermediateClkSrc - high load (LCD glass capacitance 8000pF or lower. LCD or GPIO function can be used on Vcap1 and Vcap2 pins) kSLCD_HighLoadOrSlowestClkSrc - high load (LCD glass capacitance 8000pF or lower LCD or GPIO function can be used on Vcap1 and Vcap2 pins) Adjust clock for charge pump if charge pump is enabled: kSLCD_LowLoadOrFastestClkSrc - Fasten clock source (LCD glass capacitance 8000pF or 4000pF or lower if Fast Frame Rate is set) kSLCD_LowLoadOrIntermediateClkSrc - Intermediate clock source (LCD glass capacitance 4000pF or 2000pF or lower if Fast Frame Rate is set) kSLCD_HighLoadOrIntermediateClkSrc - Intermediate clock source (LCD glass capacitance 2000pF or 1000pF or lower if Fast Frame Rate is set) kSLCD_HighLoadOrSlowestClkSrc - slowest clock source (LCD glass capacitance 1000pF or 500pF or lower if Fast Frame Rate is set)

Values:

enumerator kSLCD_LowLoadOrFastestClkSrc

Adjust in low load or selects fastest clock.

enumerator kSLCD_LowLoadOrIntermediateClkSrc

Adjust in low load or selects intermediate clock.

enumerator kSLCD_HighLoadOrIntermediateClkSrc

Adjust in high load or selects intermediate clock.

enumerator kSLCD_HighLoadOrSlowestClkSrc

Adjust in high load or selects slowest clock.

enum _sld_clock_src

SLCD clock source.

Values:

enumerator kSLCD_DefaultClk

Select default clock ERCLK32K.

enumerator kSLCD_AlternateClk1

Select alternate clock source 1 : MCGIRCLK.

enum _sld_alt_clock_div

SLCD alternate clock divider.

Values:

enumerator kSLCD_AltClkDivFactor1

No divide for alternate clock.

enumerator kSLCD_AltClkDivFactor64

Divide alternate clock with factor 64.

enumerator kSLCD_AltClkDivFactor256

Divide alternate clock with factor 256.

enumerator kSLCD_AltClkDivFactor512

Divide alternate clock with factor 512.

enum _slcd_duty_cycle

SLCD duty cycle.

Values:

enumerator kSLCD_1Div1DutyCycle

LCD use 1 BP 1/1 duty cycle.

enumerator kSLCD_1Div2DutyCycle

LCD use 2 BP 1/2 duty cycle.

enumerator kSLCD_1Div3DutyCycle

LCD use 3 BP 1/3 duty cycle.

enumerator kSLCD_1Div4DutyCycle

LCD use 4 BP 1/4 duty cycle.

enumerator kSLCD_1Div5DutyCycle

LCD use 5 BP 1/5 duty cycle.

enumerator kSLCD_1Div6DutyCycle

LCD use 6 BP 1/6 duty cycle.

enumerator kSLCD_1Div7DutyCycle

LCD use 7 BP 1/7 duty cycle.

enumerator kSLCD_1Div8DutyCycle

LCD use 8 BP 1/8 duty cycle.

enum _slcd_phase_type

SLCD segment phase type.

Values:

enumerator kSLCD_NoPhaseActivate

LCD waveform no phase activates.

enumerator kSLCD_PhaseAActivate

LCD waveform phase A activates.

enumerator kSLCD_PhaseBActivate

LCD waveform phase B activates.

enumerator kSLCD_PhaseCAActivate

LCD waveform phase C activates.

enumerator kSLCD_PhaseDAActivate

LCD waveform phase D activates.

enumerator kSLCD_PhaseEActivate

LCD waveform phase E activates.

enumerator kSLCD_PhaseFAActivate

LCD waveform phase F activates.

enumerator kSLCD_PhaseGActivate

LCD waveform phase G activates.

enumerator kSLCD_PhaseHActivate

LCD waveform phase H activates.

enum _slcd_phase_index

SLCD segment phase bit index.

Values:

enumerator kSLCD_PhaseAIndex
LCD phase A bit index.

enumerator kSLCD_PhaseBIndex
LCD phase B bit index.

enumerator kSLCD_PhaseCIndex
LCD phase C bit index.

enumerator kSLCD_PhaseDIndex
LCD phase D bit index.

enumerator kSLCD_PhaseEIndex
LCD phase E bit index.

enumerator kSLCD_PhaseFIndex
LCD phase F bit index.

enumerator kSLCD_PhaseGIndex
LCD phase G bit index.

enumerator kSLCD_PhaseHIndex
LCD phase H bit index.

enum _slcd_display_mode
SLCD display mode.

Values:

enumerator kSLCD_NormalMode
LCD Normal display mode.

enumerator kSLCD_AlternateMode
LCD Alternate display mode. For four back planes or less.

enumerator kSLCD_BlankMode
LCD Blank display mode.

enum _slcd_blink_mode
SLCD blink mode.

Values:

enumerator kSLCD_BlankDisplayBlink
Display blank during the blink period.

enumerator kSLCD_AltDisplayBlink
Display alternate display during the blink period if duty cycle is lower than 5.

enum _slcd_fault_detect_clock_prescaler
SLCD fault detect clock prescaler.

Values:

enumerator kSLCD_FaultSampleFreqDivider1
Fault detect sample clock frequency is 1/1 bus clock.

enumerator kSLCD_FaultSampleFreqDivider2
Fault detect sample clock frequency is 1/2 bus clock.

enumerator kSLCD_FaultSampleFreqDivider4
Fault detect sample clock frequency is 1/4 bus clock.

enumerator kSLCD_FaultSampleFreqDivider8

Fault detect sample clock frequency is 1/8 bus clock.

enumerator kSLCD_FaultSampleFreqDivider16

Fault detect sample clock frequency is 1/16 bus clock.

enumerator kSLCD_FaultSampleFreqDivider32

Fault detect sample clock frequency is 1/32 bus clock.

enumerator kSLCD_FaultSampleFreqDivider64

Fault detect sample clock frequency is 1/64 bus clock.

enumerator kSLCD_FaultSampleFreqDivider128

Fault detect sample clock frequency is 1/128 bus clock.

enum _slcd_fault_detect_sample_window_width

SLCD fault detect sample window width.

Values:

enumerator kSLCD_FaultDetectWindowWidth4SampleClk

Sample window width is 4 sample clock cycles.

enumerator kSLCD_FaultDetectWindowWidth8SampleClk

Sample window width is 8 sample clock cycles.

enumerator kSLCD_FaultDetectWindowWidth16SampleClk

Sample window width is 16 sample clock cycles.

enumerator kSLCD_FaultDetectWindowWidth32SampleClk

Sample window width is 32 sample clock cycles.

enumerator kSLCD_FaultDetectWindowWidth64SampleClk

Sample window width is 64 sample clock cycles.

enumerator kSLCD_FaultDetectWindowWidth128SampleClk

Sample window width is 128 sample clock cycles.

enumerator kSLCD_FaultDetectWindowWidth256SampleClk

Sample window width is 256 sample clock cycles.

enumerator kSLCD_FaultDetectWindowWidth512SampleClk

Sample window width is 512 sample clock cycles.

enum _slcd_interrupt_enable

SLCD interrupt source.

Values:

enumerator kSLCD_FaultDetectCompleteInterrupt

SLCD fault detection complete interrupt source.

enum _slcd_lowpower_behavior

SLCD behavior in low power mode.

Values:

enumerator kSLCD_EnabledInWaitStop

SLCD works in wait and stop mode.

enumerator kSLCD_EnabledInWaitOnly

SLCD works in wait mode and is disabled in stop mode.

enumerator `kSLCD_EnabledInStopOnly`

SLCD works in stop mode and is disabled in wait mode.

enumerator `kSLCD_DisabledInWaitStop`

SLCD is disabled in stop mode and wait mode.

typedef enum `_slcd_clock_prescaler` `slcd_clock_prescaler_t`

SLCD clock prescaler to generate frame frequency.

typedef enum `_slcd_blink_rate` `slcd_blink_rate_t`

SLCD blink rate.

typedef enum `_slcd_power_supply_option` `slcd_power_supply_option_t`

SLCD power supply option.

typedef enum `_slcd_regulated_voltage_trim` `slcd_regulated_voltage_trim_t`

SLCD regulated voltage trim parameter, be used to meet the desired contrast.

typedef enum `_slcd_load_adjust` `slcd_load_adjust_t`

SLCD load adjust to handle different LCD glass capacitance or configure the LCD charge pump clock source. Adjust the LCD glass capacitance if resistor bias network is enabled: `kSLCD_LowLoadOrFastestClkSrc` - Low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) `kSLCD_LowLoadOrIntermediateClkSrc` - low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) `kSLCD_HighLoadOrIntermediateClkSrc` - high load (LCD glass capacitance 8000pF or lower. LCD or GPIO function can be used on Vcap1 and Vcap2 pins) `kSLCD_HighLoadOrSlowestClkSrc` - high load (LCD glass capacitance 8000pF or lower LCD or GPIO function can be used on Vcap1 and Vcap2 pins) Adjust clock for charge pump if charge pump is enabled: `kSLCD_LowLoadOrFastestClkSrc` - Fasten clock source (LCD glass capacitance 8000pF or 4000pF or lower if Fast Frame Rate is set) `kSLCD_LowLoadOrIntermediateClkSrc` - Intermediate clock source (LCD glass capacitance 4000pF or 2000pF or lower if Fast Frame Rate is set) `kSLCD_HighLoadOrIntermediateClkSrc` - Intermediate clock source (LCD glass capacitance 2000pF or 1000pF or lower if Fast Frame Rate is set) `kSLCD_HighLoadOrSlowestClkSrc` - slowest clock source (LCD glass capacitance 1000pF or 500pF or lower if Fast Frame Rate is set)

typedef enum `_slcd_clock_src` `slcd_clock_src_t`

SLCD clock source.

typedef enum `_slcd_alt_clock_div` `slcd_alt_clock_div_t`

SLCD alternate clock divider.

typedef struct `_slcd_clock_config` `slcd_clock_config_t`

SLCD clock configuration structure.

typedef enum `_slcd_duty_cycle` `slcd_duty_cycle_t`

SLCD duty cycle.

typedef enum `_slcd_phase_type` `slcd_phase_type_t`

SLCD segment phase type.

typedef enum `_slcd_phase_index` `slcd_phase_index_t`

SLCD segment phase bit index.

typedef enum `_slcd_display_mode` `slcd_display_mode_t`

SLCD display mode.

typedef enum `_slcd_blink_mode` `slcd_blink_mode_t`

SLCD blink mode.

typedef enum *_slcd_fault_detect_clock_prescaler* slcd_fault_detect_clock_prescaler_t
SLCD fault detect clock prescaler.

typedef enum *_slcd_fault_detect_sample_window_width*
slcd_fault_detect_sample_window_width_t
SLCD fault detect sample window width.

typedef enum *_slcd_interrupt_enable* slcd_interrupt_enable_t
SLCD interrupt source.

typedef enum *_slcd_lowpower_behavior* slcd_lowpower_behavior
SLCD behavior in low power mode.

typedef struct *_slcd_fault_detect_config* slcd_fault_detect_config_t
SLCD fault frame detection configuration structure.

typedef struct *_slcd_config* slcd_config_t
SLCD configuration structure.

struct *_slcd_clock_config*
#include <fsl_slcd.h> SLCD clock configuration structure.

Public Members

slcd_clock_src_t clkSource
Clock source. “slcd_clock_src_t” is recommended to be used. The SLCD is optimized to operate using a 32.768kHz clock input.

slcd_alt_clock_div_t altClkDivider
The divider to divide the alternate clock used for alternate clock source.

slcd_clock_prescaler_t clkPrescaler
Clock prescaler.

struct *_slcd_fault_detect_config*
#include <fsl_slcd.h> SLCD fault frame detection configuration structure.

Public Members

bool faultDetectIntEnable
Fault frame detection interrupt enable flag.

bool faultDetectBackPlaneEnable
True means the pin id fault detected is back plane otherwise front plane.

uint8_t faultDetectPinIndex
Fault detected pin id from 0 to 63.

slcd_fault_detect_clock_prescaler_t faultPrescaler
Fault detect clock prescaler.

slcd_fault_detect_sample_window_width_t width
Fault detect sample window width.

struct *_slcd_config*
#include <fsl_slcd.h> SLCD configuration structure.

Public Members

slcd_power_supply_option_t powerSupply

Power supply option.

slcd_regulated_voltage_trim_t voltageTrim

Regulated voltage trim used for the internal regulator VIREG to adjust to facilitate contrast control.

slcd_clock_config_t *clkConfig

Clock configure.

slcd_load_adjust_t loadAdjust

Load adjust to handle glass capacitance.

slcd_display_mode_t displayMode

SLCD display mode.

slcd_duty_cycle_t dutyCycle

Duty cycle.

slcd_lowpower_behavior lowPowerBehavior

SLCD behavior in low power mode.

uint32_t slcdLowPinEnabled

Setting enabled SLCD pin 0 ~ pin 31. Setting bit n to 1 means enable pin n.

uint32_t slcdHighPinEnabled

Setting enabled SLCD pin 32 ~ pin 63. Setting bit n to 1 means enable pin (n + 32).

uint32_t backPlaneLowPin

Setting back plane pin 0 ~ pin 31. Setting bit n to 1 means setting pin n as back plane. It should never have the same bit setting as the frontPlane Pin.

uint32_t backPlaneHighPin

Setting back plane pin 32 ~ pin 63. Setting bit n to 1 means setting pin (n + 32) as back plane. It should never have the same bit setting as the frontPlane Pin.

slcd_fault_detect_config_t *faultConfig

Fault frame detection configure. If not requirement, set to NULL.

2.36 Trdc_soc

FSL_TRDC_SOC_DRIVER_VERSION

Driver version 1.1.0.

TRDC_MBC_NSE_BLK_CLR_ALL_MEMSEL(x)

TRDC_MBC_NSE_BLK_CLR_ALL_DID_SEL(x)

FSL_FEATURE_TRDC_DOMAIN_COUNT

TRDC feature.

TRDC_MBC_COUNT

TRDC base address convert macro.

TRDC_MBC_OFFSET(x)

TRDC_MBC_ARRAY_STEP

FSL_COMPONENT_ID

2.37 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.

Chapter 3

Middleware

3.1 MultiCore

3.1.1 Multicore SDK

Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

Multicore SDK (MCSDK) Release Notes

Overview These are the release notes for the NXP Multicore Software Development Kit (MCSDK) version 25.06.00.

This software package contains components for efficient work with multicore devices as well as for the multiprocessor communication.

What is new

- eRPC [CHANGELOG](#)
- RMPmsg-Lite [CHANGELOG](#)
- MCMgr [CHANGELOG](#)
- Supported evaluation boards (multicore examples):
 - LPCXpresso55S69
 - FRDM-K32L3A6
 - MIMXRT1170-EVKB
 - MIMXRT1160-EVK
 - MIMXRT1180-EVK
 - MCX-N5XX-EVK
 - MCX-N9XX-EVK
 - FRDM-MCXN947
 - MIMXRT700-EVK

- KW47-EVK
- KW47-LOC
- FRDM-MCXW72
- MCX-W72-EVK
- Supported evaluation boards (multiprocessor examples):
 - LPCXpresso55S36
 - FRDM-K22F
 - FRDM-K32L2B
 - MIMXRT685-EVK
 - MIMXRT1170-EVKB
 - MIMXRT1180
 - FRDM-MCXN236
 - FRDM-MCXC242
 - FRDM-MCXC444
 - MCX-N9XX-EVK
 - FRDM-MCXN947
 - MIMXRT700-EVK

Development tools The Multicore SDK (MCSDK) was compiled and tested with development tools referred in: [Development tools](#)

Release contents This table describes the release contents. Not all MCUXpresso SDK packages contain the whole set of these components.

Deliverable			Location
Multicore SDK	location	<MCUXpressoSDK_install_dir>/middleware/multicore/	
<MCSDK_dir> Documentation			<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call component			<MCSDK_dir>/erpc/
Multicore Manager component			<MCSDK_dir>/mcmgr/
RPMMsg-Lite			<MCSDK_dir>/rpmmsg_lite/
Multicore demo applications			<MCUXpressoSDK_install_dir>/examples/multicore_examples/
Multiprocessor demo applications			<MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/

Multicore SDK release overview Together, the Multicore SDK (MCSDK) and the MCUXpresso SDK (SDK) form a framework for the development of software for NXP multicore devices. The MCSDK release consists of the following elementary software components for multicore:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR) - included just in SDK for multicore devices
- Remote Processor Messaging - Lite (RPMMsg-Lite) - included just in SDK for multicore devices

The MCSDK is also accompanied with documentation and several multicore and multiprocessor demo applications.

Demo applications The multicore demo applications demonstrate the usage of the MCSDK software components on supported multicore development boards.

The following multicore demo applications are located together with other MCUXpresso SDK examples in

the <MCUXpressoSDK_install_dir>/examples/multicore_examples subdirectories.

- erpc_matrix_multiply_mu
- erpc_matrix_multiply_mu_rtos
- erpc_matrix_multiply_rpmsg
- erpc_matrix_multiply_rpmsg_rtos
- erpc_two_way_rpc_rpmsg_rtos
- freertos_message_buffers
- hello_world
- multicore_manager
- rpmsg_lite_pingpong
- rpmsg_lite_pingpong_rtos
- rpmsg_lite_pingpong_tzm

The eRPC multicore component can be leveraged for inter-processor communication and remote procedure calls between SoCs / development boards.

The following multiprocessor demo applications are located together with other MCUXpresso SDK examples in

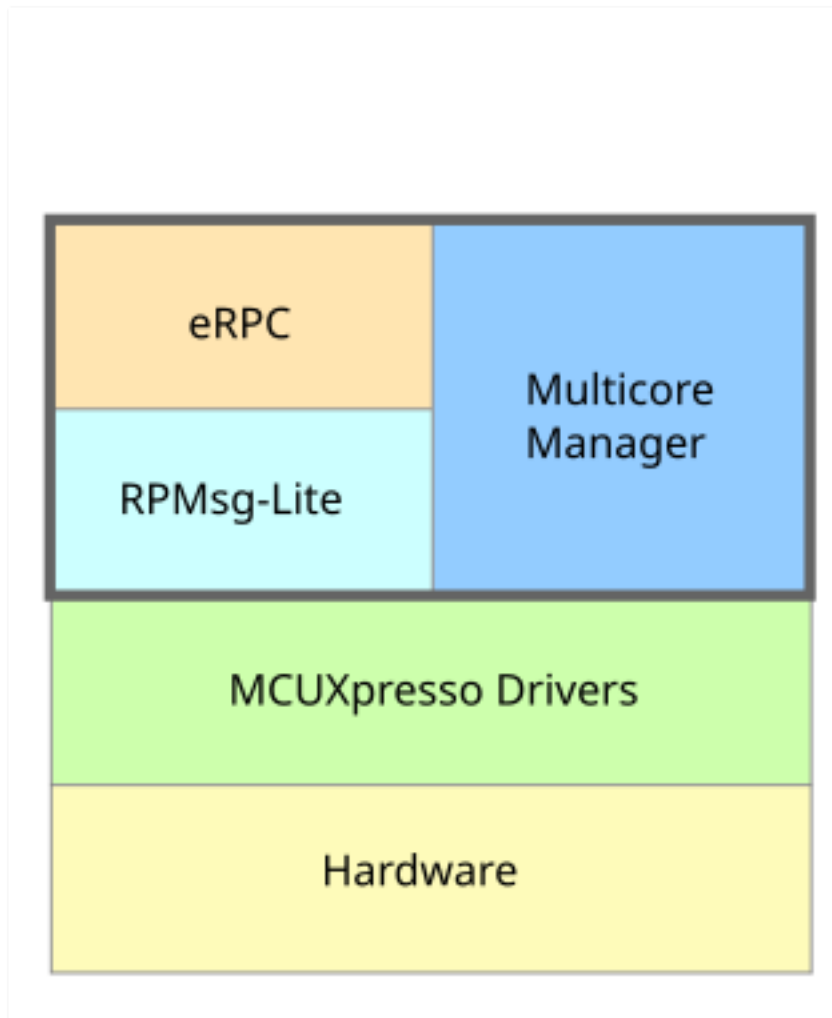
the <MCUXpressoSDK_install_dir>/examples/multiprocessor_examples subdirectories.

- erpc_client_matrix_multiply_spi
- erpc_server_matrix_multiply_spi
- erpc_client_matrix_multiply_uart
- erpc_server_matrix_multiply_uart
- erpc_server_dac_adc
- erpc_remote_control

Getting Started with Multicore SDK (MCSDK)

Overview Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

The following figure highlights the layers and main software components of the MCSDK.

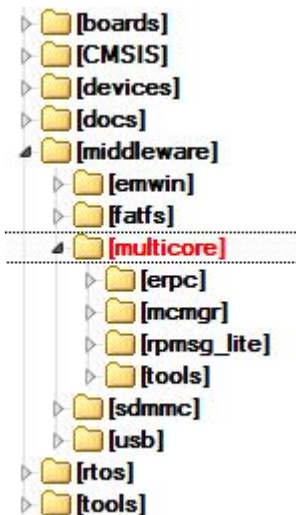


All the MCSDK-related files are located in `<MCUXpressoSDK_install_dir>/middleware/multicore` folder.

For supported toolchain versions, see the *Multicore SDK v25.06.00 Release Notes* (document MCS-DKRN). For the latest version of this and other MCSDK documents, visit www.nxp.com.

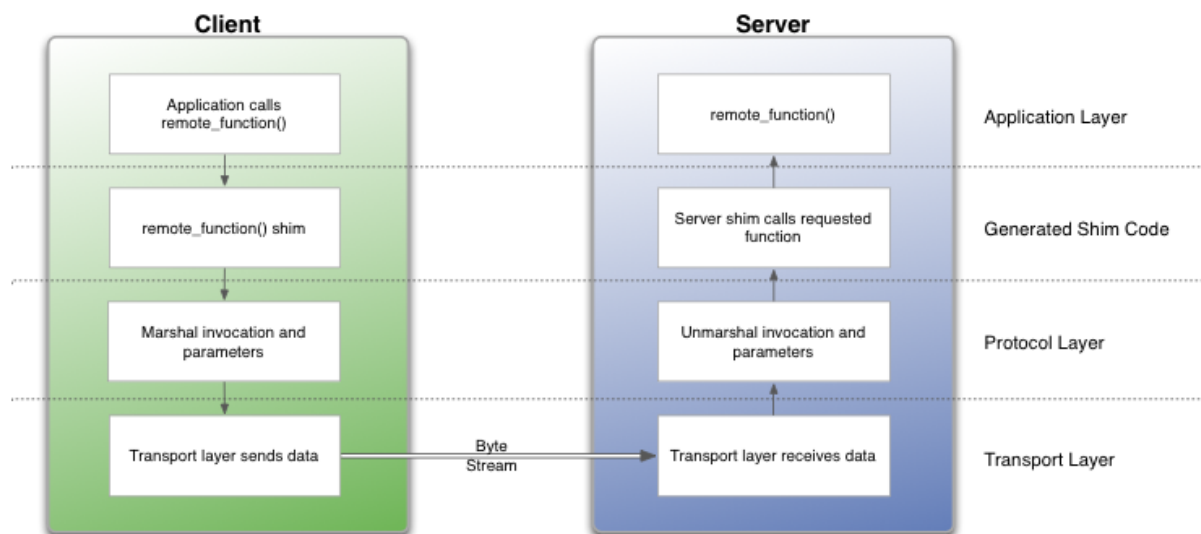
Multicore SDK (MCSDK) components The MCSDK consists of the following software components:

- **Embedded Remote Procedure Call (eRPC):** This component is a combination of a library and code generator tool that implements a transparent function call interface to remote services (running on a different core).
- **Multicore Manager (MCMGR):** This library maintains information about all cores and starts up secondary/auxiliary cores.
- **Remote Processor Messaging - Lite (RPMsg-Lite):** Inter-Processor Communication library.



Embedded Remote Procedure Call (eRPC) The Embedded Remote Procedure Call (eRPC) is the RPC system created by NXP. The RPC is a mechanism used to invoke a software routine on a remote system via a simple local function call.

When a remote function is called by the client, the function's parameters and an identifier for the called routine are marshaled (or serialized) into a stream of bytes. This byte stream is transported to the server through a communications channel (IPC, TCP/IP, UART, and so on). The server unmarshals the parameters, determines which function was invoked, and calls it. If the function returns a value, it is marshaled and sent back to the client.



RPC implementations typically use a combination of a tool (erpcgen) and IDL (interface definition language) file to generate source code to handle the details of marshaling a function's parameters and building the data stream.

Main eRPC features:

- Scalable from BareMetal to Linux OS - configurable memory and threading policies.
- Focus on embedded systems - intrinsic support for C, modular, and lightweight implementation.
- Abstracted transport interface - RPMsg is the primary transport for multicore, UART, or SPI-based solutions can be used for multichip.

The eRPC library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc` folder. For detailed information about the eRPC, see the documentation available in the

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc folder.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems.

The main MCMGR features:

- Maintains information about all cores in system.
- Secondary/auxiliary cores startup and shutdown.
- Remote core monitoring and event handling.

The MCMGR library is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr folder. For detailed information about the MCMGR library, see the documentation available in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/doc folder.

Remote Processor Messaging Lite (RPMsg-Lite) RPMsg-Lite is a lightweight implementation of the RPMsg protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system. Compared to the legacy OpenAMP implementation, RPMsg-Lite offers a code size reduction, API simplification, and improved modularity.

The main RPMsg protocol features:

- Shared memory interprocessor communication.
- Virtio-based messaging bus.
- Application-defined messages sent between endpoints.
- Portable to different environments/platforms.
- Available in upstream Linux OS.

The RPMsg-Lite library is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite folder. For detailed information about the RPMsg-Lite, see the RPMsg-Lite User's Guide located in the <MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite/doc folder.

MCSDK demo applications Multicore and multiprocessor example applications are stored together with other MCUXpresso SDK examples, in the dedicated multicore subfolder.

Location		Folder
Multicore projects	example	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/
Multiprocessor projects	example	<MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/<application_name>/

See the *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) and *Getting Started with MCUXpresso SDK for XXX Derivatives* documents for more information about the MCUXpresso SDK example folder structure and the location of individual files that form the example application projects. These documents also contain information about building, running, and debugging multicore demo applications in individual supported IDEs. Each example application also contains a readme file that describes the operation of the example and required setup steps.

Inter-Processor Communication (IPC) levels The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Particular ways and levels of IPC are described in this chapter.

IPC using low-level drivers

The NXP multicore SoCs are equipped with peripheral modules dedicated for data exchange between individual cores. They deal with the Mailbox peripheral for LPC parts and the Messaging Unit (MU) peripheral for Kinetis and i.MX parts. The common attribute of both modules is the ability to provide a means of IPC, allowing multiple CPUs to share resources and communicate with each other in a simple manner.

The most lightweight method of IPC uses the MCUXpresso SDK low-level drivers for these peripherals. Using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via the dedicated registers (could be a scalar or a pointer to shared memory) and also to trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific multicore device. The MCUXpresso SDK is accompanied with the RPMsg-Lite documentation that shows how to use this API in multicore applications.

Messaging mechanism

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to send between multiple endpoints created on each of the CPUs. The RPMsg-Lite library of the MCSDK provides this ability and serves as the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for messages exchange without the need of a locking mechanism.

The RPMsg-Lite provides the abstraction layer and can be easily ported to different multicore platforms and environments (Operating Systems). The advantages of such a messaging system are ease of use (there is no need to study behavior of the used underlying hardware) and smooth application code portability between platforms due to unified messaging API.

However, this costs several kB of code and data memory. The MCUXpresso SDK is accompanied by the RPMsg-Lite documentation and several multicore examples. You can also obtain the latest RPMsg-Lite code from the GitHub account github.com/nxp-mcuxpresso/rpmsg-lite.

Remote procedure calls

To facilitate the IPC even more and to allow the remote functions invocation, the remote procedure call mechanism can be implemented. The eRPC of the MCSDK serves for these purposes and allows the ability to invoke a software routine on a remote system via a simple local function call. Utilizing different transport layers, it is possible to communicate between individual cores of multicore SoCs (via RPMsg-Lite) or between separate processors (via SPI, UART, or TCP/IP). The eRPC is mostly applicable to the MPU parts with enough of memory resources like i.MX parts.

The eRPC library allows you to export existing C functions without having to change their prototypes (in most cases). It is accompanied by the code generator tool that generates the shim code for serialization and invocation based on the IDL file with definitions of data types and remote interfaces (API).

If the communicating peer is running as a Linux OS user-space application, the generated code can be either in C/C++ or Python.

Using the eRPC simplifies the access to services implemented on individual cores. This way, the following types of applications running on dedicated cores can be easily interfaced:

- Communication stacks (USB, Thread, Bluetooth Low Energy, Zigbee)
- Sensor aggregation/fusion applications
- Encryption algorithms
- Virtual peripherals

The eRPC is publicly available from the following GitHub account: github.com/EmbeddedRPC/erpc. Also, the MCUXpresso SDK is accompanied by the eRPC code and several multicore and multiprocessor eRPC examples.

The mentioned IPC levels demonstrate the scalability of the Multicore SDK library. Based on application needs, different IPC techniques can be used. It depends on the complexity, required speed, memory resources, system design, and so on. The MCSDK brings users the possibility for quick and easy development of multicore and multiprocessor applications.

Changelog Multicore SDK

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

[25.06.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.14.0
 - eRPC generator (erpcgen) v1.14.0
 - Multicore Manager (MCMgr) v5.0.0
 - RPSMsg-Lite v5.2.0

[25.03.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.7
 - RPSMsg-Lite v5.1.4

[24.12.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.6
 - RPSMsg-Lite v5.1.3

[2.16.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.5
 - RPSMsg-Lite v5.1.2

[2.15.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.12.0
 - eRPC generator (erpcgen) v1.12.0
 - Multicore Manager (MCMgr) v4.1.5
 - RPSMsg-Lite v5.1.1

[2.14.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.11.0
 - eRPC generator (erpcgen) v1.11.0
 - Multicore Manager (MCMgr) v4.1.4
 - RPSMsg-Lite v5.1.0

[2.13.0_imxrt1180a0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.10.0
 - eRPC generator (erpcgen) v1.10.0
 - Multicore Manager (MCMgr) v4.1.3
 - RPSMsg-Lite v5.0.0

[2.13.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.10.0
 - eRPC generator (erpcgen) v1.10.0
 - Multicore Manager (MCMgr) v4.1.3
 - RPSMsg-Lite v5.0.0

[2.12.0_imx93]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.1
 - eRPC generator (erpcgen) v1.9.1
 - Multicore Manager (MCMgr) v4.1.2
 - RPSMsg-Lite v4.0.1

[2.12.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.1
 - eRPC generator (erpcgen) v1.9.1
 - Multicore Manager (MCMgr) v4.1.2
 - RPSMsg-Lite v4.0.0

[2.11.1]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.0
 - eRPC generator (erpcgen) v1.9.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.2.1

[2.11.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.0
 - eRPC generator (erpcgen) v1.9.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.2.0

[2.10.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.8.1
 - eRPC generator (erpcgen) v1.8.1
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.1.2

[2.9.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.8.0
 - eRPC generator (erpcgen) v1.8.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.1.1

[2.8.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.4
 - eRPC generator (erpcgen) v1.7.4
 - Multicore Manager (MCMgr) v4.1.0
 - RPSMsg-Lite v3.1.0

[2.7.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.3
 - eRPC generator (erpcgen) v1.7.3
 - Multicore Manager (MCMgr) v4.1.0
 - RPSMsg-Lite v3.0.0

[2.6.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.2
 - eRPC generator (erpcgen) v1.7.2
 - Multicore Manager (MCMgr) v4.0.3
 - RPSMsg-Lite v2.2.0

[2.5.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.1
 - eRPC generator (erpcgen) v1.7.1
 - Multicore Manager (MCMgr) v4.0.2
 - RPSMsg-Lite v2.0.2

[2.4.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.0
 - eRPC generator (erpcgen) v1.7.0
 - Multicore Manager (MCMgr) v4.0.1
 - RPSMsg-Lite v2.0.1

[2.3.1]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.6.0
 - eRPC generator (erpcgen) v1.6.0
 - Multicore Manager (MCMgr) v4.0.0
 - RPSMsg-Lite v1.2.0

[2.3.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.5.0
 - eRPC generator (erpcgen) v1.5.0
 - Multicore Manager (MCMgr) v3.0.0
 - RPSMsg-Lite v1.2.0

[2.2.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.4.0
 - eRPC generator (erpcgen) v1.4.0
 - Multicore Manager (MCMgr) v2.0.1
 - RPSMsg-Lite v1.1.0

[2.1.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.3.0
 - eRPC generator (erpcgen) v1.3.0

[2.0.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.2.0
 - eRPC generator (erpcgen) v1.2.0
 - Multicore Manager (MCMgr) v2.0.0
 - RPSMsg-Lite v1.0.0

[1.1.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.1.0
 - Multicore Manager (MCMgr) v1.1.0
 - Open-AMP / RPSMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

[1.0.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.0.0
 - Multicore Manager (MCMgr) v1.0.0
 - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

Multicore SDK Components

RPMSG-Lite

MCUXpresso SDK : mcuxsdk-middleware-rpmsg-lite

Overview This repository is for MCUXpresso SDK RPMSG-Lite middleware delivery and it contains RPMSG-Lite component 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](#) for the complete delivery of MCUXpresso SDK to be able to build and run RPMSG-Lite examples that are based on mcux-sdk-middleware-rpmsg-lite component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [RPMSG-Lite - 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 rpmsg-lite project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

RPMSG-Lite This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (<https://github.com/OpenAMP/open-amp>), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For Further documentation, please look at doxygen documentation at: <https://nxp-mcuxpresso.github.io/rpmsg-lite/>

Motivation to create RPMsg-Lite There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

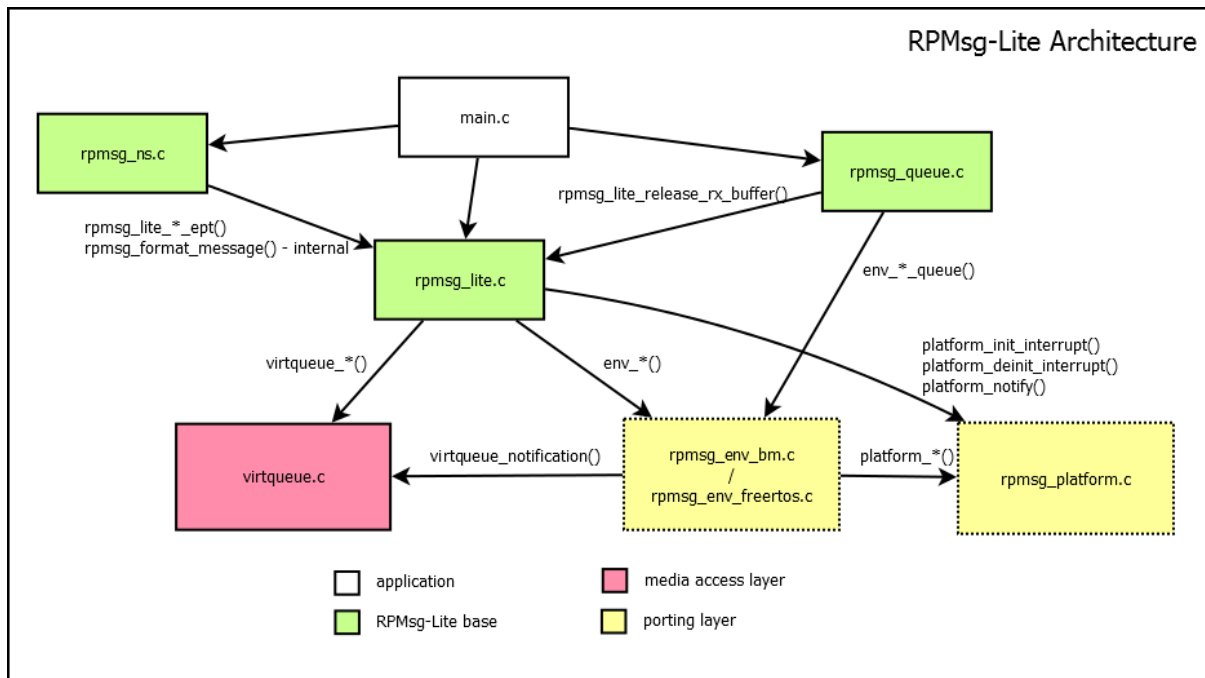
Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

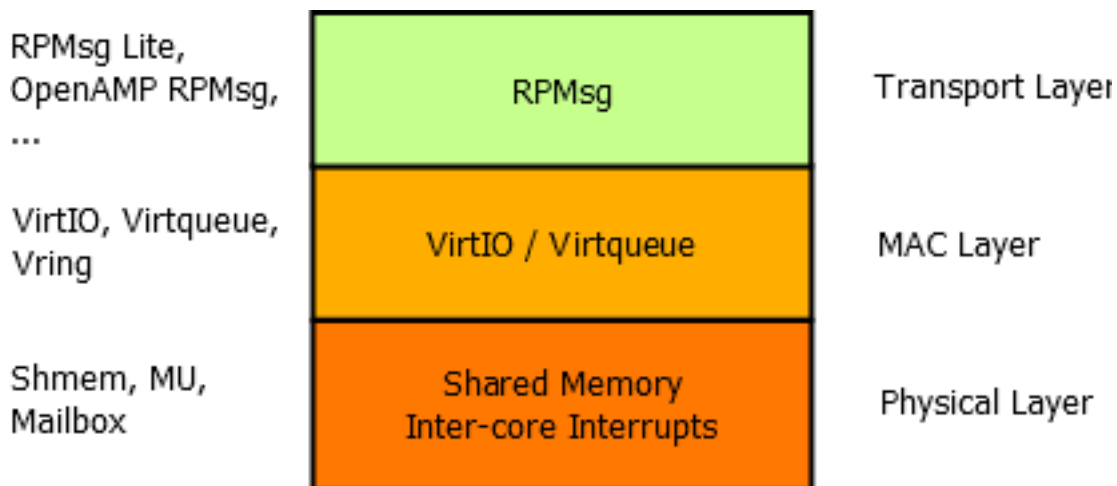
Implementation The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in `rpmsg_lite.c`. Two optional components are used to implement a blocking receive API (in `rpmsg_queue.c`) and dynamic “named” endpoint creation and deletion announcement service (in `rpmsg_ns.c`).

The actual “media access” layer is implemented in `virtqueue.c`, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as `vring` or `virtqueue`.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in `rpmsg_env_bm.c`, and the FreeRTOS environment is implemented in `rpmsg_env_freertos.c` etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in `rpmsg_platform.c` and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional) This subcomponent is optional and requires implementation of the `env_*_queue()` functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional) This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action

in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage The application should put the `/rpmmsg_lite/lib/include` directory to the include path and in the application, include either the `rpmmsg_lite.h` header file, or optionally also include the `rpmmsg_queue.h` and/or `rpmmsg_ns.h` files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, `rpmmsg_env_myrtos.c`) and to include it in the project build.

The initialization of the stack is done by calling the `rpmmsg_lite_master_init()` on the master side and the `rpmmsg_lite_remote_init()` on the remote side. This initialization function must be called prior to any RPMMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the `rpmmsg_lite_create_ept()` function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the `RL_USE_STATIC_API` option is set to 1. If not, the stack internally calls `env_alloc()` to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the `rpmmsg_queue_create()` function must be called before calling `rpmmsg_lite_create_ept()`. The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to `rpmmsg_queue_rx_cb()`. Then, it is possible to use `rpmmsg_queue_receive()` function to listen on a queue object for incoming messages. The `rpmmsg_lite_send()` function is used to send messages to the other side.

The RPMMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.

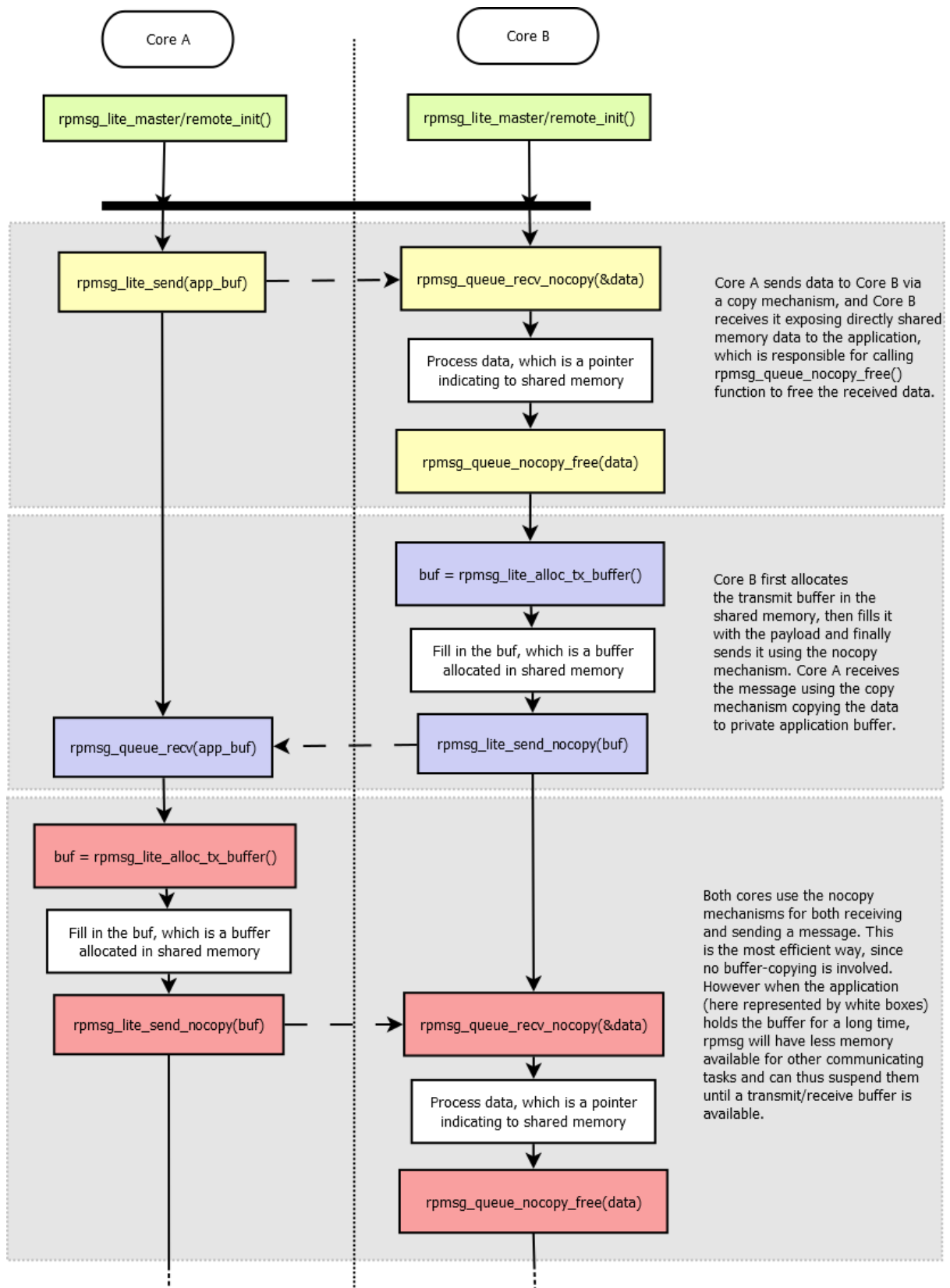
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the `rpmmsg_lite_alloc_tx_buffer()` function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the `rpmmsg_lite_alloc_tx_buffer()` size output parameter).
- Call the `rpmmsg_lite_send_nocopy()` function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the `rpmmsg_lite_send_nocopy()` function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the `rpmmsg_queue_rcv_nocopy()` function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the `rpmmsg_queue_nocopy_free()` function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMMsg-Lite objects he has created in case of deinitialization. In order to do this, the function `rpmmsg_queue_destroy()` is used to destroy a queue, `rpmmsg_lite_destroy_ept()` is used to destroy an endpoint and finally, `rpmmsg_lite_deinit()` is used to deinitialize the RPMMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples RPSMsg Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages. To get the board list with multicore support (RPSMsg Lite included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded,

see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for RPMsg_Lite multicore examples with 'rpmsg_lite_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg_Lite multicore examples is using the [mcuxsdk-manifests](#) Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in [readme section](#). Once done the armgcc rpmsg_lite examples can be found in

mcuxsdk/examples/_<board_name>/multicore_examples

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the RPMsg_Lite examples use the 'rpmsg_lite_' name prefix.

Notes

Environment layers implementation Several environment layers are provided in lib/rpmsg_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg_env_bm.c
- rpmsg_env_freertos.c
- rpmsg_env_xos.c
- rpmsg_env_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

Shared memory configuration It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in linker file is also a good practise, it is recommended to use linker files from MCUXpressoSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

Configuration options The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg_default_config.h header file. This configuration can be customized by the user by including rpmsg_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

Config- uration option	De- fault value	Usage
RL_MS_PE (1)		Delay in milliseconds used in non-blocking API functions for polling.
RL_BUFFE (496)		Size of the buffer payload, it must be equal to (240, 496, 1008, ...) [$2^n - 16$]
RL_BUFFE (2)		Number of the buffers, it must be power of two (2, 4, ...)
RL_API_H (1)		Zero-copy API functions enabled/disabled.
RL_USE_S' (0)		Static API functions (no dynamic allocation) enabled/disabled.
RL_USE_D (0)		Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.
RL_CLEAF (0)		Clearing used buffers before returning back to the pool of free buffers enabled/disabled.
RL_USE_M (0)		When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.
RL_USE_E (0)		When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).
RL_DEBU (0)		When enabled buffer pointers passed to <code>rpmsg_lite_send_nocopy()</code> and <code>rpmsg_lite_release_rx_buffer()</code> functions (enabled by <code>RL_API_HAS_ZEROCOPY</code> config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.
RL_ALLO (0)		When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).
RL_ALLO (0)		It allows to define custom shared memory configuration and replacing the shared memory related global settings from <code>rpmsg_config.h</code> . This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg-Lite instances use the same shared memory arrangement as defined by common config macros).
RL_ASSERT	see <code>rpmsg</code>	Assert implementation.

How to format rpmsg-lite code To format code, use the application developed by Google, named *clang-format*. This tool is part of the [llvm](#) project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the `.clang-format` file, placed in a root of the rpmsg-lite directory where Python script `run_clang_format.py` can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

References

[1] M. Novak, M. Cingel, **Lockless Shared Memory Based Multicore Communication Protocol**
Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

Changelog RPMMSG-Lite All notable changes to this project will be documented in this file. The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

Unreleased

Fixed

- Fixed CERT-C INT31-C violation in platform_notify function in rpmsg_platform.c for imxrt700_m33, imxrt700_hifi4, imxrt700_hifi1 platforms

v5.2.0

Added

- Add MCXL20 porting layer and unit testing
- New utility macro RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

Changed

- Change rpmsg_platform.c to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.
- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

Fixed

- Fixed env_wait_for_link_up function to handle timeout in link state checks for baremetal and qnx environment, RL_BLOCK mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure RL_PLATFORM_HIGHEST_LINK_ID remains within safe range for 16-bit casting in virtqueue ID creation.
- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in shmem_length - (uint32_t)RL_VRING_OVERHEAD and shmem_length - 2U * shmem_config.vring_size expressions.
- Fixed CERT INT31-C violation in platform_interrupt_disable() and similar functions by replacing unsafe cast from uint32_t to int32_t with a return of 0 constant.
- Fixed unsigned integer underflow in rpmsg_lite_alloc_tx_buffer() where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in rpmsg_lite.c where size parameter was cast from uint32_t to uint16_t without proper validation.
 - Applied consistent masking approach to both size and flags parameters: (uint16_t)(value & 0xFFFFU).
 - This fix prevents potential data loss when size values exceed 65535.

- Fixed CERT INT31-C violation in `env_memset` functions by explicitly converting `int32_t` values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard `memset()` function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
 - `rpmsg_env_freertos.c`: Added validation before converting `int32_t` to `UBaseType_t`.
 - `rpmsg_env_qnx.c`: Fixed format string and added validation before assigning to `mqstat` fields.
 - `rpmsg_env_threadx.c`: Added validation to prevent integer overflow and negative values.
 - `rpmsg_env_xos.c`: Added range checking before casting to `uint16_t`.
 - `rpmsg_env_zephyr.c`: Added validation before passing values to `k_msgq_init`.
- Fixed a CERT INT31-C compliance issue in `env_get_current_queue_size()` function where an unsigned queue count was cast to a signed `int32_t` without proper validation, which could lead to lost or misinterpreted data if queue size exceeded `INT32_MAX`.
- Fixed CERT INT31-C violation in `rpmsg_platform.c` where `memcmp()` return value (signed int) was compared with unsigned constant without proper type handling.
- Fixed CERT INT31-C violation in `rpmsg_platform.c` where casting from `uint32_t` to `uint16_t` could potentially result in data loss. Changed length variable type from `uint16_t` to `uint32_t` to properly handle memory address differences without truncation.
- Fixed potential integer overflow in `env_sleep_msec()` function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.
- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on `imx943` platform.
- Added `RL_MAX_BUFFER_COUNT` (32768U) and `RL_MAX_VRING_ALIGN` (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in `vring_need_event()`, added cast to `uint16_t` for each operand.

v5.1.4 - 27-Mar-2025

Added

- Add KW43B43 porting layer

Changed

- Doxygen bump to version 1.9.6

v5.1.3 - 13-Jan-2025

Added

- Memory cache management of shared memory. Enable with `#define RL_USE_DCACHE (1)` in `rpmsg_config.h` in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for imx95, imxrt700, mcmxw71x, mcmxw72x, kw47b42 added.

v5.1.2 - 08-Jul-2024

Changed

- Zephyr-related changes.
- Minor Misra corrections.

v5.1.1 - 19-Jan-2024

Added

- Test suite provided.
- Zephyr support added.

Changed

- Minor changes in platform and env. layers, minor test code updates.

v5.1.0 - 02-Aug-2023

Added

- RPMsg-Lite: Added aarch64 support.

Changed

- RPMsg-Lite: Increased the queue size to $(2 * RL\_BUFFER\_COUNT)$ to cover zero copy cases.
- Code formatting using LLVM16.

Fixed

- Resolved issues in ThreadX env. layer implementation.

v5.0.0 - 19-Jan-2023

Added

- Timeout parameter added to `rpmsg_lite_wait_for_link_up` API function.

Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- VRING_SIZE is set based on number of used buffers now (as calculated in vring_init) - updated for all platforms that are not communicating to Linux rpmsg counterpart.

Fixed

- Fixed wrong RL_VRING_OVERHEAD macro comment in platform.h files
- Misra corrections.

v4.0.0 - 20-Jun-2022

Added

- Added support for custom shared memory arrangement per the RPMsg_Lite instance.
- Introduced new rpmsg_lite_wait_for_link_up() API function - this allows to avoid using busy loops in rtos environments, GitHub PR [#21](#).

Changed

- Adjusted rpmsg_lite_is_link_up() to return RL_TRUE/RL_FALSE.

v3.2.0 - 17-Jan-2022

Added

- Added support for i.MX8 MP multicore platform.

Changed

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR [#14](#).
- Aligned rpmsg_env_xos.c and some platform layers to latest static allocation support.

Fixed

- Minor Misra and typo corrections, GitHub PR [#19](#), [#20](#).

v3.1.2 - 16-Jul-2021

Added

- Addressed MISRA 21.6 rule violation in rpmsg_env.h (use SDK's PRINTF in MCUXpressoSDK examples, otherwise stdio printf is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

Fixed

- Fixed incorrect description of the `rpmsg_lite_get_endpoint_from_addr` function.

Changed

- Updated `RL_BUFFER_COUNT` documentation (issue [#10](#)).
- Updated `imxrt600_hifi4` platform layer.

v3.1.1 - 15-Jan-2021

Added

- Introduced `RL_ALLOW_CONSUMED_BUFFERS_NOTIFICATION` config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

Changed

- Several MISRA C-2012 violations addressed.

v3.1.0 - 22-Jul-2020

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in `rpmsg_lite_rx_callback()` for QNX env.
- Correction of `rpmsg_lite_instance` structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

Changed

- Clang update to v10.0.0, code re-formatted.

v3.0.0 - 20-Dec-2019

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.
- Code formatted.

v2.2.0 - 20-Mar-2019**Added**

- Added configuration macro `RL_DEBUG_CHECK_BUFFERS`.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the `RL_USE_ENVIRONMENT_CONTEXT` configuration macro).
- Data types consolidation.

v1.1.0 - 28-Apr-2017**Added**

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremetal and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

Multicore Manager**MCUXpresso SDK : mcuxsdk-middleware-mcmgr (Multicore Manager)**

Overview This repository is for MCUXpresso SDK Multicore Manager middleware delivery and it contains Multicore Manager component 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](#) for the complete delivery of MCUXpresso SDK to be able to build and run Multicore Manager examples that are based on mcux-sdk-middleware-mcmgr component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

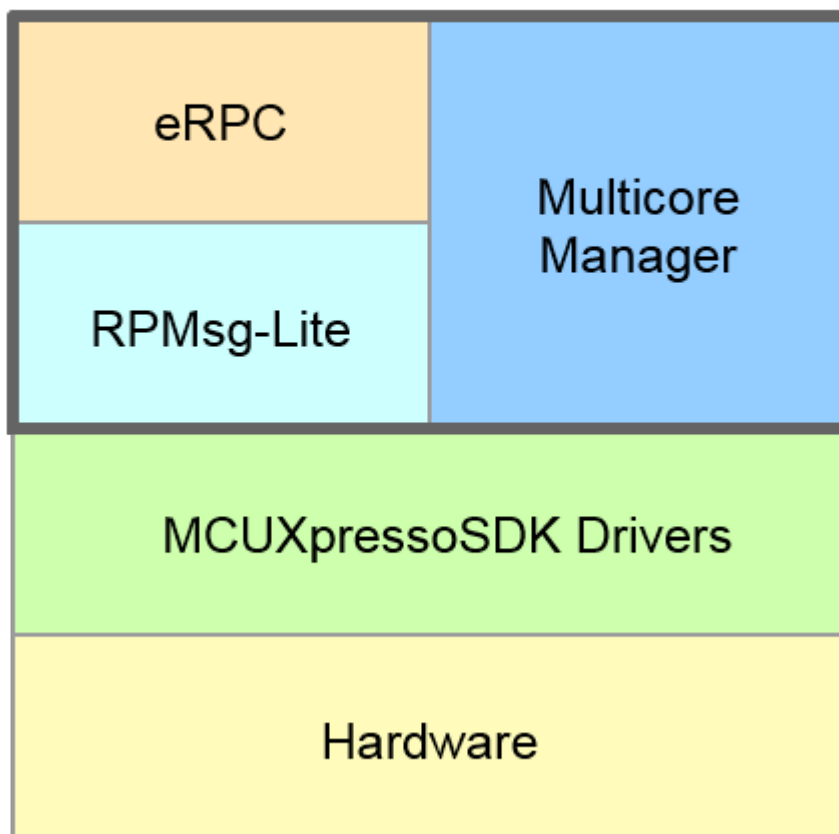
Visit [Multicore Manager - Documentation](#) to review details on the contents in this sub-repo.

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 mcmgr project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

Usage of the MCMGR software component The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,
    ↪ starting synchronously. */
    MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
    .
    .
    .
    /* Stop secondary core execution. */
    MCMGR_StopCore(kMCMGR_Core1);
}
```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR_StopCore / MCMGR_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR_RegisterEvent() API. Triggering these events is done using the MCMGR_TriggerEvent() API. mcmgr_event_type_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()
↪ called on the secondary core side */
void RPMsgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPMsgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
```

(continues on next page)

(continued from previous page)

```

/* Register the application event before starting the secondary core */
MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPSMsgRemoteReadyEventHandler, (void_)
↪)RPSMsgRemoteReadyEventData);

/* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpsmsg_lite_base address_
↪as startup data, starting synchronously. */
MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpsmsg_lite_
↪base, kMCMGR_Start_Synchronous);

/* Wait until the secondary core application signals the rpsmsg remote has been initialized and is ready to_
↪communicate. */
while(APP_RPSMSG_READY_EVENT_DATA != RPSMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
.
.
.
}

```

Code for the secondary side:

```

#include "mcmgr.h"

#define APP_RPSMSG_READY_EVENT_DATA (1)

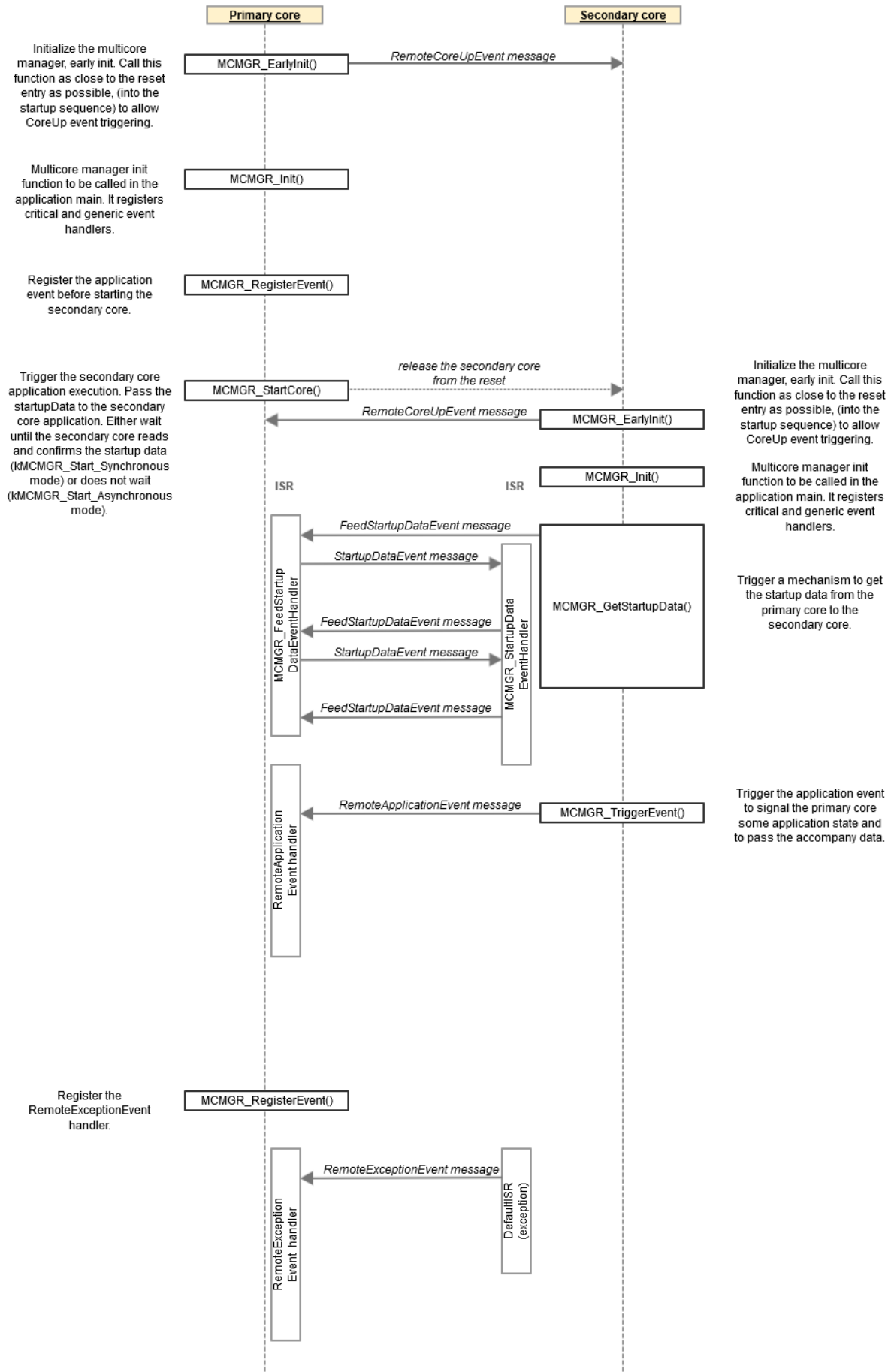
void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
    .
    .
    .

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RPSMSG_
    ↪READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPSMSG_
    ↪READY_EVENT_DATA);
    .
    .
    .
}

```

MCMGR Data Exchange Diagram The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.



Changelog Multicore Manager All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

Unreleased

Added

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.

v5.0.0

Added

- Added MCMGR_BUSY_POLL_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more then two cores. Breaking API change by adding parameter `coreNum` specifying core number in functions bellow.
 - MCMGR_GetStartupData(uint32_t *startupData, mcmgr_core_t coreNum)
 - MCMGR_TriggerEvent(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
 - MCMGR_TriggerEventForce(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
 - typedef void (*mcmgr_event_callback_t)(uint16_t data, void *context, mcmgr_core_t coreNum);

When registering the event with function MCMGR_RegisterEvent() user now needs to provide `callbackData` pointer to array of elements per every core in system (see README.md for example). In case of systems with only two cores the `coreNum` in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: Porting-GuideTo_v5.md

- Updated all porting files to support new MCMGR API.
- Added new platform specific include file `mcmgr_platform.h`. It will contain common platform specific macros that can be then used in `mcmgr` and application. e.g. platform core count MCMGR_CORECOUNT 4.
- Move all header files to new `inc` directory.
- Added new platform-specific include files `inc/platform/<platform_name>/mcmgr_platform.h`.

Added

- Add MCXL20 porting layer and unit testing

v4.1.7

Fixed

- `mcmgr_stop_core_internal()` function now returns `kStatus_MCMGR_NotImplemented` status code instead of `kStatus_MCMGR_Success` when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

[v4.1.6]**Added**

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New `MCMGR_ProcessDeferredRxIsr()` API added.

[v4.1.5]**Added**

- Add notification into `MCMGR_EarlyInit` and `mcmgr_early_init_internal` functions to avoid using uninitialized data in their implementations.

[v4.1.4]**Fixed**

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

[v4.1.3]**Added**

- Add porting layers for imxrt1180.

Fixed

- `mu_isr()` updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- `mcmgr_mu_internal.c` code adaptation to new supported SoCs.

[v4.1.2]**Fixed**

- Update `mcmgr_stop_core_internal()` implementations to set core state to `kMCMGR_ResetCoreState`.

[v4.1.0]

Fixed

- Code adjustments to address MISRA C-2012 Rules

[v4.0.3]

Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

[v4.0.2]

Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

[v4.0.1]

Fixed

- Code formatting, removed unused code

[v4.0.0]

Added

- Add new MCMGR_TriggerEventForce() API.

[v3.0.0]

Removed

- Removed MCMGR_LoadApp(), MCMGR_MapAddress() and MCMGR_SignalReady()

Modified

- Modified MCMGR_GetStartupData()

Added

- Added MCMGR_EarlyInit(), MCMGR_RegisterEvent() and MCMGR_TriggerEvent()
- Added the ability for remote core monitoring and event handling

[v2.0.1]

Fixed

- Updated to be Misra compliant.

[v2.0.0]**Added**

- Support for lpcxpresso54114 board.

[v1.1.0]**Fixed**

- Ported to KSDK 2.0.0.

[v1.0.0]**Added**

- Initial release.

eRPC**MCUXpresso SDK : mcuxsdk-middleware-erpc**

Overview This repository is for MCUXpresso SDK eRPC middleware delivery and it contains eRPC component 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](#) for the complete delivery of MCUXpresso SDK to be able to build and run eRPC examples that are based on mcux-sdk-middleware-erpc component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [eRPC - 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 eRPC project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

eRPC

- [*MCUXpresso SDK : mcuxsdk-middleware-erpc*](#)

- [*Overview*](#)
- [*Documentation*](#)
- [*Setup*](#)
- [*Contribution*](#)

- [*eRPC*](#)

- [*About*](#)
- [*Releases*](#)
 - * [*Edge releases*](#)
- [*Documentation*](#)
- [*Examples*](#)
- [*References*](#)
- [*Directories*](#)
- [*Building and installing*](#)
 - * [*Requirements*](#)
 - [*Windows*](#)
 - [*Mac OS X*](#)
 - * [*Building*](#)
 - [*CMake and KConfig*](#)
 - [*Make*](#)
 - * [*Installing for Python*](#)
- [*Known issues and limitations*](#)
- [*Code providing*](#)

About

eRPC (Embedded RPC) is an open source Remote Procedure Call (RPC) system for multichip embedded systems and heterogeneous multicore SoCs.

Unlike other modern RPC systems, such as the excellent [Apache Thrift](#), eRPC distinguishes itself by being designed for tightly coupled systems, using plain C for remote functions, and having a small code size (<5kB). It is not intended for high performance distributed systems over a network.

eRPC does not force upon you any particular API style. It allows you to export existing C functions, without having to change their prototypes. (There are limits, of course.) And although the

internal infrastructure is written in C++, most users will be able to use only the simple C setup APIs shown in the examples below.

A code generator tool called `erpcgen` is included. It accepts input IDL files, having an `.erpc` extension, that have definitions of your data types and remote interfaces, and generates the shim code that handles serialization and invocation. `erpcgen` can generate either C/C++ or Python code.

Example `.erpc` file:

```
// Define a data type.
enum LEDName { kRed, kGreen, kBlue }

// An interface is a logical grouping of functions.
interface IO {
    // Simple function declaration with an empty reply.
    set_led(LEDName whichLed, bool onOrOff) -> void
}
```

Client side usage:

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* init eRPC client IO service */
    initIO_client(client_manager);

    // Now we can call the remote function to turn on the green LED.
    set_led(kGreen, true);

    /* deinit objects */
    deinitIO_client();
    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}
```

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* scope for client service */
    {
        /* init eRPC client IO service */
        IO_client client(client_manager);

        // Now we can call the remote function to turn on the green LED.
        client.set_led(kGreen, true);
    }

    /* deinit objects */
```

(continues on next page)

(continued from previous page)

```

    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

Server side usage:

```

// Implement the remote function.
void set_led(LEDName whichLed, bool onOrOff) {
    // implementation goes here
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    erpc_service_t service = create_IO_service();

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, service);

    // Run the server.
    erpc_server_run();

    /* deinit objects */
    destroy_IO_service(service);
    erpc_server_deinit(server);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

```

// Implement the remote function.
class IO : public IO_interface
{
    /* eRPC call definition */
    void set_led(LEDName whichLed, bool onOrOff) override {
        // implementation goes here
    }
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    IO IOImpl;
    IO_service io(&IOImpl);

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, &io);

    /* poll for requests */
}

```

(continues on next page)

(continued from previous page)

```

erpc_status_t err = server.run();

/* deinit objects */
erpc_server_deinit(server);
erpc_mbf_dynamic_deinit(message_buffer_factory);
erpc_transport_tcp_deinit(transport);
}

```

A number of transports are supported, and new transport classes are easy to write.

Supported transports can be found in *erpc/erpc_c/transport* folder. E.g:

- CMSIS UART
- NXP Kinetis SPI and DSPI
- POSIX and Windows serial port
- TCP/IP (mostly for testing)
- [NXP RPMsg-Lite / RPMsg TTY](#)
- SPIdev Linux
- USB CDC
- NXP Messaging Unit

eRPC is available with an unrestrictive BSD 3-clause license. See the [LICENSE file](#) for the full license text.

Releases [eRPC releases](#)

Edge releases Edge releases can be found on [eRPC CircleCI](#) webpage. Choose build of interest, then platform target and choose ARTIFACTS tab. Here you can find binary application from chosen build.

Documentation [Documentation](#) is in the wiki section.

[eRPC Infrastructure documentation](#)

Examples *Example IDL* is available in the *examples/* folder.

Plenty of eRPC multicore and multiprocessor examples can be also found in NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages.

To get the board list with multicore support (eRPC included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for eRPC multicore examples (RPMsg_Lite or Messaging Unit transports used) or

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples for eRPC multiprocessor examples (UART or SPI transports used).

eRPC examples use the 'erpc_' name prefix.

Another way of getting NXP MCUXpressoSDK eRPC multicore and multiprocessor examples is using the [mcux-sdk](#) Github repo. Follow the description how to use the West tool to clone and

update the mcuxsdk repo in [readme Overview section](#). Once done the armgcc eRPC examples can be found in

mcuxsdk/examples/<board_name>/multicore_examples or in

mcuxsdk/examples/<board_name>/multiprocessor_examples folders.

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the eRPC examples use the 'erpc_' name prefix.

References This section provides links to interesting erpc-based projects, articles, blogs or guides:

- [erpc \(EmbeddedRPC\) getting started notes](#)
- [ERPC Linux Local Environment Construction and Use](#)
- [The New Wio Terminal eRPC Firmware](#)

Directories *doc* - Documentation.

doxygen - Configuration and support files for running Doxygen over the eRPC C++ infrastructure and erpcgen code.

erpc_c - Holds C/C++ infrastructure for eRPC. This is the code you will include in your application.

erpc_python - Holds Python version of the eRPC infrastructure.

erpcgen - Holds source code for erpcgen and makefiles or project files to build erpcgen on Windows, Linux, and OS X.

erpcsniffer - Holds source code for erpcsniffer application.

examples - Several example IDL files.

mk - Contains common makefiles for building eRPC components.

test - Client/server tests. These tests verify the entire communications path from client to server and back.

utilities - Holds utilities which bring additional benefit to eRPC apps developers.

Building and installing These build instructions apply to host PCs and embedded Linux. For bare metal or RTOS embedded environments, you should copy the *erpc_c* directory into your application sources.

CMake and KConfig build:

It builds a static library of the eRPC C/C++ infrastructure, the *erpcgen* executable, and optionally the unit tests and examples.

CMake is compatible with gcc and clang. On Windows local MingGW downloaded by *script* can be used.

Make build:

It builds a static library of the eRPC C/C++ infrastructure, the *erpcgen* executable, and optionally the unit tests.

The makefiles are compatible with gcc or clang on Linux, OS X, and Cygwin. A Windows build of *erpcgen* using Visual Studio is also available in the *erpcgen/VisualStudio_v14* directory. There is also an Xcode project file in the *erpcgen* directory, which can be used to build *erpcgen* for OS X.

Requirements eRPC now support building **erpcgen**, **erpc_lib**, **tests** and **C examples** using CMake.

Requirements when using CMake:

- **CMake** (minimal version 3.20.0)
- Generator - **Make**, **Ninja**, ...
- **C/C++ compiler** - **GCC**, **CLANG**, ...
- **Bison** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

Requirements when using Make:

- **Make**
- **C/C++ compiler** - **GCC**, **CLANG**, ...
- **Bison** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

Windows Related steps to build **erpcgen** using **Visual Studio** are described in `erpcgen/VisualStudio_v14/readme_erpcgen.txt`.

To install MinGW, Bison, Flex locally on Windows:

```
./install_dependencies.ps1
* ```

#### Linux

```bash
./install_dependencies.sh
```

Mandatory for case, when build for different architecture is needed

- `gcc-multilib`, `g++-multilib`

## Mac OS X

```
./install_dependencies.sh
```

## Building

**CMake and KConfig** eRPC use CMake and KConfig to configurate and build eRPC related targets. KConfig can be edited by *prj.conf* or *menuconfig* when building.

Generate project, config and build. In *erpc/* execute:

```
cmake -B ./build # in erpc/build generate cmake project
cmake --build ./build --target menuconfig # Build menuconfig and configurate erpcgen, erpc_lib, tests and ↵
↵examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

**\*\*CMake will use the system's default compilers and generator**

If you want to use Windows and locally installed MinGW, use *CMake preset* :

```
cmake --preset mingw64 # Generate project in ./build using mingw64's make and compilers
cmake --build ./build --target menuconfig # Build menuconfig and configurate erpcgen, erpc_lib, tests and ↵
↵examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

**Make** To build the library and erpcgen, run from the repo root directory:

```
make
```

To install the library, erpcgen, and include files, run:

```
make install
```

You may need to sudo the make install.

By default this will install into /usr/local. If you want to install elsewhere, set the PREFIX environment variable. Example for installing into /opt:

```
make install PREFIX=/opt
```

List of top level Makefile targets:

- erpc: build the liberpc.a static library
- erpcgen: build the erpcgen tool
- erpcsniffer: build the sniffer tool
- test: build the unit tests under the *test* directory
- all: build all of the above
- install: install liberpc.a, erpcgen, and include files

eRPC code is validated with respect to the C++ 11 standard.

**Installing for Python** To install the Python infrastructure for eRPC see instructions in the *erpc python readme*.

### Known issues and limitations

- Static allocations controlled by the ERPC\_ALLOCATION\_POLICY config macro are not fully supported yet, i.e. not all erpc objects can be allocated statically now. It deals with the ongoing process and the full static allocations support will be added in the future.

**Code providing** Repository on Github contains two main branches: **main** and **develop**. Code is developed on **develop** branch. Release version is created via merging **develop** branch into **main** branch.

---

Copyright 2014-2016 Freescale Semiconductor, Inc.

Copyright 2016-2025 NXP

### eRPC Getting Started

**Overview** This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc` folder.

**Create an eRPC application** This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
  1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
  2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
  3. Add infrastructure files.
  4. Add user code for client and server applications.
  5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

**Multicore server application** The “Matrix multiply” eRPC server project is located in the following folder:

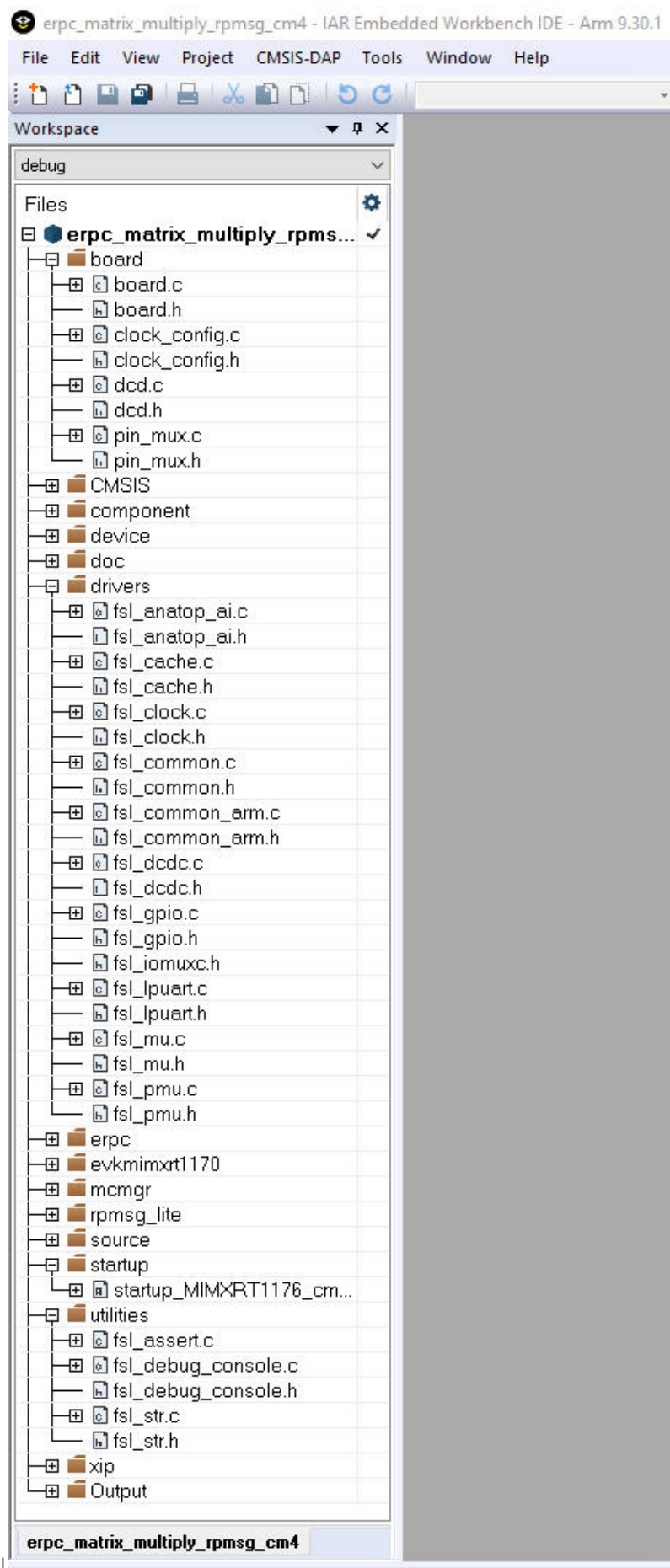
`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpm4/iar`

The project files for the eRPC server have the `_cm4` suffix.

**Server project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`





|

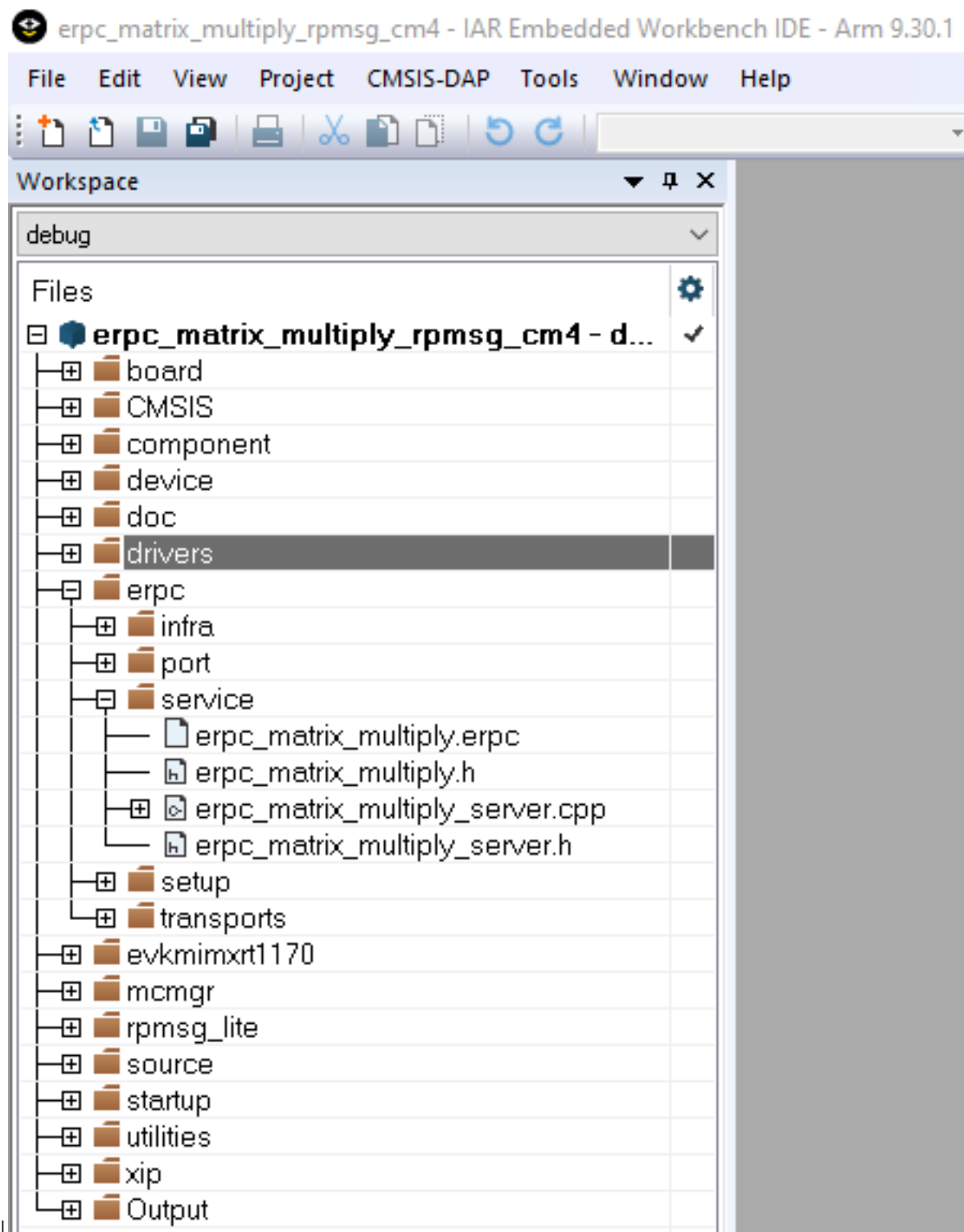
**Parent topic:**Multicore server application

**Server related generated files** The server-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_server.h
- erpc\_\_matrix\_\_multiply\_\_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_common/erpc\_matrix\_multiply/



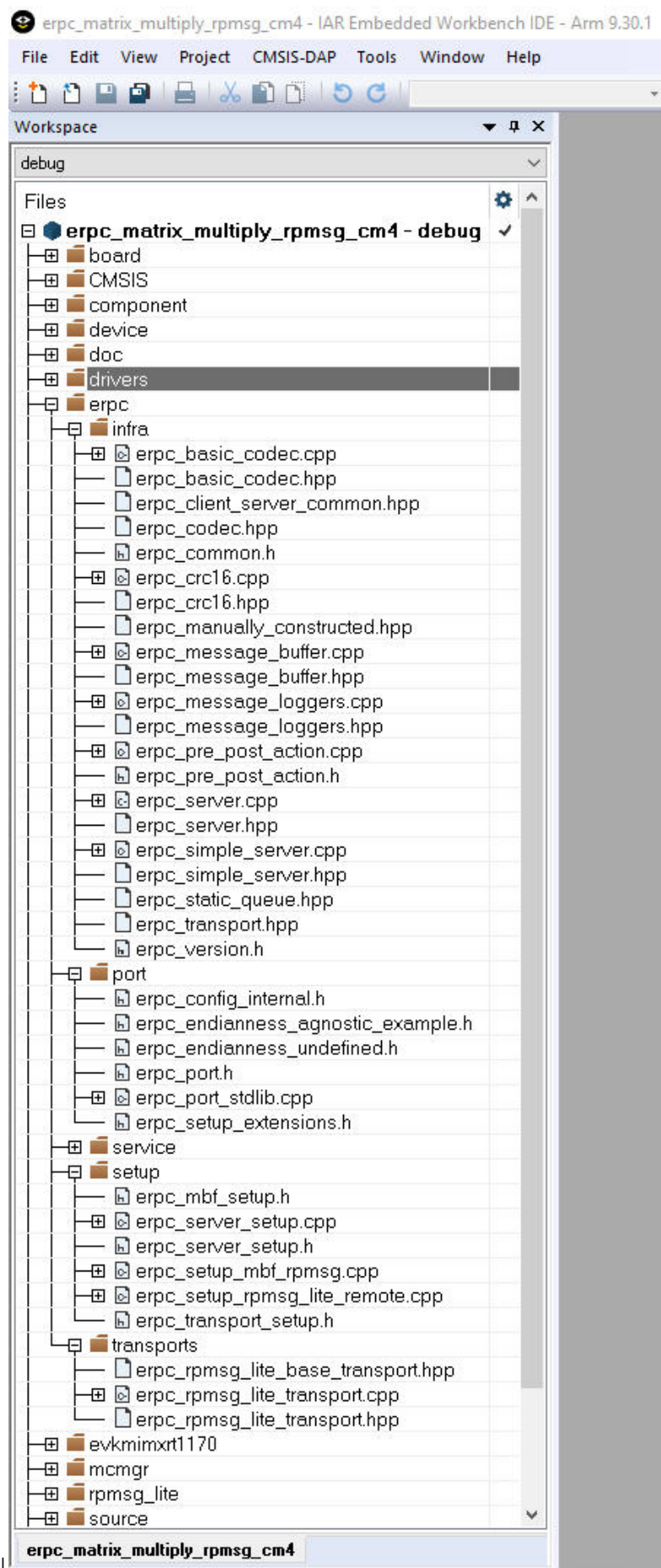
**Parent topic:**Multicore server application

**Server infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
  - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
  - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
  - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
  - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
  - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.



|

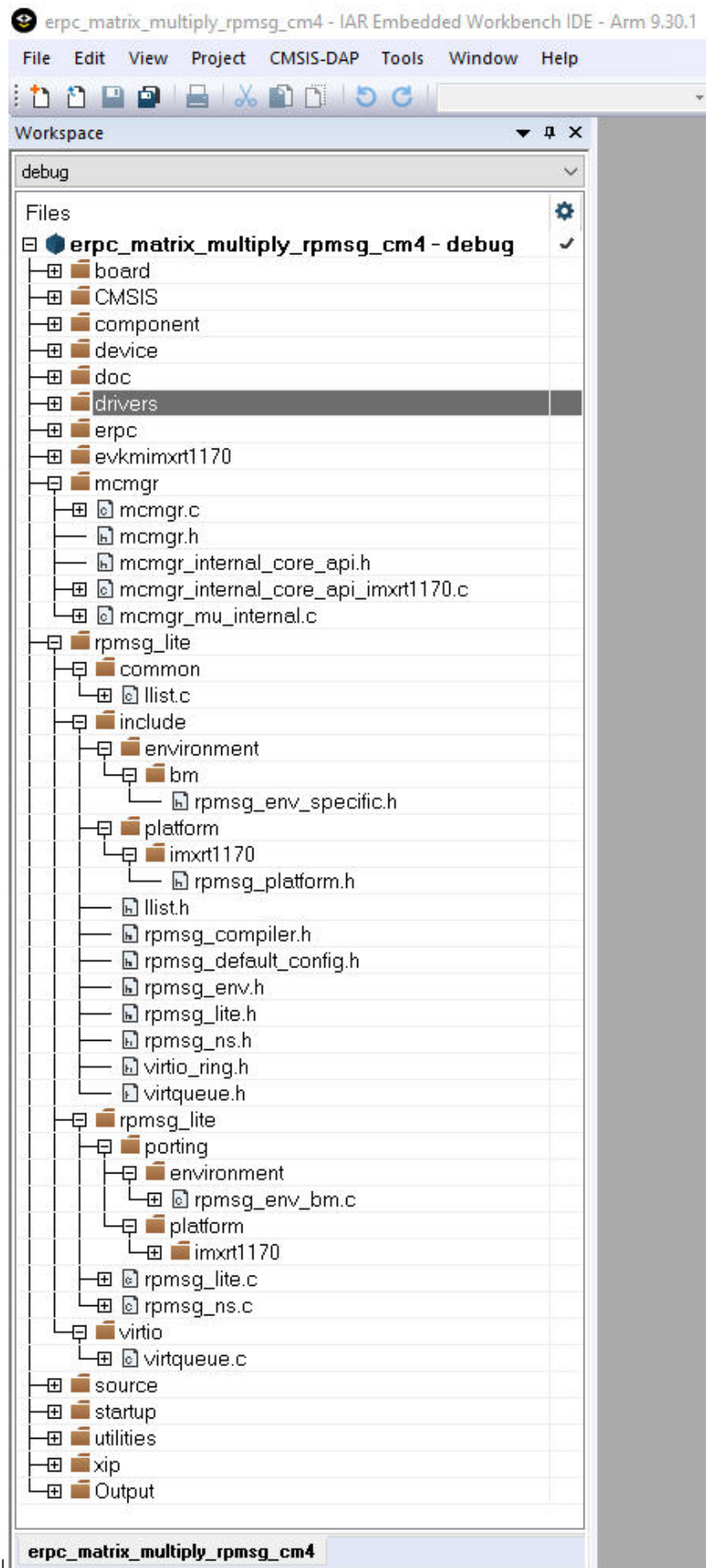
**Parent topic:** Multicore server application

**Server multicore infrastructure files** Because of the RPSMsg-Lite (transport layer), it is also necessary to include RPSMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpsmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



Parent topic: Multicore server application

**Server user code** The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

The `main_core1.c` file contains two functions:

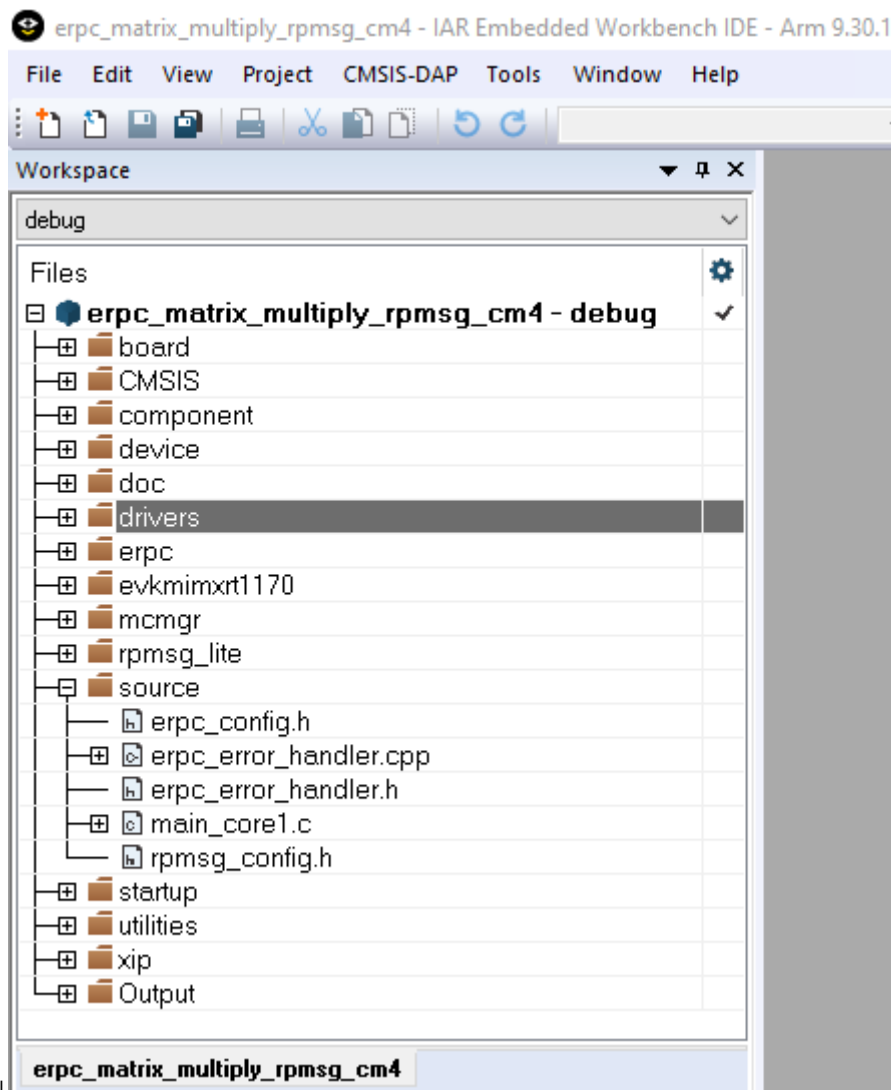
- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
 ...
}
int main()
{
 ...
 /* RPSMsg-Lite transport layer initialization */
 erpc_transport_t transport;
 transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
 ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_rpmsg_init(transport);
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server);
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}
```



Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg` folder.



**Parent topic:**Multicore server application

**Parent topic:**[Create an eRPC application](#)

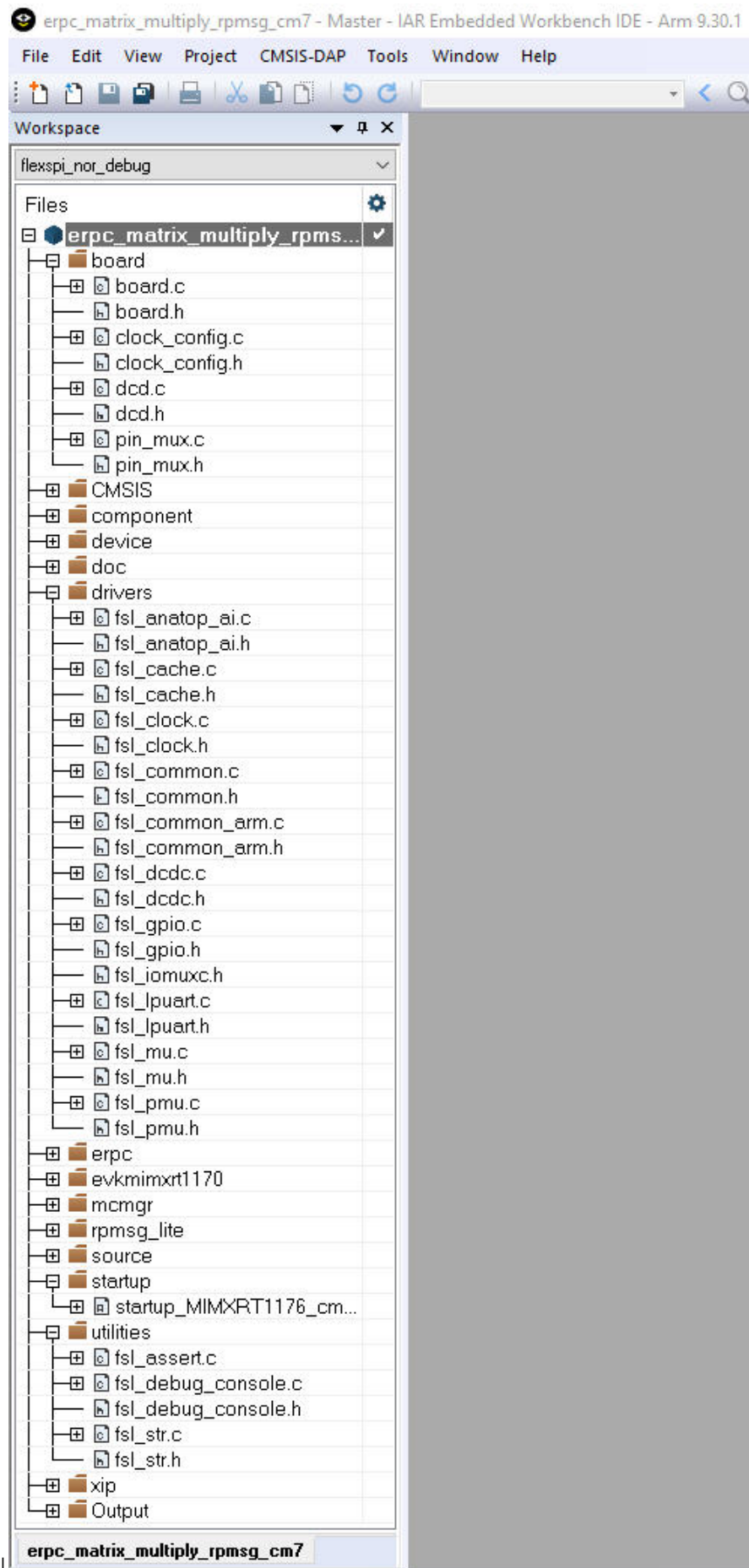
**Multicore client application** The “Matrix multiply” eRPC client project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar`

Project files for the eRPC client have the `_cm7` suffix.

**Client project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



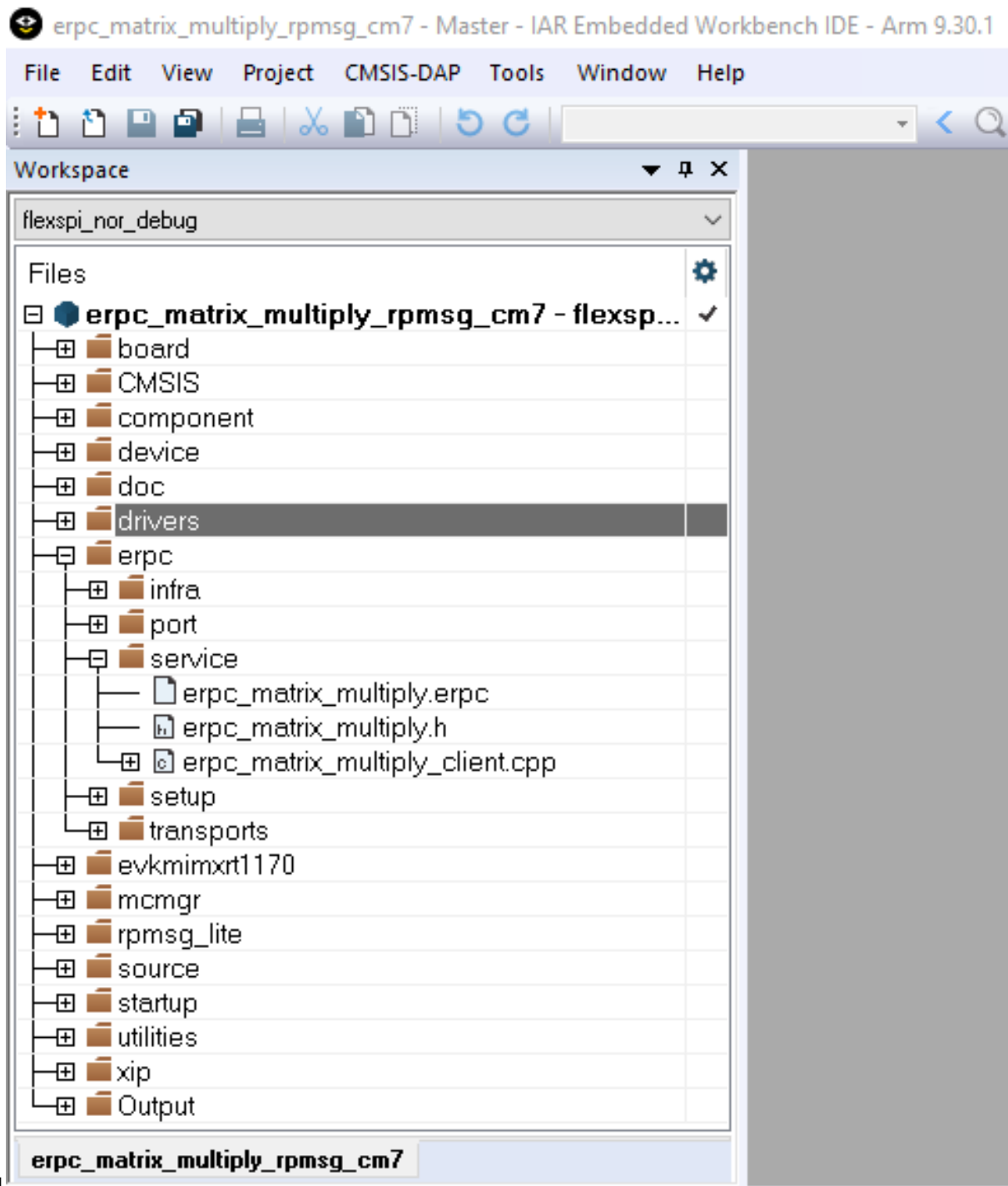
|

**Parent topic:**Multicore client application

**Client-related generated files** The client-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



**Parent topic:**Multicore client application

**Client infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

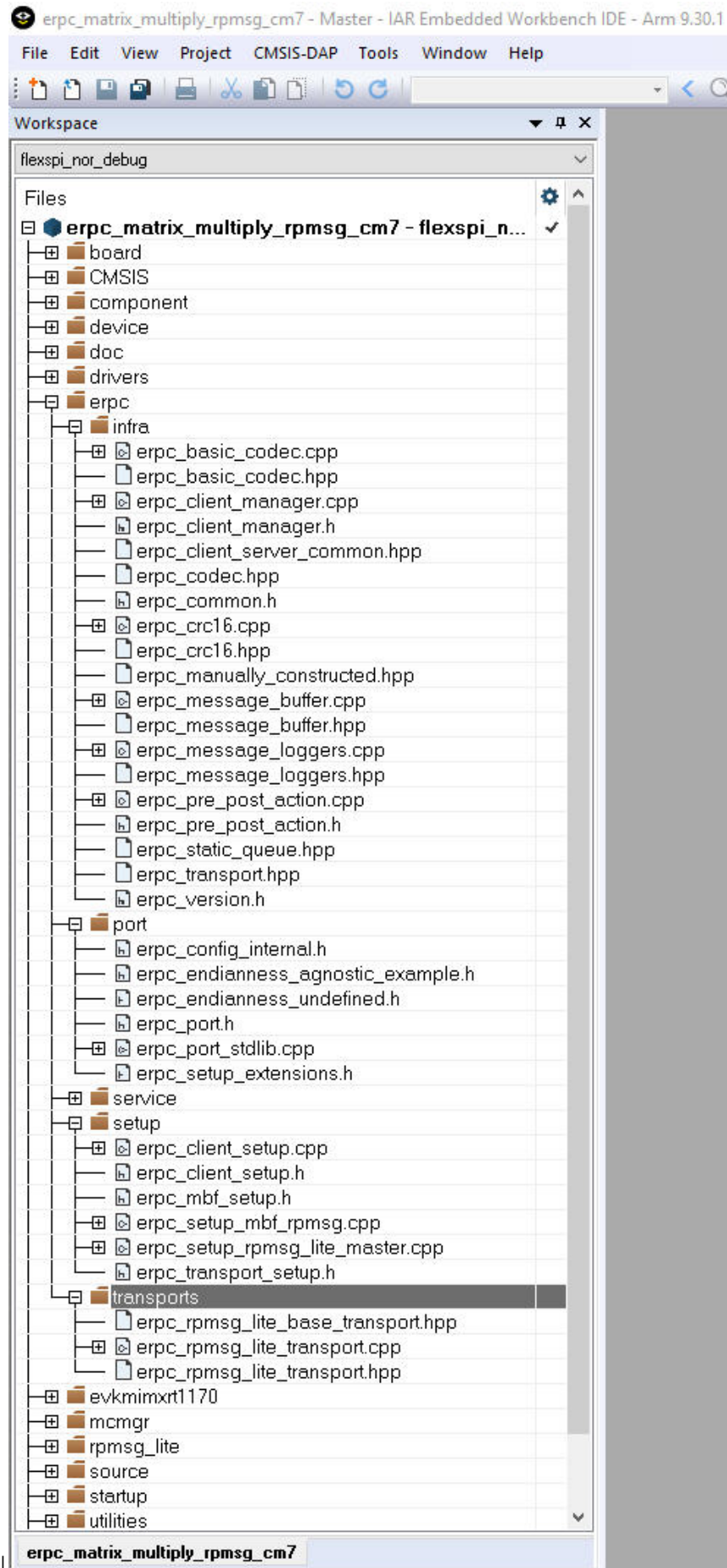
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files needs to be added into the client project.



|

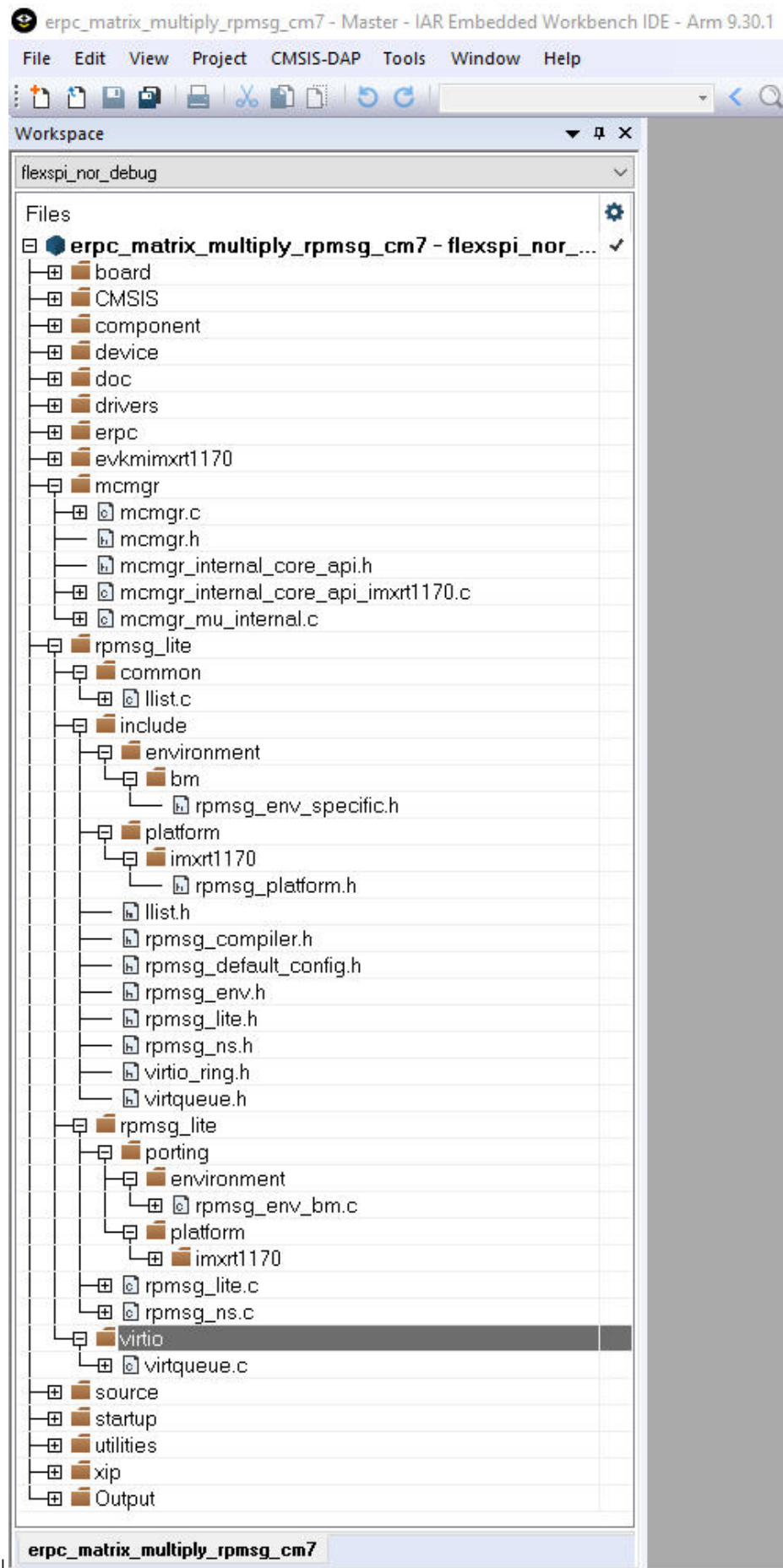
**Parent topic:** Multicore client application

**Client multicore infrastructure files** Because of the RPSMsg-Lite (transport layer), it is also necessary to include RPSMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpsmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`





|

**Parent topic:**Multicore client application

**Client user code** The client's user code is stored in the main\_core0.c file, located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_example/erpc\_matrix\_multiply\_rpmsg/cm7

The main\_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc\_error\_handler.h and erpc\_error\_handler.cpp files.

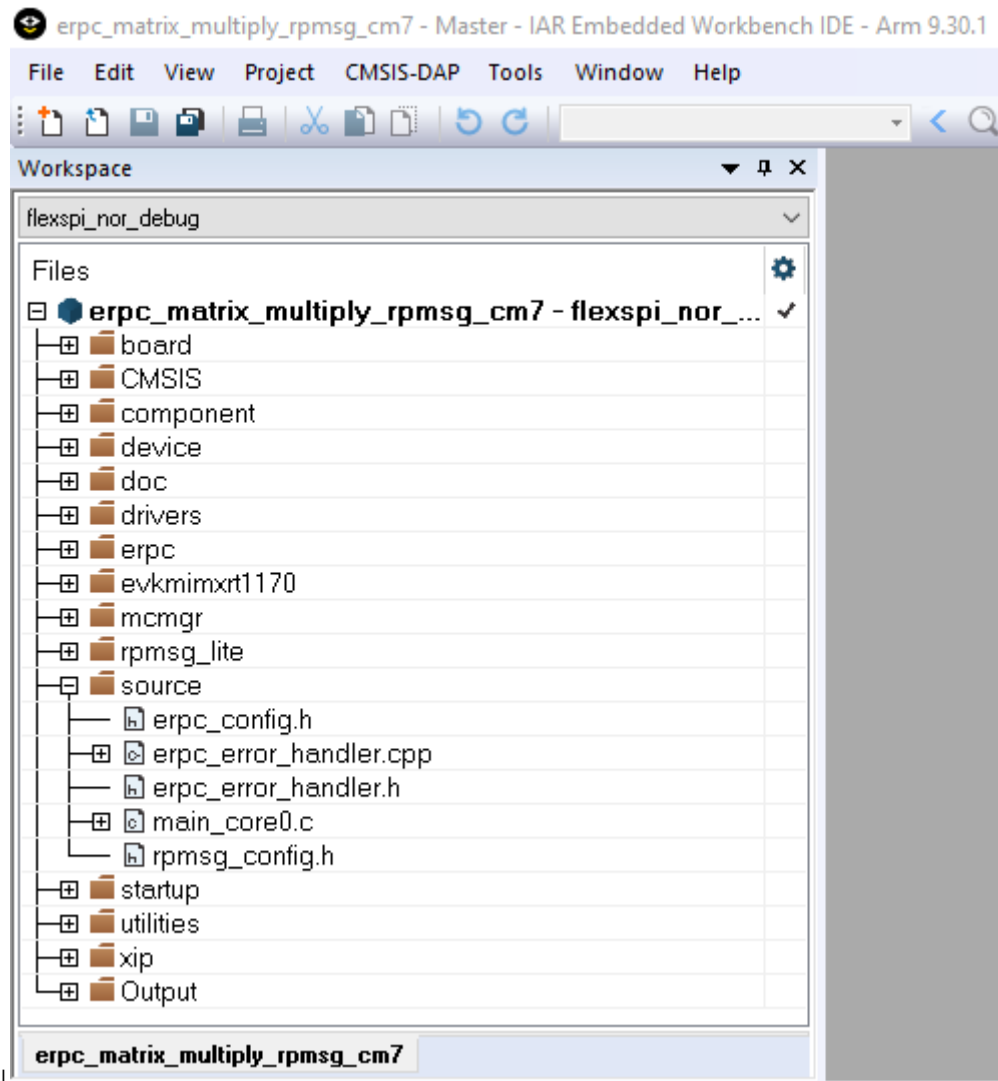
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
 /* Invoke the erpcMatrixMultiply function */
 erpcMatrixMultiply(matrix1, matrix2, result_matrix);
 ...
 /* Check if some error occurred in eRPC */
 if (g_erpc_error_occurred)
 {
 /* Exit program loop */
 break;
 }
 ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_matrix\_multiply\_rpmsg



Parent topic: Multicore client application

Parent topic: [Create an eRPC application](#)

**Multiprocessor server application** The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMs-Lite). The RPMs-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|      |                                                                         |
|------|-------------------------------------------------------------------------|
| SPI  | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp          |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp |
| UART | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp            |

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.hpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.cpp

|

**Server user code** The server's user code is stored in the main\_server.c file, located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_server\_matrix\_multiply\_<transport\_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
 ...
}
int main()
{
 ...
 /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver.
 ↪operations */
 erpc_transport_t transport;
 transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_dynamic_init();
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server)
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}
```

**Parent topic:**Multiprocessor server application

**Multiprocessor client application** The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_client\_matrix\_multiply\_<transport\_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RMPMsg-Lite). The RMPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|      |                                                                           |
|------|---------------------------------------------------------------------------|
| SPI  | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp           |
|      | <eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp |
|      | <eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp |
| UART | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp              |
|      | <eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp     |
|      | <eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp     |
|      |                                                                           |

**Client user code** The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

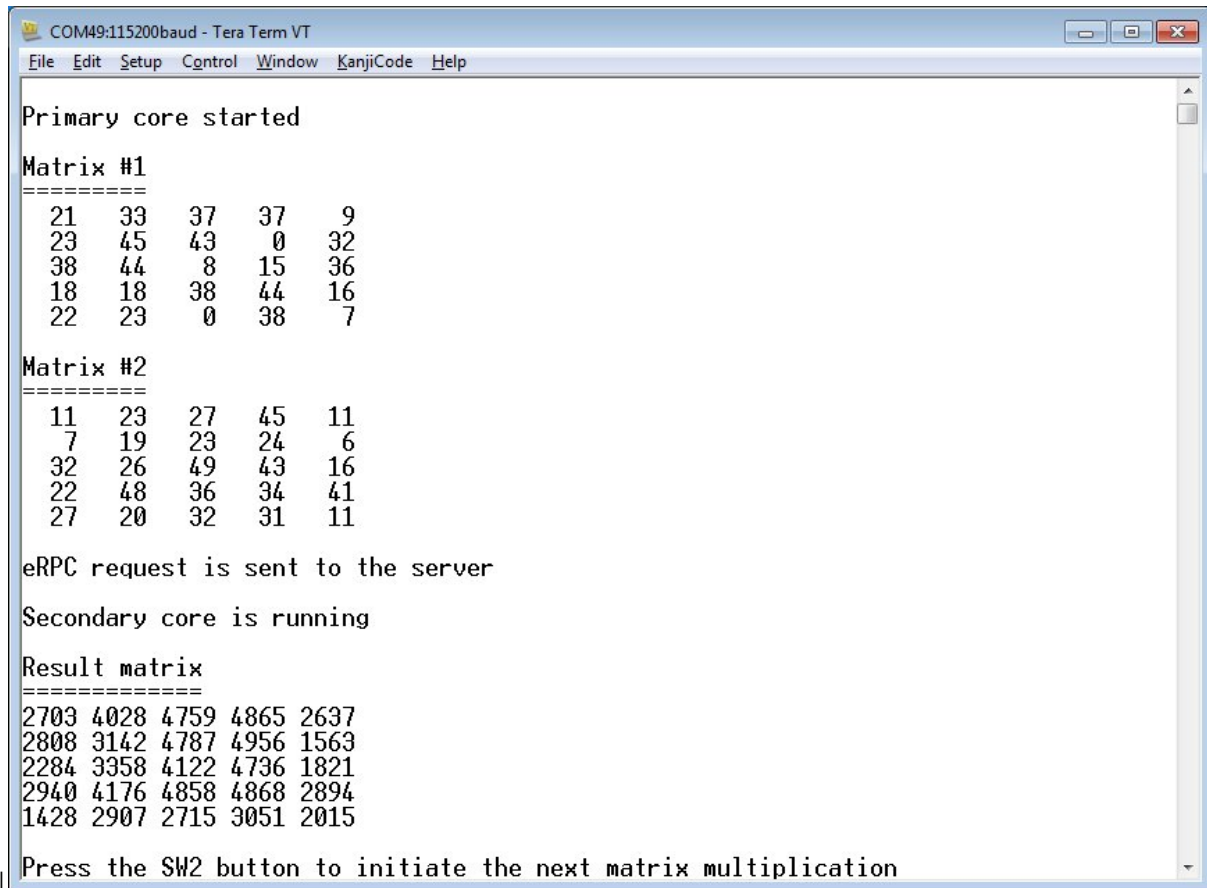
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
 /* Invoke the erpcMatrixMultiply function */
 erpcMatrixMultiply(matrix1, matrix2, result_matrix);
 ...
 /* Check if some error occurred in eRPC */
 if (g_erpc_error_occurred)
 {
 /* Exit program loop */
 break;
 }
 ...
}
```

**Parent topic:**Multiprocessor client application

**Parent topic:**Multiprocessor server application

Parent topic:[Create an eRPC application](#)

**Running the eRPC application** Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK\_install\_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



```

COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Primary core started

Matrix #1
=====
 21 33 37 37 9
 23 45 43 0 32
 38 44 8 15 36
 18 18 38 44 16
 22 23 0 38 7

Matrix #2
=====
 11 23 27 45 11
 7 19 23 24 6
 32 26 49 43 16
 22 48 36 34 41
 27 20 32 31 11

eRPC request is sent to the server

Secondary core is running

Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015

Press the SW2 button to initiate the next matrix multiplication

```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:[Create an eRPC application](#)

Parent topic:[eRPC example](#)

**eRPC example** This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RPMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The “Matrix multiply” application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC communicates with the eRPC server that runs on another SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

| Multicore application source and project files | `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore/`  
 | Multiprocessor application source and project files | `<MCUXpressoSDK_install_dir>/boards/<board_name>/multi`  
`<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr`  
 | | eRPC source files | `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/` | | RPLite  
 source files | `<MCUXpressoSDK_install_dir>/middleware/multicore/rplite/` |

**Designing the eRPC application** The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has been declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

**Parent topic:** [eRPC example](#)

**Creating the IDL file** The created IDL file is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/`

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc\_matrix\_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix\_size* follows next. The *matrix\_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.
- As shown above, the function’s declaration contains three parameters of *Matrix* type: *matrix1* and *matrix2* are input parameters, while *result\_matrix* is the output parameter. Additionally, the returned data type is declared as *void*.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

**Parent topic:** [eRPC example](#)

**Using the eRPC generator tool** | Windows OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86`  
| Linux OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64`  
`<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86`  
| | Mac OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac`

The files for the “Matrix multiply” example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (\*.erpc) is located; then run the following command:

```
erpcgen <IDL_file>.erpc
```

- In the “Matrix multiply” example, the command should look like:

```
erpcgen erpc_matrix_multiply.erpc
```

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- “-?”/”—help” – Shows supported commands.
- “-o <filePath>”/”—output<filePath>” – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen -o <path_to_output>
<path_to_IDL>/<IDL_file_name>.erpc
```

For the “Matrix multiply” example, when the command is executed from the default erpcgen location, it looks like:

```
erpcgen -o
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu
```

In both cases, the following four files are generated into the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service` folder:

- erpc\_matrix\_multiply.h
- erpc\_matrix\_multiply\_client.cpp
- erpc\_matrix\_multiply\_server.h
- erpc\_matrix\_multiply\_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/service` folder.

**For Linux OS users:**

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as `./erpcgen...` instead of as `erpcgen ....`



Parent topic: [eRPC example](#)

**Create an eRPC application** This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
  1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
  2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
  3. Add infrastructure files.
  4. Add user code for client and server applications.
  5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

**Multicore server application** The “Matrix multiply” eRPC server project is located in the following folder:

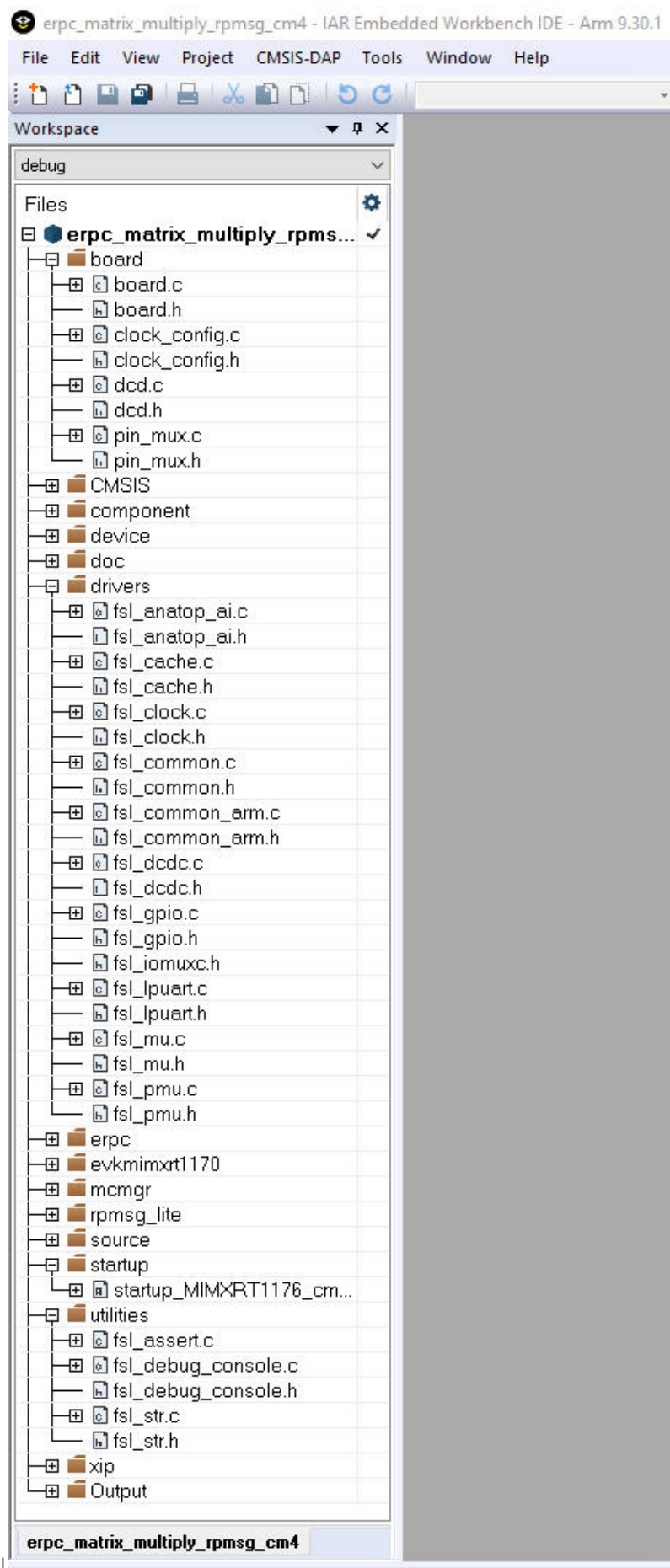
`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmmsg/cm4/iar/`

The project files for the eRPC server have the `_cm4` suffix.

**Server project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`





|

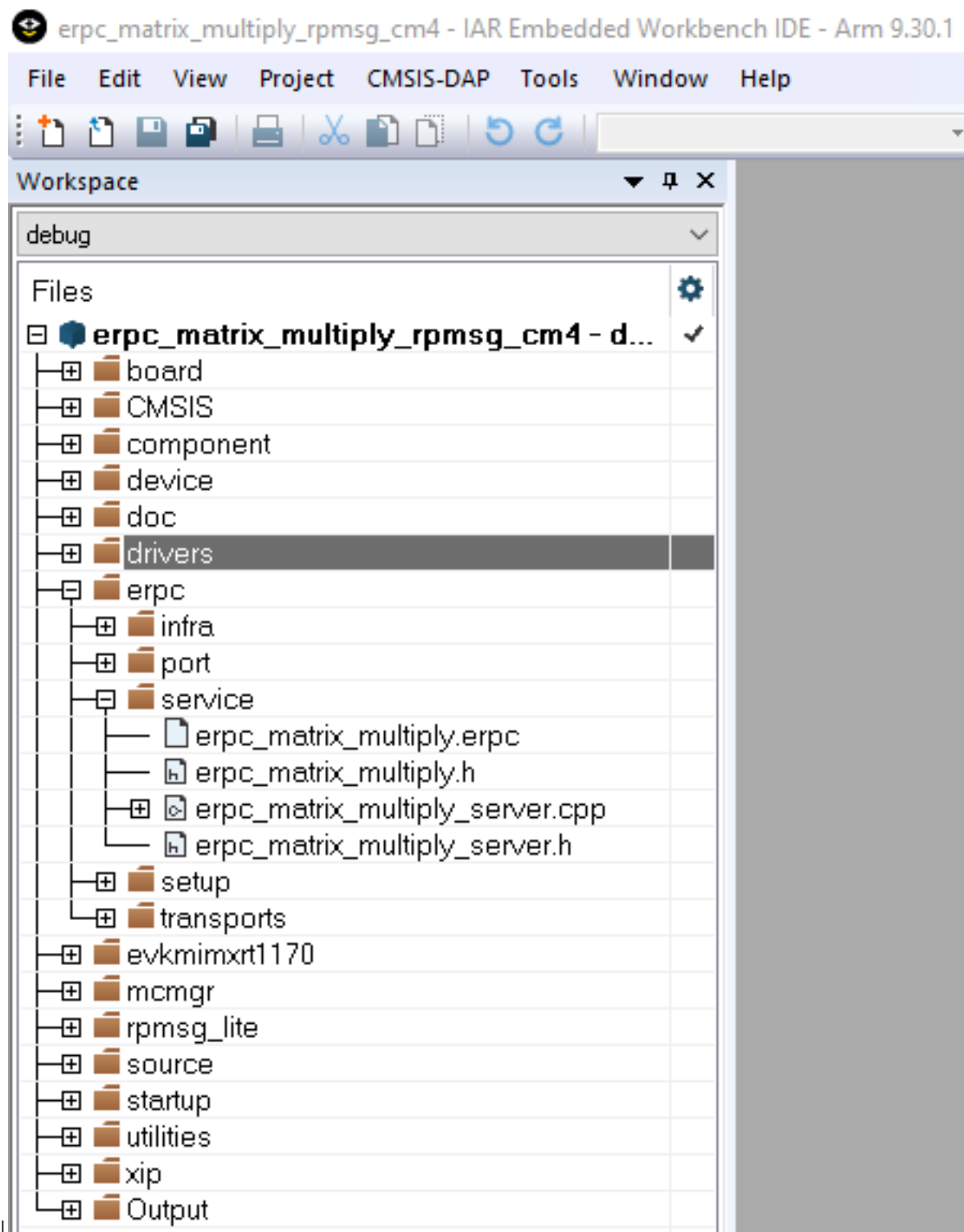
**Parent topic:**Multicore server application

**Server related generated files** The server-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_server.h
- erpc\_\_matrix\_\_multiply\_\_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_common/erpc\_matrix\_multiply/



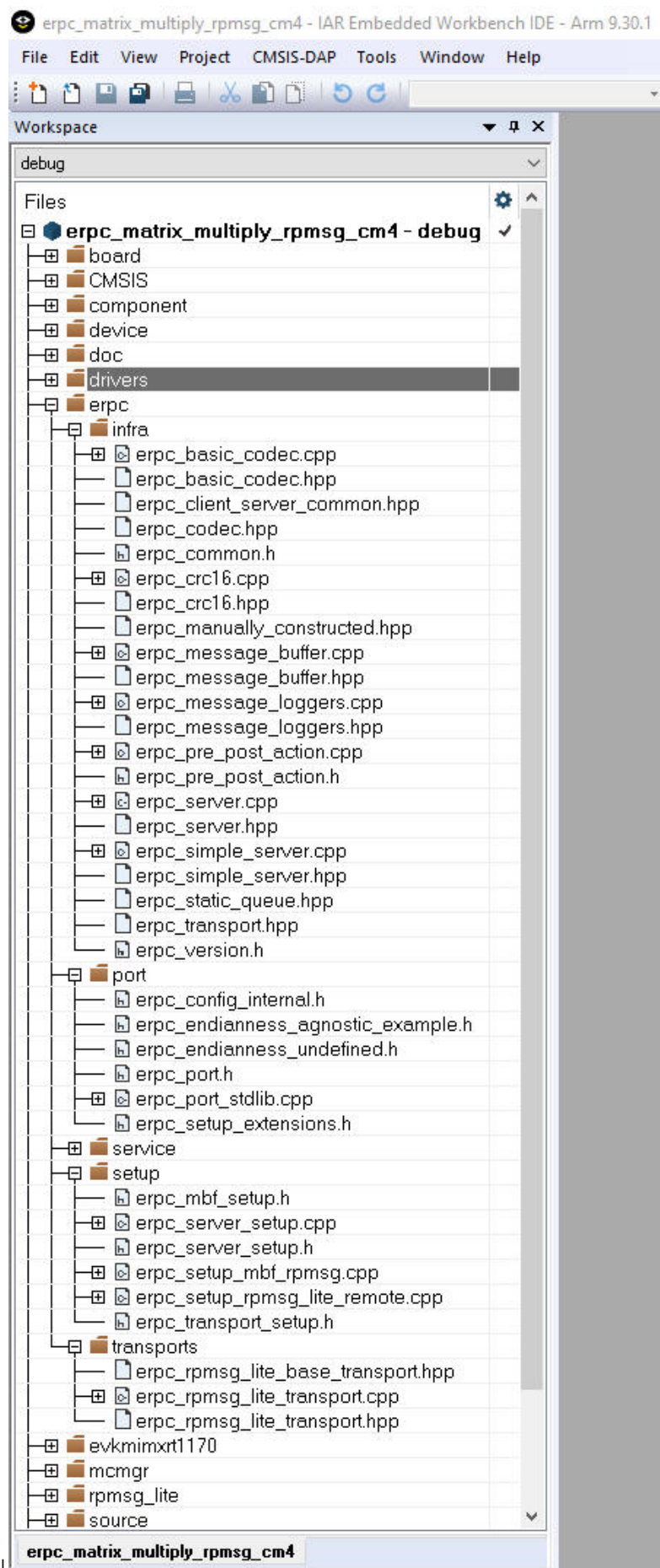
**Parent topic:** Multicore server application

**Server infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
  - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
  - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
  - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
  - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
  - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.



|

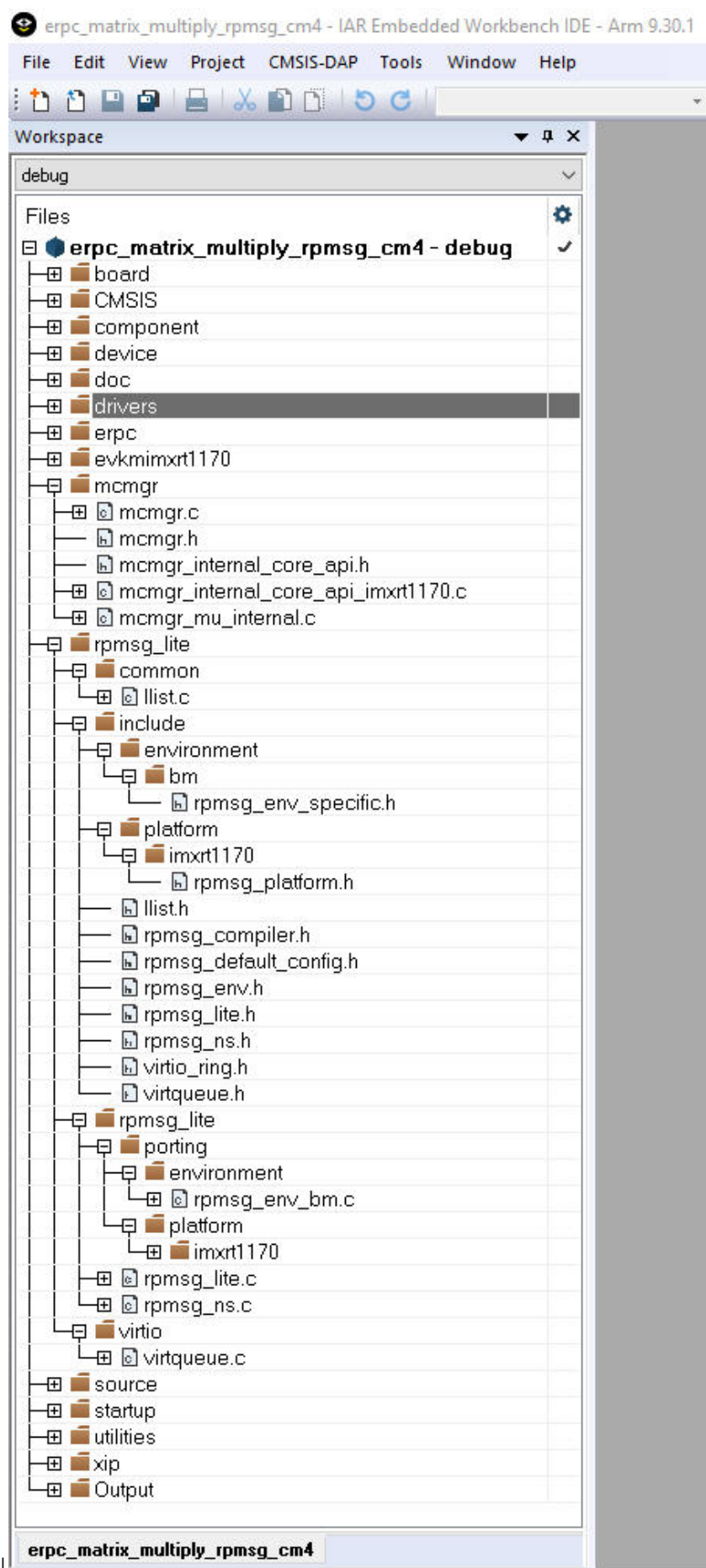
**Parent topic:** Multicore server application

**Server multicore infrastructure files** Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



**Parent topic:** Multicore server application

**Server user code** The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

The `main_core1.c` file contains two functions:

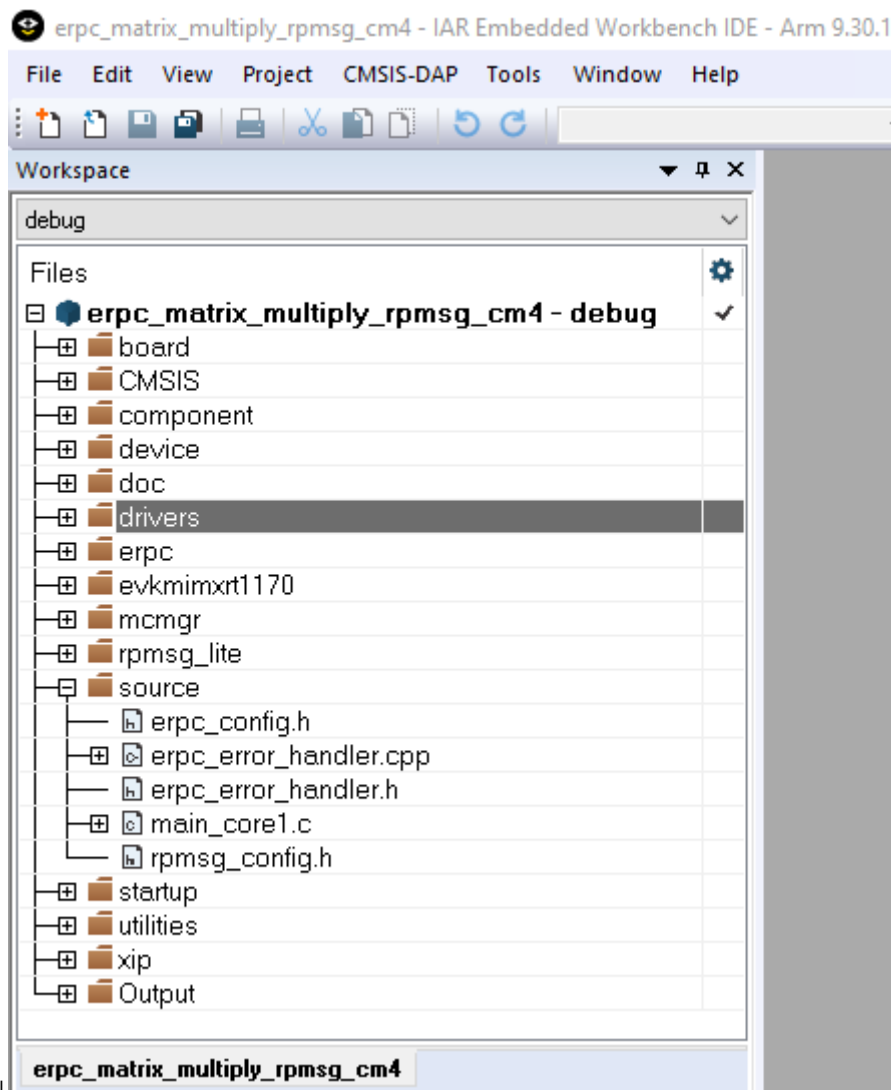
- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
 ...
}
int main()
{
 ...
 /* RPSMsg-Lite transport layer initialization */
 erpc_transport_t transport;
 transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
 ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_rpmsg_init(transport);
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server);
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}
```



Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg` folder.



**Parent topic:**Multicore server application

**Parent topic:**[Create an eRPC application](#)

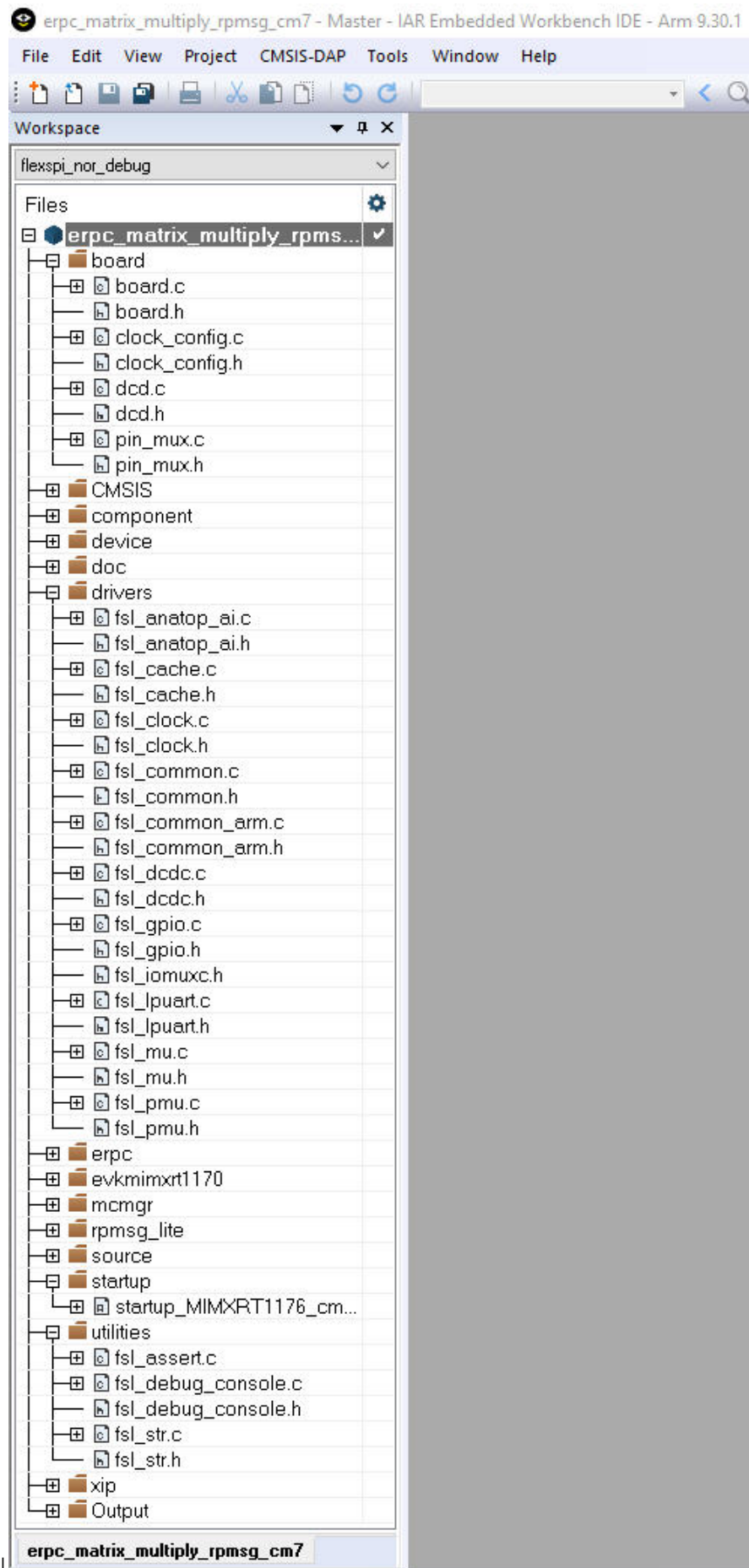
**Multicore client application** The “Matrix multiply” eRPC client project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar`

Project files for the eRPC client have the `_cm7` suffix.

**Client project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



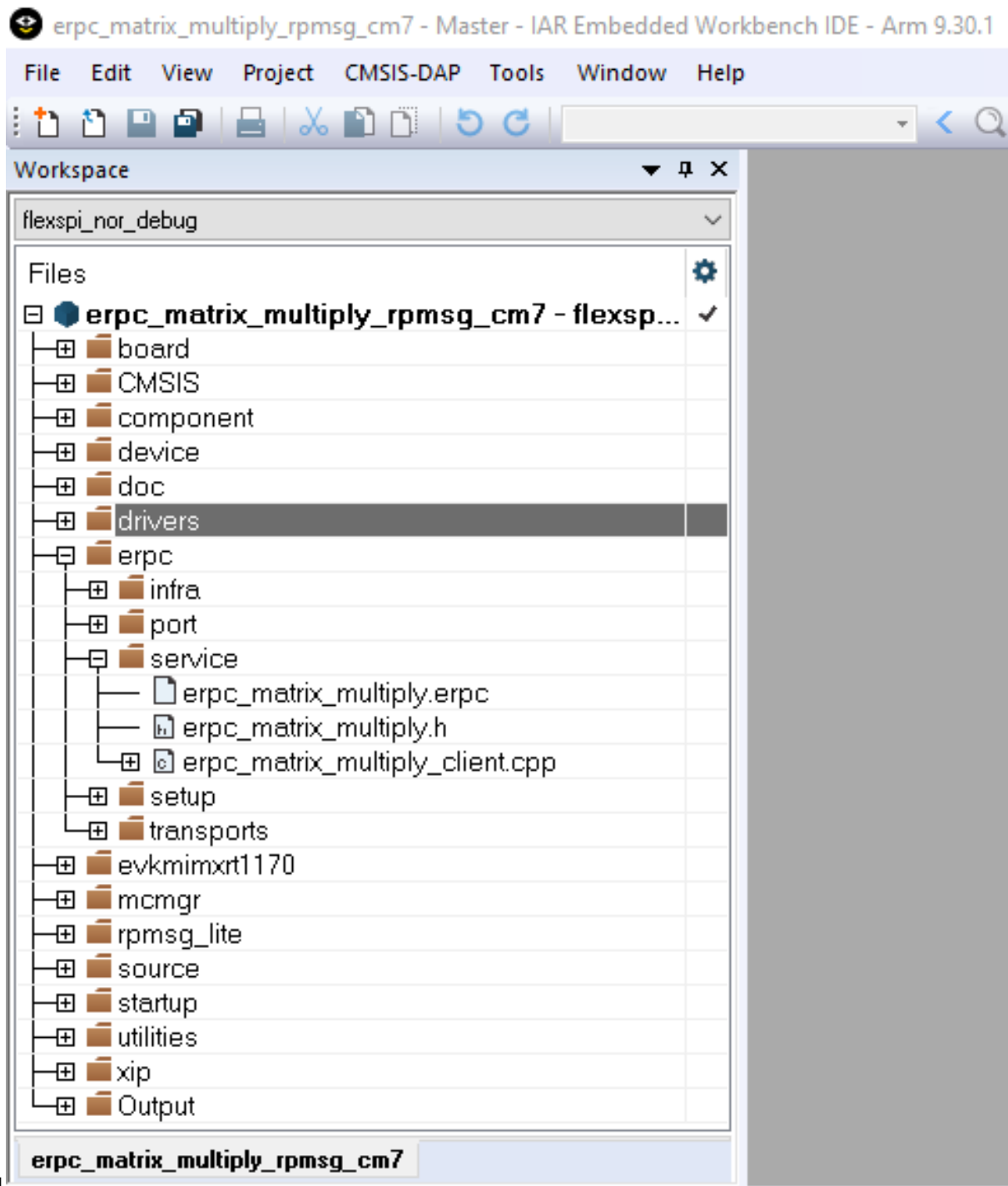
|

**Parent topic:**Multicore client application

**Client-related generated files** The client-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



**Parent topic:** Multicore client application

**Client infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

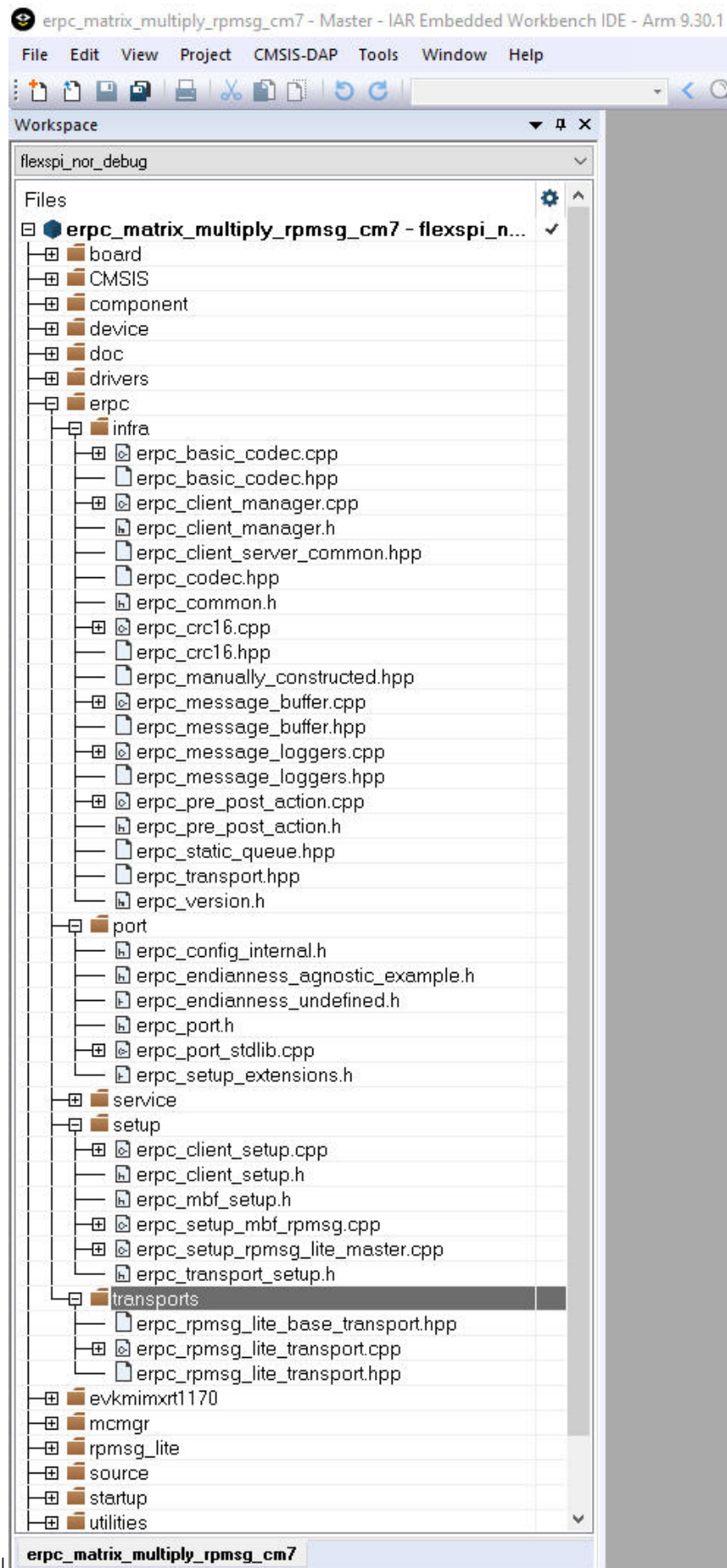
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files needs to be added into the client project.



|

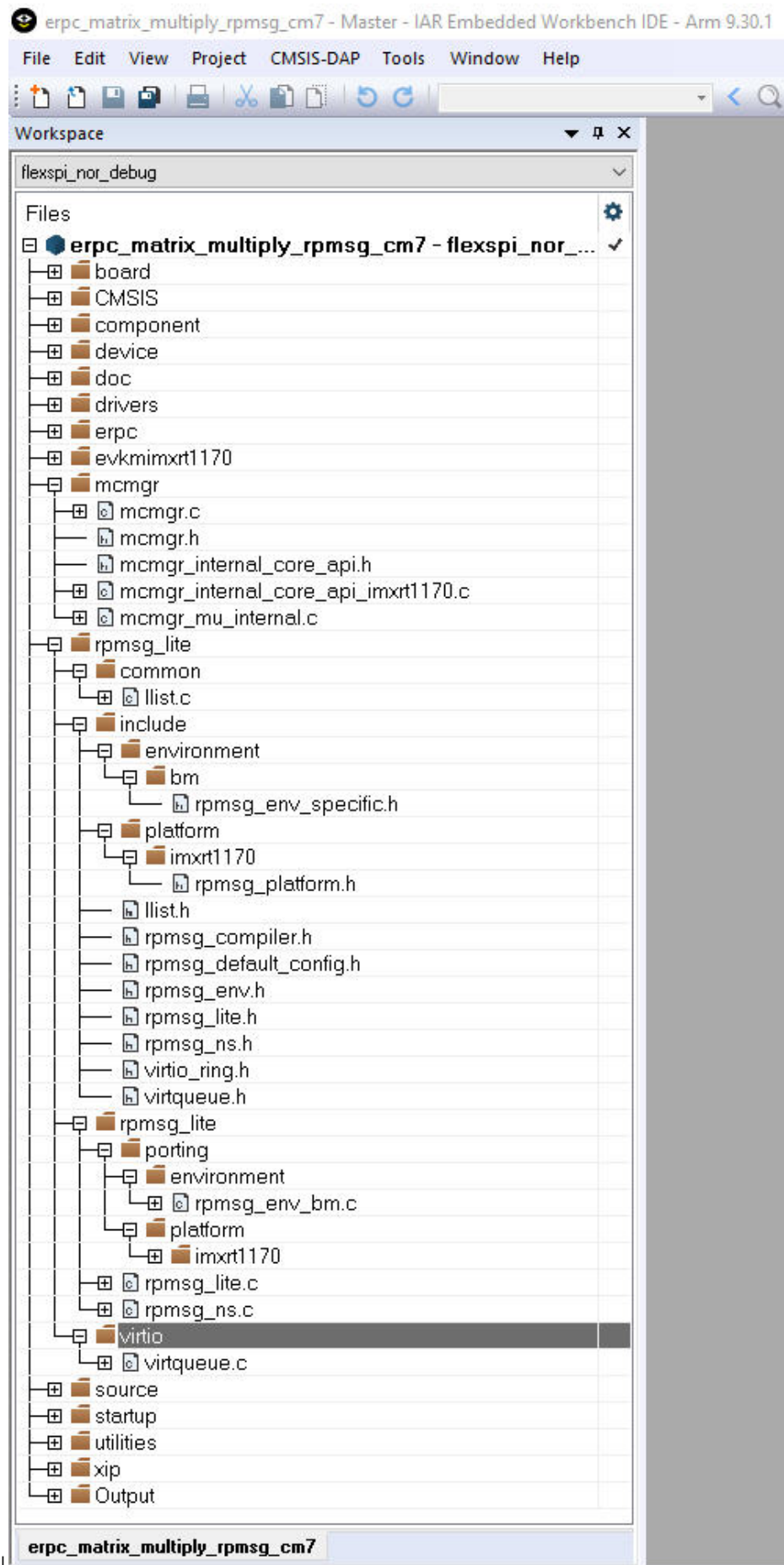
**Parent topic:**Multicore client application

**Client multicore infrastructure files** Because of the RPSMsg-Lite (transport layer), it is also necessary to include RPSMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpsmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`





|

**Parent topic:**Multicore client application

**Client user code** The client's user code is stored in the main\_core0.c file, located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_example/erpc\_matrix\_multiply\_rpmsg/cm7

The main\_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc\_error\_handler.h and erpc\_error\_handler.cpp files.

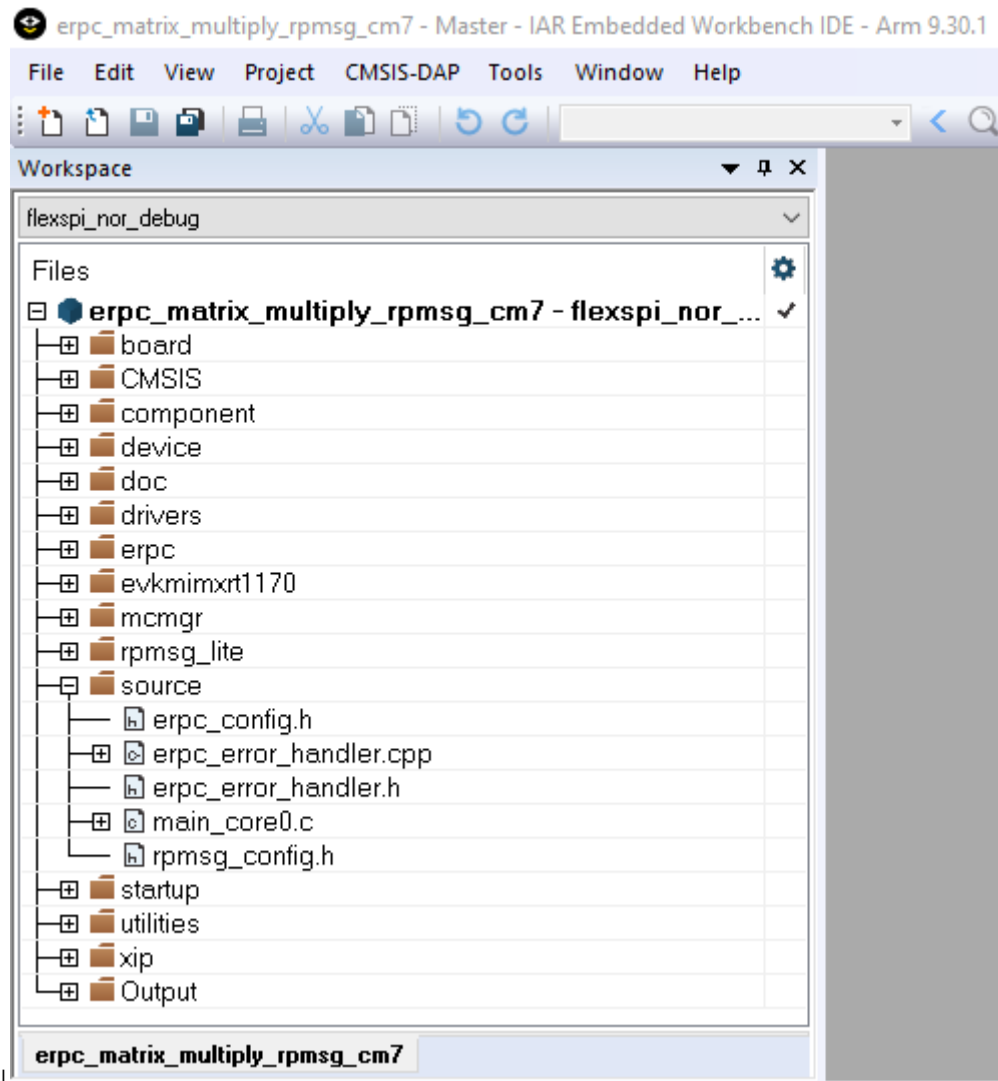
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPSMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
 /* Invoke the erpcMatrixMultiply function */
 erpcMatrixMultiply(matrix1, matrix2, result_matrix);
 ...
 /* Check if some error occurred in eRPC */
 if (g_erpc_error_occurred)
 {
 /* Exit program loop */
 break;
 }
 ...
}
```

Except for the application main file, there are configuration files for the RPSMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_matrix\_multiply\_rpmsg



Parent topic: Multicore client application

Parent topic: [Create an eRPC application](#)

**Multiprocessor server application** The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMMsg-Lite). The RPMMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|      |                                                                         |
|------|-------------------------------------------------------------------------|
| SPI  | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp          |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp |
| UART | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp            |

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.hpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.cpp

|

**Server user code** The server's user code is stored in the main\_server.c file, located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_server\_matrix\_multiply\_<transport\_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
 ...
}
int main()
{
 ...
 /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver.
 ↪operations */
 erpc_transport_t transport;
 transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_dynamic_init();
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server)
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}
```

**Parent topic:**Multiprocessor server application

**Multiprocessor client application** The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_client\_matrix\_multiply\_<transport\_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPSMsg-Lite). The RPSMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|      |                                                                           |
|------|---------------------------------------------------------------------------|
| SPI  | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp           |
|      | <eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp |
|      | <eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp |
| UART | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp              |
|      | <eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp     |
|      | <eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp     |
|      |                                                                           |

**Client user code** The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

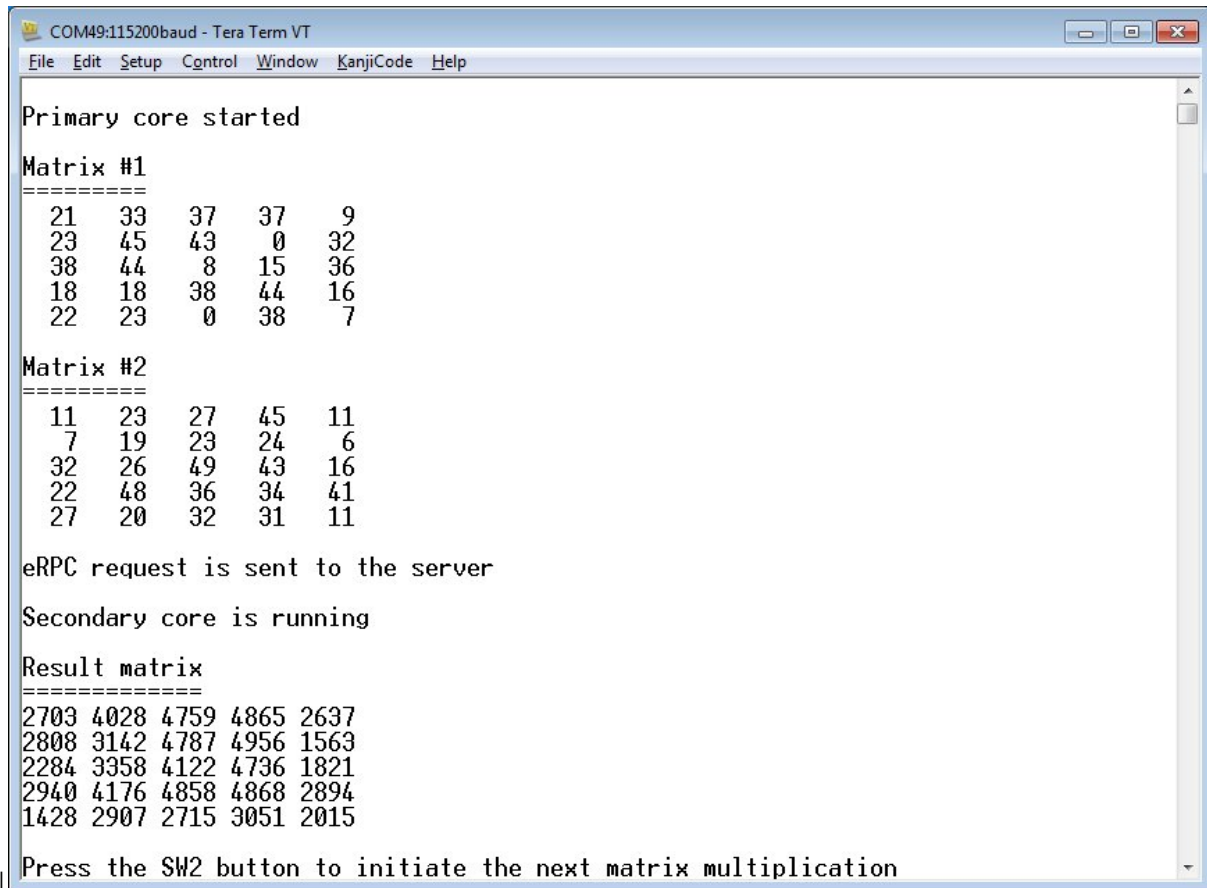
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
 /* Invoke the erpcMatrixMultiply function */
 erpcMatrixMultiply(matrix1, matrix2, result_matrix);
 ...
 /* Check if some error occurred in eRPC */
 if (g_erpc_error_occurred)
 {
 /* Exit program loop */
 break;
 }
 ...
}
```

**Parent topic:**Multiprocessor client application

**Parent topic:**Multiprocessor server application

**Parent topic:** [Create an eRPC application](#)

**Running the eRPC application** Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK\_install\_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



```
COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Primary core started

Matrix #1
=====
 21 33 37 37 9
 23 45 43 0 32
 38 44 8 15 36
 18 18 38 44 16
 22 23 0 38 7

Matrix #2
=====
 11 23 27 45 11
 7 19 23 24 6
 32 26 49 43 16
 22 48 36 34 41
 27 20 32 31 11

eRPC request is sent to the server

Secondary core is running

Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015

Press the SW2 button to initiate the next matrix multiplication
```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

**Parent topic:** [Create an eRPC application](#)

**Parent topic:** [eRPC example](#)

**Other uses for an eRPC implementation** The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the [github.com/EmbeddedRPC](https://github.com/EmbeddedRPC).

**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.

**Changelog eRPC** All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

## Unreleased

### 1.14.0

#### Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

#### Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.
- erpc: Fixed Makefile, install the erpc\_simple\_server header, GitHub PR #447.
- erpc\_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

### 1.13.0

### Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr\_uart, zephyr\_mbox).
- eRPC: Add new Zephyr-related examples.

### Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

### Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

## 1.12.0

### Added

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in Message-Buffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

### Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

## 1.11.0

### Fixed

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.

- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU\_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSBIO.GetNumPorts API use with libusb python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcpy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

## 1.10.0

### Fixed

- eRPC: MU transport layer switched to blocking MU\_SendMsg() API use.

## 1.10.0

### Added

- eRPC: Add TCP\_NODELAY option to python, GitHub PR #298.

### Fixed

- eRPC: MUPort adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.
- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.



- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT\_TRUE with EXPECT\_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg\_lite based transports to changed rpmsg\_lite\_wait\_for\_link\_up() API parameters.
- eRPC, erpcgen: Better distinguish which file can and cannot be linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc\_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest\_enumErrorCode\_allDirection, serialize enums as int32 instead of uint32.

### 1.9.1

#### Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.
- eRPC: Remove ERPC\_PRE\_POST\_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcpy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid of serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.
- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg\_lite\_wait\_for\_link\_up() in rpmsg\_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.

- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

## 1.9.0

### Added

- eRPC: Allow used LIBUSB\_SIO device index being specified from the Python command line argument.

### Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run\_clang\_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g\_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc\_setup\_mbf\_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related API, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

## 1.8.1

### Added

- eRPC: New i2c\_slave\_transport transport introduced.

### Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message\_loggers and pre\_post\_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.
- erpcgen: Fix bison file, GitHub PR #156.

## 1.8.0

### Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char\*, GitHub PR #125.

### Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native \_WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix\_multiply.py example: Add -serial and -baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc\_framed\_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '\$' character from generated symbol name in '\_\$union' suffix, GitHub PR #103.
- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

## 1.7.4

**Added**

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

**Fixed**

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

**1.7.3****Fixed**

- eRPC: Improved the test\_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

**1.7.2****Added**

- eRPC: Add missing doxygen comments for transports.

**Fixed**

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.

**1.7.1****Fixed**

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

## 1.7.0

### Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.
- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

### Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

## 1.6.0

### Added

- eRPC: Added @nullable support for scalar types.

### Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

## 1.5.0

### Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.

- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

#### 1.4.0

##### Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

##### Fixed

- eRPC: win\_flex\_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

#### [1.3.0]

##### Added

- eRPC: New annotation types introduced (@length, @max\_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

#### [1.2.0]

##### Added

- eRPC source directory organization changed.
- Many eRPC improvements.

#### [1.1.0]

##### Added

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

#### [1.0.0]

##### Added

- Initial Release



# 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**

**Overview** The purpose of this document is to describes the [FreeRTOS kernel repo](#) integration into the [NXP MCUXpresso Software Development Kit: mcuxsdk](#). MCUXpresso SDK provides a comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on MCUs from NXP. This project involves the FreeRTOS kernel repo fork with:

- cmake and Kconfig support to allow the configuration and build in MCUXpresso SDK ecosystem
- FreeRTOS OS additions, such as [FreeRTOS driver wrappers](#), RTOS ready FatFs file system, and the implementation of FreeRTOS tickless mode

The history of changes in FreeRTOS kernel repo for MCUXpresso SDK are summarized in [CHANGELOG\\_mcuxsdk.md](#) file.

The MCUXpresso SDK framework also contains a set of FreeRTOS examples which show basic FreeRTOS OS features. This makes it easy to start a new FreeRTOS project or begin experimenting with FreeRTOS OS. Selected drivers and middleware are RTOS ready with related FreeRTOS adaptation layer.

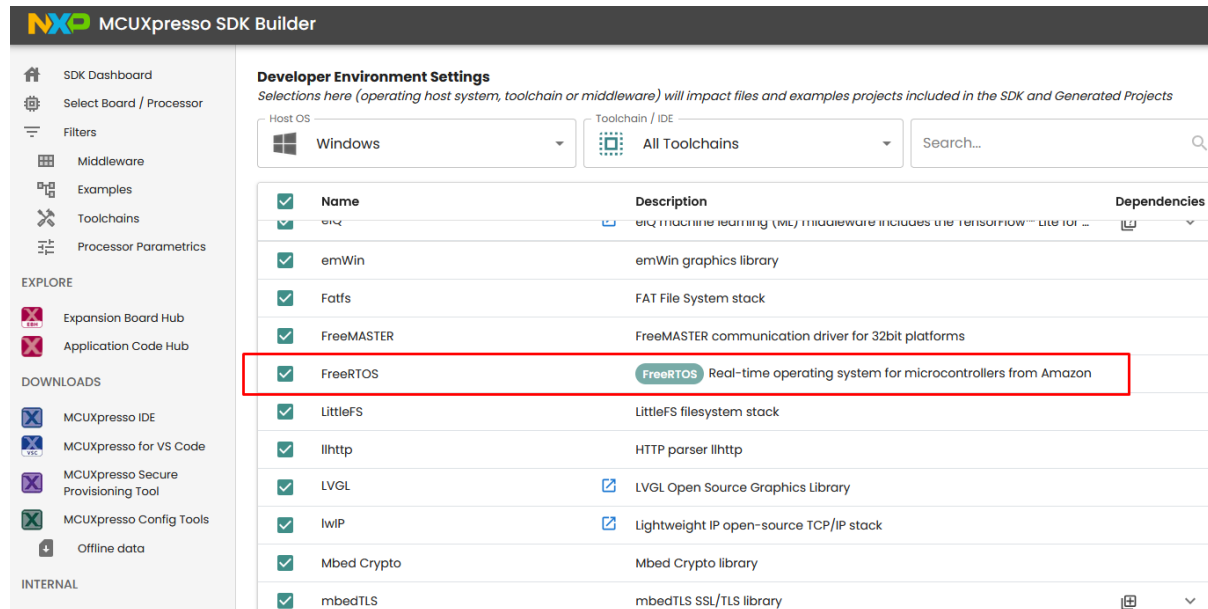
**FreeRTOS example applications** The FreeRTOS examples are written to demonstrate basic FreeRTOS features and the interaction between peripheral drivers and the RTOS.

**List of examples** The list of freertos\_examples, their description and availability for individual supported MCUXpresso SDK development boards can be obtained here: [https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos\\_examples/index.html](https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos_examples/index.html)



**Location of examples** The FreeRTOS examples are located in [mcuxsdk-examples](#) repository, see the `freertos_examples` folder.

Once using MCUXpresso SDK zip packages created via the [MCUXpresso SDK Builder](#) the FreeRTOS kernel library and associated `freertos_examples` are added into final zip package once FreeRTOS components is selected on the Developer Environment Settings page:



The FreeRTOS examples in MCUXpresso SDK zip packages are located in `<MCUXpressoSDK_install_dir>/boards/<board_name>/freertos_examples/` subfolders.

**Building a FreeRTOS example application** For information how to use the cmake and Kconfig based build and configuration system and how to build `freertos_examples` visit: [MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide](#)

Tip: To list all FreeRTOS example projects and targets that can be built via the west build command, use this `west list_project` command in `mcuxsdk` workspace:

```
west list_project -p examples/freertos_examples
```

**FreeRTOS aware debugger plugin** NXP provides FreeRTOS task aware debugger for GDB. The plugin is compatible with Eclipse-based (MCUXpressoIDE) and is available after the installation.

| Task List (FreeRTOS) |           |             |            |          |              |                          |               |
|----------------------|-----------|-------------|------------|----------|--------------|--------------------------|---------------|
| TCB#                 | Task Name | Task Handle | Task State | Priority | Stack Usage  | Event Object             | Runtime       |
| 1                    | task_one  | 0x1fffecc8  | Blocked    | 1 (1)    | 0 B / 880 B  | MyCountingSemaphore (Rx) | 0x0 (0.0%)    |
| 2                    | task_two  | 0x1ffff130  | Blocked    | 2 (2)    | 0 B / 888 B  | MyCountingSemaphore (Rx) | 0x1 (0.1%)    |
| 3                    | IDLE      | 0x1ffff330  | Running    | 0 (0)    | 0 B / 296 B  |                          | 0x3e5 (99.6%) |
| 4                    | Tmr Svc   | 0x1ffff6b8  | Blocked    | 17 (17)  | 28 B / 672 B | TmrQ (Rx)                | 0x3 (0.3%)    |

## FreeRTOS kernel for MCUXpresso SDK ChangeLog

**Changelog FreeRTOS kernel for MCUXpresso SDK** All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

**[Unreleased]****Added**

- Kconfig added CONFIG\_FREERTOS\_USE\_CUSTOM\_CONFIG\_FRAGMENT config to optionally include custom FreeRTOSConfig fragment include file FreeRTOSConfig\_frag.h. File must be provided by application.
- Added missing Kconfig option for configUSE\_PICOLIBC\_TLS.
- Add correct header files to build when configUSE\_NEWLIB\_REENTRANT and configUSE\_PICOLIBC\_TLS is selected in config.

**[11.1.0\_rev0]**

- update amazon freertos version

**[11.0.1\_rev0]**

- update amazon freertos version

**[10.5.1\_rev0]**

- update amazon freertos version

**[10.4.3\_rev1]**

- Apply CM33 security fix from 10.4.3-LTS-Patch-2. See rtos\freertos\freertos\_kernel\History.txt
- Apply CM33 security fix from 10.4.3-LTS-Patch-1. See rtos\freertos\freertos\_kernel\History.txt

**[10.4.3\_rev0]**

- update amazon freertos version.

**[10.4.3\_rev0]**

- update amazon freertos version.

**[9.0.0\_rev3]**

- New features:
  - Tickless idle mode support for Cortex-A7. Add fsl\_tickless\_epit.c and fsl\_tickless\_generic.h in portable/IAR/ARM\_CA9 folder.
  - Enabled float context saving in IAR for Cortex-A7. Added configUSE\_TASK\_FPU\_SUPPORT macros. Modified port.c and portmacro.h in portable/IAR/ARM\_CA9 folder.
- Other changes:
  - Transformed ARM\_CM core specific tickless low power support into generic form under freertos/Source/portable/low\_power\_tickless/.

### [9.0.0\_rev2]

- New features:
  - Enabled MCUXpresso thread aware debugging. Add `freertos_tasks_c_additions.h` and `configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H` and `configFREERTOS_MEMORY_SCHEME` macros.

### [9.0.0\_rev1]

- New features:
  - Enabled `-flto` optimization in GCC by adding `attribute((used))` for `vTaskSwitchContext`.
  - Enabled KDS Task Aware Debugger. Apply FreeRTOS patch to enable `configRECORD_STACK_HIGH_ADDRESS` macro. Modified files are `task.c` and `FreeRTOS.h`.

### [9.0.0\_rev0]

- New features:
  - Example `freertos_sem_static`.
  - Static allocation support RTOS driver wrappers.
- Other changes:
  - Tickless idle rework. Support for different timers is in separated files (`fsl_tickless_systick.c`, `fsl_tickless_lptmr.c`).
  - Removed configuration option `configSYSTICK_USE_LOW_POWER_TIMER`. Low power timer is now selected by linking of appropriate file `fsl_tickless_lptmr.c`.
  - Removed `configOVERRIDE_DEFAULT_TICK_CONFIGURATION` in RVDS port. Use of `attribute((weak))` is the preferred solution. Not same as `_weak`!

### [8.2.3]

- New features:
  - Tickless idle mode support.
  - Added template application for Kinetis Expert (KEx) tool (`template_application`).
- Other changes:
  - Folder structure reduction. Keep only Kinetis related parts.

## FreeRTOS kernel Readme

**MCUXpresso SDK: FreeRTOS kernel** This repository is a fork of FreeRTOS kernel (<https://github.com/FreeRTOS/FreeRTOS-Kernel>)(11.1.0). Modifications have been made to adapt to NXP MCUXpresso SDK. `CMakeLists.txt` and `Kconfig` added to enable FreeRTOS kernel repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository `mcuxsdk-manifests`(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

For more information about the FreeRTOS kernel repo adoption see [README\\_mcuxsdk.md: FreeRTOS kernel for MCUXpresso SDK](#) document.



**Getting started** This repository contains FreeRTOS kernel source/header files and kernel ports only. This repository is referenced as a submodule in [FreeRTOS/FreeRTOS](#) repository, which contains pre-configured demo application projects under [FreeRTOS/Demo](#) directory.

The easiest way to use FreeRTOS is to start with one of the pre-configured demo application projects. That way you will have the correct FreeRTOS source files included, and the correct include paths configured. Once a demo application is building and executing you can remove the demo application files, and start to add in your own application source files. See the [FreeRTOS Kernel Quick Start Guide](#) for detailed instructions and other useful links.

Additionally, for FreeRTOS kernel feature information refer to the [Developer Documentation](#), and [API Reference](#).

Also for contributing and creating a Pull Request please refer to *the instructions here*.

**Getting help** If you have any questions or need assistance troubleshooting your FreeRTOS project, we have an active community that can help on the [FreeRTOS Community Support Forum](#).

## To consume FreeRTOS-Kernel

**Consume with CMake** If using CMake, it is recommended to use this repository using FetchContent. Add the following into your project's main or a subdirectory's CMakeLists.txt:

- Define the source and version/tag you want to use:

```
FetchContent_Declare(freertos_kernel
 GIT_REPOSITORY https://github.com/FreeRTOS/FreeRTOS-Kernel.git
 GIT_TAG main #Note: Best practice to use specific git-hash or tagged version
)
```

In case you prefer to add it as a git submodule, do:

```
git submodule add https://github.com/FreeRTOS/FreeRTOS-Kernel.git <path of the submodule>
git submodule update --init
```

- Add a freertos\_config library (typically an INTERFACE library) The following assumes the directory structure:

– include/FreeRTOSConfig.h

```
add_library(freertos_config INTERFACE)

target_include_directories(freertos_config SYSTEM
INTERFACE
 include
)

target_compile_definitions(freertos_config
INTERFACE
 projCOVERAGE_TEST=0
)
```

In case you installed FreeRTOS-Kernel as a submodule, you will have to add it as a subdirectory:

```
add_subdirectory(${FREERTOS_PATH})
```

- Configure the FreeRTOS-Kernel and make it available
  - this particular example supports a native and cross-compiled build option.

```
set(FREERTOS_HEAP "4" CACHE STRING "" FORCE)
Select the native compile PORT
set(FREERTOS_PORT "GCC_POSIX" CACHE STRING "" FORCE)
Select the cross-compile PORT
if (CMAKE_CROSSCOMPILING)
 set(FREERTOS_PORT "GCC_ARM_CA9" CACHE STRING "" FORCE)
endif()

FetchContent_MakeAvailable(freertos_kernel)
```

- In case of cross compilation, you should also add the following to `freertos_config`:

```
target_compile_definitions(freertos_config INTERFACE ${definitions})
target_compile_options(freertos_config INTERFACE ${options})
```

### Consuming stand-alone - Cloning this repository

To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Kernel.git
```

Using SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Kernel.git
```

### Repository structure

- The root of this repository contains the three files that are common to every port - `list.c`, `queue.c` and `tasks.c`. The kernel is contained within these three files. `croutine.c` implements the optional co-routine functionality - which is normally only used on very memory limited systems.
- The `./portable` directory contains the files that are specific to a particular microcontroller and/or compiler. See the readme file in the `./portable` directory for more information.
- The `./include` directory contains the real time kernel header files.
- The `./template_configuration` directory contains a sample `FreeRTOSConfig.h` to help jumpstart a new project. See the *FreeRTOSConfig.h* file for instructions.

**Code Formatting** FreeRTOS files are formatted using the “`uncrustify`” tool. The configuration file used by `uncrustify` can be found in the [FreeRTOS/CI-CD-GitHub-Actions's uncrustify.cfg](#) file.

**Line Endings** File checked into the FreeRTOS-Kernel repository use unix-style LF line endings for the best compatibility with git.

For optimal compatibility with Microsoft Windows tools, it is best to enable the git `autocrlf` feature. You can enable this setting for the current repository using the following command:

```
git config core.autocrlf true
```

**Git History Optimizations** Some commits in this repository perform large refactors which touch many lines and lead to unwanted behavior when using the `git blame` command. You can configure git to ignore the list of large refactor commits in this repository with the following command:

```
git config blame.ignoreRevsFile .git-blame-ignore-revs
```

**Spelling and Formatting** We recommend using [Visual Studio Code](#), commonly referred to as VSCode, when working on the FreeRTOS-Kernel. The FreeRTOS-Kernel also uses [cSpell](#) as part of its spelling check. The config file for which can be found at [cspell.config.yaml](#). There is additionally a [cSpell plugin for VSCode](#) that can be used as well. [.cSpellWords.txt](#) contains words that are not traditionally found in an English dictionary. It is used by the spellchecker to verify the various jargon, variable names, and other odd words used in the FreeRTOS code base are correct. If your pull request fails to pass the spelling and you believe this is a mistake, then add the word to [.cSpellWords.txt](#). When adding a word please then sort the list, which can be done by running the bash command: `sort -u .cSpellWords.txt -o .cSpellWords.txt`. Note that only the FreeRTOS-Kernel Source Files, *include*, *portable/MemMang*, and *portable/Common* files are checked for proper spelling, and formatting at this time.

### 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

**MCUXpresso SDK: backoffAlgorithm Library** This repository is a fork of backoffAlgorithm library (<https://github.com/FreeRTOS/backoffalgorithm>)(1.3.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable backoffAlgorithm repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository [mcuxsdk-manifests](https://github.com/nxp-mcuxpresso/mcuxsdk-manifests)(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**backoffAlgorithm Library** This repository contains the backoffAlgorithm library, a utility library to calculate backoff period using an exponential backoff with jitter algorithm for retrying network operations (like failed network connection with server). This library uses the “Full Jitter” strategy for the exponential backoff with jitter algorithm. More information about the algorithm can be seen in the [Exponential Backoff and Jitter](#) AWS blog.

The backoffAlgorithm library is distributed under the *MIT Open Source License*.

Exponential backoff with jitter is typically used when retrying a failed network connection or operation request with the server. An exponential backoff with jitter helps to mitigate failed network operations with servers, that are caused due to network congestion or high request load on the server, by spreading out retry requests across multiple devices attempting network operations. Besides, in an environment with poor connectivity, a client can get disconnected at any time. A backoff strategy helps the client to conserve battery by not repeatedly attempting reconnections when they are unlikely to succeed.

See memory requirements for this library [here](#).

**backoffAlgorithm v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.**

**backoffAlgorithm v1.0.0 source code is part of the FreeRTOS 202012.00 LTS release.**

**Reference example** The example below shows how to use the backoffAlgorithm library on a POSIX platform to retry a DNS resolution query for amazon.com.

```
#include "backoff_algorithm.h"
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <unistd.h>
#include <time.h>

/* The maximum number of retries for the example code. */
#define RETRY_MAX_ATTEMPTS (5U)

/* The maximum back-off delay (in milliseconds) for between retries in the example. */
#define RETRY_MAX_BACKOFF_DELAY_MS (5000U)

/* The base back-off delay (in milliseconds) for retry configuration in the example. */
#define RETRY_BACKOFF_BASE_MS (500U)

int main()
{
 /* Variables used in this example. */
 BackoffAlgorithmStatus_t retryStatus = BackoffAlgorithmSuccess;
 BackoffAlgorithmContext_t retryParams;
 char serverAddress[] = "amazon.com";
 uint16_t nextRetryBackoff = 0;

 int32_t dnsStatus = -1;
 struct addrinfo hints;
 struct addrinfo ** pListHead = NULL;
 struct timespec tp;

 /* Add hints to retrieve only TCP sockets in getaddrinfo. */
 (void) memset(&hints, 0, sizeof(hints));

 /* Address family of either IPv4 or IPv6. */
 hints.ai_family = AF_UNSPEC;
 /* TCP Socket. */
 hints.ai_socktype = (int32_t) SOCK_STREAM;
 hints.ai_protocol = IPPROTO_TCP;

 /* Initialize reconnect attempts and interval. */
 BackoffAlgorithm_InitializeParams(&retryParams,
 RETRY_BACKOFF_BASE_MS,
 RETRY_MAX_BACKOFF_DELAY_MS,
 RETRY_MAX_ATTEMPTS);

 /* Seed the pseudo random number generator used in this example (with call to
 * rand() function provided by ISO C standard library) for use in backoff period
 * calculation when retrying failed DNS resolution. */

 /* Get current time to seed pseudo random number generator. */
 (void) clock_gettime(CLOCK_REALTIME, &tp);
 /* Seed pseudo random number generator with seconds. */
 srand(tp.tv_sec);

 do
 {
 /* Perform a DNS lookup on the given host name. */
 dnsStatus = getaddrinfo(serverAddress, NULL, &hints, pListHead);
 }
```

(continues on next page)



(continued from previous page)

```

/* Retry if DNS resolution query failed. */
if(dnsStatus != 0)
{
 /* Generate a random number and get back-off value (in milliseconds) for the next retry.
 * Note: It is recommended to use a random number generator that is seeded with
 * device-specific entropy source so that backoff calculation across devices is different
 * and possibility of network collision between devices attempting retries can be avoided.
 *
 * For the simplicity of this code example, the pseudo random number generator, rand()
 * function is used. */
 retryStatus = BackoffAlgorithm_GetNextBackoff(&retryParams, rand(), &nextRetryBackoff);

 /* Wait for the calculated backoff period before the next retry attempt of querying DNS.
 * As usleep() takes nanoseconds as the parameter, we multiply the backoff period by 1000. */
 (void) usleep(nextRetryBackoff * 1000U);
}
} while((dnsStatus != 0) && (retryStatus != BackoffAlgorithmRetriesExhausted));

return dnsStatus;
}

```

**Building the library** A compiler that supports **C90 or later** such as *gcc* is required to build the library.

Additionally, the library uses a header file introduced in ISO C99, *stdint.h*. For compilers that do not provide this header file, the *source/include* directory contains *stdint.readme*, which can be renamed to *stdint.h* to build the backoffAlgorithm library.

For instance, if the example above is copied to a file named *example.c*, *gcc* can be used like so:

```
gcc -I source/include example.c source/backoff_algorithm.c -o example
./example
```

*gcc* can also produce an output file to be linked:

```
gcc -I source/include -c source/backoff_algorithm.c
```

## Building unit tests

**Checkout Unity Submodule** By default, the submodules in this repository are configured with `update=none` in *.gitmodules*, to avoid increasing clone time and disk space usage of other repositories (like [amazon-freertos](#) that submodules this repository).

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/Unity
```

## Platform Prerequisites

- For running unit tests
  - C89 or later compiler like *gcc*
  - CMake 3.13.0 or later
- For running the coverage target, *gcov* is additionally required.



### Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described [above](#).)
2. Create build directory: `mkdir build && cd build`
3. Run `cmake` while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

**Contributing** See *CONTRIBUTING.md* for information on contributing.

### 4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

#### MCUXpresso SDK: coreHTTP Client Library

This repository is a fork of coreHTTP Client library (<https://github.com/FreeRTOS/corehttp>)(3.0.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreHTTP Client repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

#### coreHTTP Client Library

This repository contains a C language HTTP client library designed for embedded platforms. It has no dependencies on any additional libraries other than the standard C library, [lhttp](#), and a customer-implemented transport interface. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8. This library has also undergone both static code analysis from [Coverity static analysis](#), and validation of memory safety and data structure invariance through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**coreHTTP v3.0.0 source code is part of the FreeRTOS 202210.00 LTS release.**

**coreHTTP v2.0.0 source code is part of the FreeRTOS 202012.00 LTS release.**

**coreHTTP Config File** The HTTP client library exposes configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *core\_http\_config\_defaults.h*. To provide custom values for the configuration macros, a custom config file named *core\_http\_config.h* can be provided by the user application to the library.

By default, a *core\_http\_config.h* custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide `HTTP_DO_NOT_USE_CUSTOM_CONFIG` as a compile time preprocessor macro.

**The HTTP client library can be built by either:**

- Defining a `core_http_config.h` file in the application, and adding it to the include directories for the library build. **OR**
- Defining the `HTTP_DO_NOT_USE_CUSTOM_CONFIG` preprocessor macro for the library build.

**Building the Library** The `httpFilePaths.cmake` file contains the information of all source files and header include paths required to build the HTTP client library.

As mentioned in the *previous section*, either a custom config file (i.e. `core_http_config.h`) OR `HTTP_DO_NOT_USE_CUSTOM_CONFIG` macro needs to be provided to build the HTTP client library.

For a CMake example of building the HTTP library with the `httpFilePaths.cmake` file, refer to the `coverity_analysis` library target in `test/CMakeLists.txt` file.

## Building Unit Tests

### Platform Prerequisites

- For running unit tests, the following are required:
  - **C90 compiler** like `gcc`
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is required for this repository's [CMock test framework](#).
- For running the coverage target, the following are required:
  - `gcov`
  - `lcov`

### Steps to build Unit Tests

1. Go to the root directory of this repository.
2. Run the `cmake` command: `cmake -S test -B build -DBUILD_CLONE_SUBMODULES=ON`
3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Reference examples** The AWS IoT Device SDK for Embedded C repository contains demos of using the HTTP client library [here](#) on a POSIX platform. These can be used as reference examples for the library API.

## Documentation

**Existing Documentation** For pre-generated documentation, please see the documentation linked in the locations below:

| Location                                                       |
|----------------------------------------------------------------|
| <a href="#">AWS IoT Device SDK for Embedded C FreeRTOS.org</a> |

Note that the latest included version of coreHTTP may differ across repositories.

**Generating Documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

**Contributing** See *CONTRIBUTING.md* for information on contributing.

## 4.1.5 corejson

JSON parser.

### Readme

**MCUXpresso SDK: coreJSON Library** This repository is a fork of coreJSON library (<https://github.com/FreeRTOS/corejson>)(3.2.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreJSON repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**coreJSON Library** This repository contains the coreJSON library, a parser that strictly enforces the ECMA-404 JSON standard and is suitable for low memory footprint embedded devices. The coreJSON library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8, and checks against deviations from mandatory rules in the [MISRA coding standard](#). Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from [Coverity static analysis](#), and validation of memory safety through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**coreJSON v3.2.0 source code is part of the FreeRTOS 202210.00 LTS release.**

**coreJSON v3.0.0 source code is part of the FreeRTOS 202012.00 LTS release.**

### Reference example

```

#include <stdio.h>
#include "core_json.h"

int main()
{
 // Variables used in this example.
 JSONStatus_t result;
 char buffer[] = "{\"foo\":\"abc\",\"bar\":{\"foo\":\"xyz\"}}";
 size_t bufferLength = sizeof(buffer) - 1;
 char queryKey[] = "bar.foo";
 size_t queryKeyLength = sizeof(queryKey) - 1;
 char * value;
 size_t valueLength;

 // Calling JSON_Validate() is not necessary if the document is guaranteed to be valid.
 result = JSON_Validate(buffer, bufferLength);

 if(result == JSONSuccess)
 {
 result = JSON_Search(buffer, bufferLength, queryKey, queryKeyLength,
 &value, &valueLength);
 }

 if(result == JSONSuccess)
 {
 // The pointer "value" will point to a location in the "buffer".
 char save = value[valueLength];
 // After saving the character, set it to a null byte for printing.
 value[valueLength] = '\0';
 // "Found: bar.foo -> xyz" will be printed.
 printf("Found: %s -> %s\n", queryKey, value);
 // Restore the original character.
 value[valueLength] = save;
 }

 return 0;
}

```

A search may descend through nested objects when the queryKey contains matching key strings joined by a separator, .. In the example above, bar has the value { "foo": "xyz" }. Therefore, a search for query key bar.foo would output xyz.

**Building coreJSON** A compiler that supports **C90 or later** such as *gcc* is required to build the library.

Additionally, the library uses 2 header files introduced in ISO C99, *stdbool.h* and *stdint.h*. For compilers that do not provide this header file, the *source/include* directory contains *stdbool.readme* and *stdint.readme*, which can be renamed to *stdbool.h* and *stdint.h* respectively.

For instance, if the example above is copied to a file named *example.c*, *gcc* can be used like so:

```
gcc -I source/include example.c source/core_json.c -o example
./example
```

*gcc* can also produce an output file to be linked:

```
gcc -I source/include -c source/core_json.c
```

## Documentation

**Existing documentation** For pre-generated documentation, please see the documentation linked in the locations below:

| Location                                                       |
|----------------------------------------------------------------|
| <a href="#">AWS IoT Device SDK for Embedded C FreeRTOS.org</a> |

Note that the latest included version of the coreJSON library may differ across repositories.

**Generating documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

## Building unit tests

**Checkout Unity Submodule** By default, the submodules in this repository are configured with `update=none` in `.gitmodules`, to avoid increasing clone time and disk space usage of other repositories (like [amazon-freertos](#) that submodules this repository).

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/Unity
```

## Platform Prerequisites

- For running unit tests
  - C90 compiler like gcc
  - CMake 3.13.0 or later
  - Ruby 2.0.0 or later is additionally required for the Unity test framework (that we use).
- For running the coverage target, gcov is additionally required.

## Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described [above](#).)
2. Create build directory: `mkdir build && cd build`
3. Run `cmake` while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Contributing** See *CONTRIBUTING.md* for information on contributing.

### 4.1.6 coremqtt

MQTT publish/subscribe messaging library.

#### MCUXpresso SDK: coreMQTT Library

This repository is a fork of coreMQTT library (<https://github.com/FreeRTOS/coremqtt>)(2.1.1). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreMQTT repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

#### coreMQTT Client Library

This repository contains the coreMQTT library that has been optimized for a low memory footprint. The coreMQTT library is compliant with the [MQTT 3.1.1](#) standard. It has no dependencies on any additional libraries other than the standard C library, a customer-implemented network transport interface, and *optionally* a user-implemented platform time function. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8, and checks against deviations from mandatory rules in the [MISRA coding standard](#). Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from [Coverity static analysis](#), and validation of memory safety through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**coreMQTT v2.1.1 source code is part of the FreeRTOS 20210.01 LTS release.**

**MQTT Config File** The MQTT client library exposes build configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *core\_mqtt\_config\_defaults.h*. To provide custom values for the configuration macros, a custom config file named *core\_mqtt\_config.h* can be provided by the application to the library.

By default, a *core\_mqtt\_config.h* custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide `MQTT_DO_NOT_USE_CUSTOM_CONFIG` as a compile time preprocessor macro.

**Thus, the MQTT library can be built by either:**

- Defining a *core\_mqtt\_config.h* file in the application, and adding it to the include directories list of the library
- OR**
- Defining the `MQTT_DO_NOT_USE_CUSTOM_CONFIG` preprocessor macro for the library build.

**Sending metrics to AWS IoT** When establishing a connection with AWS IoT, users can optionally report the Operating System, Hardware Platform and MQTT client version information of their device to AWS. This information can help AWS IoT provide faster issue resolution and technical support. If users want to report this information, they can send a specially formatted string (see below) in the username field of the MQTT CONNECT packet.

#### Format

The format of the username string with metrics is:

```
<Actual_Username>?SDK=<OS_Name>&Version=<OS_Version>&Platform=<Hardware_Platform>&MQTTLib=<MQTT_Library_name>@<MQTT_Library_version>
```

#### Where

- <Actual\_Username> is the actual username used for authentication, if username and password are used for authentication. When username and password based authentication is not used, this is an empty value.
- <OS\_Name> is the Operating System the application is running on (e.g. FreeRTOS)
- <OS\_Version> is the version number of the Operating System (e.g. V10.4.3)
- <Hardware\_Platform> is the Hardware Platform the application is running on (e.g. WinSim)
- <MQTT\_Library\_name> is the MQTT Client library being used (e.g. coreMQTT)
- <MQTT\_Library\_version> is the version of the MQTT Client library being used (e.g. 1.0.2)

#### Example

- Actual\_Username = "iotuser", OS\_Name = FreeRTOS, OS\_Version = V10.4.3, Hardware\_Platform\_Name = WinSim, MQTT\_Library\_Name = coremqtt, MQTT\_Library\_version = 2.1.1. If username is not used, then "iotuser" can be removed.

```
/* Username string:
 * iotuser?SDK=FreeRTOS&Version=v10.4.3&Platform=WinSim&MQTTLib=coremqtt@2.1.1
 */

#define OS_NAME "FreeRTOS"
#define OS_VERSION "V10.4.3"
#define HARDWARE_PLATFORM_NAME "WinSim"
#define MQTT_LIB "coremqtt@2.1.1"

#define USERNAME_STRING "iotuser?SDK=" OS_NAME "&Version=" OS_VERSION "&Platform=" HARDWARE_PLATFORM_NAME "&MQTTLib=" MQTT_LIB
#define USERNAME_STRING_LENGTH ((uint16_t) (sizeof(USERNAME_STRING) - 1))

MQTTConnectInfo_t connectInfo;
connectInfo.pUserName = USERNAME_STRING;
connectInfo.userNameLength = USERNAME_STRING_LENGTH;
mqttStatus = MQTT_Connect(pMqttContext, &connectInfo, NULL, CONNACK_RECV_TIMEOUT_MS,
↳ pSessionPresent);
```

**Upgrading to v2.0.0 and above** With coreMQTT versions >=v2.0.0, there are breaking changes. Please refer to the *coreMQTT version >=v2.0.0 Migration Guide*.

**Building the Library** The *mqttFilePaths.cmake* file contains the information of all source files and the header include path required to build the MQTT library.

Additionally, the MQTT library requires two header files that are not part of the ISO C90 standard library, *stdbool.h* and *stdint.h*. For compilers that do not provide these header files, the



*source/include* directory contains the files *stdbool.readme* and *stdint.readme*, which can be renamed to *stdbool.h* and *stdint.h*, respectively, to provide the type definitions required by MQTT.

As mentioned in the previous section, either a custom config file (i.e. *core\_mqtt\_config.h*) OR `MQTT_DO_NOT_USE_CUSTOM_CONFIG` macro needs to be provided to build the MQTT library.

For a CMake example of building the MQTT library with the *mqttFilePaths.cmake* file, refer to the *coverity\_analysis* library target in *test/CMakeLists.txt* file.

## Building Unit Tests

**Checkout CMock Submodule** By default, the submodules in this repository are configured with `update=none` in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories (like [amazon-freertos](#) that submodules this repository).

To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/CMock
```

## Platform Prerequisites

- Docker

or the following:

- For running unit tests
  - **C90 compiler** like gcc
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

## Steps to build Unit Tests

1. If using docker, launch the container:
  1. `docker build -t coremqtt .`
  2. `docker run -it -v "$PWD":/workspaces/coreMQTT -w /workspaces/coreMQTT coremqtt`
2. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described [above](#))
3. Run the *cmake* command: `cmake -S test -B build`
4. Run this command to build the library and unit tests: `make -C build all`
5. The generated test executables will be present in *build/bin/tests* folder.
6. Run `cd build && ctest` to execute all tests and view the test run summary.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The *test/cbmc/proofs* directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).



**Reference examples** Please refer to the demos of the MQTT client library in the following locations for reference examples on POSIX and FreeRTOS platforms:

| Platform | Location                                            | Transport Interface Implementation                 |
|----------|-----------------------------------------------------|----------------------------------------------------|
| POSIX    | <a href="#">AWS IoT Device SDK for Embedded C</a>   | POSIX sockets for TCP/IP and OpenSSL for TLS stack |
| FreeRTOS | <a href="#">FreeRTOS/FreeRTOS</a>                   | FreeRTOS+TCP for TCP/IP and mbedTLS for TLS stack  |
| FreeRTOS | <a href="#">FreeRTOS AWS Reference Integrations</a> | Based on Secure Sockets Abstraction                |

## Documentation

**Existing Documentation** For pre-generated documentation, please see the documentation linked in the locations below:

| Location                                          |
|---------------------------------------------------|
| <a href="#">AWS IoT Device SDK for Embedded C</a> |
| <a href="#">FreeRTOS.org</a>                      |

Note that the latest included version of coreMQTT may differ across repositories.

**Generating Documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

**Contributing** See *CONTRIBUTING.md* for information on contributing.

### 4.1.7 coremqtt-agent

The coreMQTT Agent library is a high level API that adds thread safety to the coreMQTT library.

#### Readme

**MCUXpresso SDK: coreMQTT Agent Library** This repository is a fork of coreMQTT Agent library (<https://github.com/FreeRTOS/coremqtt-agent>)(1.2.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreMQTT Agent repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**coreMQTT Agent Library** The coreMQTT Agent library is a high level API that adds thread safety to the [coreMQTT](#) library. The library provides thread safe equivalents to the coreMQTT's APIs, greatly simplifying its use in multi-threaded environments. The coreMQTT Agent library manages the MQTT connection by serializing the access to the coreMQTT library and reducing implementation overhead (e.g., removing the need for the application to repeatedly call to MQTT\_ProcessLoop). This allows your multi-threaded applications to share the same MQTT connection, and enables you to design an embedded application without having to worry about coreMQTT thread safety.

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8, and checks against deviations from mandatory rules in the [MISRA coding standard](#). Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from [Coverity static analysis](#), and validation of memory safety through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**Cloning this repository** This repo uses [Git Submodules](#) to bring in dependent components.

To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/coreMQTT-Agent.git --recurse-submodules
```

Using SSH:

```
git clone git@github.com:FreeRTOS/coreMQTT-Agent.git --recurse-submodules
```

If you have downloaded the repo without using the `--recurse-submodules` argument, you need to run:

```
git submodule update --init --recursive
```

**coreMQTT Agent Library Configurations** The MQTT Agent library uses the same `core_mqtt_config.h` configuration file as coreMQTT, with the addition of configuration constants listed at the top of `core_mqtt_agent.h` and `core_mqtt_agent_command_functions.h`. Documentation for these configurations can be found [here](#).

To provide values for these configuration values, they must be either:

- Defined in `core_mqtt_config.h` used by coreMQTT **OR**
- Passed as compile time preprocessor macros

**Porting the coreMQTT Agent Library** In order to use the MQTT Agent library on a platform, you need to supply thread safe functions for the agent's *messaging interface*.

**Messaging Interface** Each of the following functions must be thread safe.

| Function Pointer          | Description                                                                                                                                                                                                                                                                   |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MQTTAgentMessageSend_t    | A function that sends commands (as MQTTAgentCommand_t * pointers) to be received by MQTTAgent_CommandLoop. This can be implemented by pushing to a thread safe queue.                                                                                                         |
| MQTTAgentMessageRecv_t    | A function used by MQTTAgent_CommandLoop to receive MQTTAgentCommand_t * pointers that were sent by API functions. This can be implemented by receiving from a thread safe queue.                                                                                             |
| MQTTAgentCommandGet_t     | A function that returns a pointer to an allocated MQTTAgentCommand_t structure, which is used to hold information and arguments for a command to be executed in MQTTAgent_CommandLoop(). If using dynamic memory, this can be implemented using malloc().                     |
| MQTTAgentCommandRelease_t | A function called to indicate that a command structure that had been allocated with the MQTTAgentCommandGet_t function pointer will no longer be used by the agent, so it may be freed or marked as not in use. If using dynamic memory, this can be implemented with free(). |

Reference implementations for the interface functions can be found in the [reference examples](#) below.

## Additional Considerations

**Static Memory** If only static allocation is used, then the MQTTAgentCommandGet\_t and MQTTAgentCommandRelease\_t could instead be implemented with a pool of MQTTAgentCommand\_t structures, with a queue or semaphore used to control access and provide thread safety. The below [reference examples](#) use static memory with a command pool.

**Subscription Management** The MQTT Agent does not track subscriptions for MQTT topics. The receipt of any incoming PUBLISH packet will result in the invocation of a single MQTTAgentIncomingPublishCallback\_t callback, which is passed to MQTTAgent\_Init() for initialization. If it is desired for different handlers to be invoked for different incoming topics, then the publish callback will have to manage subscriptions and fan out messages. A platform independent subscription manager example is implemented in the [reference examples](#) below.

**Building the Library** You can build the MQTT Agent source files that are in the *source* directory, and add *source/include* to your compiler's include path. Additionally, the MQTT Agent library requires the coreMQTT library, whose files follow the same *source/* and *source/include* pattern as the agent library; its build instructions can be found [here](#).

If using CMake, the *mqttAgentFilePaths.cmake* file contains the above information of the source files and the header include path from this repository. The same information is found for coreMQTT from *mqttFilePaths.cmake* in the *coreMQTT submodule*.

For a CMake example of building the MQTT Agent library with the *mqttAgentFilePaths.cmake* file, refer to the *coverity\_analysis* library target in *test/CMakeLists.txt* file.

## Building Unit Tests

**Checkout CMock Submodule** To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/CMock
```

### Unit Test Platform Prerequisites

- For running unit tests
  - **C90 compiler** like gcc
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

### Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described [above](#))
2. Run the *cmake* command: `cmake -S test -B build`
3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Reference examples** Please refer to the demos of the MQTT Agent library in the following locations for reference examples on FreeRTOS platforms:

| Location                             |
|--------------------------------------|
| <a href="#">coreMQTT Agent Demos</a> |
| <a href="#">FreeRTOS/FreeRTOS</a>    |

**Documentation** The MQTT Agent API documentation can be found [here](#).

**Generating documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages yourself, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

**Getting help** You can use your Github login to get support from both the FreeRTOS community and directly from the primary FreeRTOS developers on our [active support forum](#). You can find a list of [frequently asked questions](#) [here](#).

**Contributing** See *CONTRIBUTING.md* for information on contributing.

**License** This library is licensed under the MIT License. See the *LICENSE* file.

### 4.1.8 corepkcs11

PKCS #11 key management library.

#### Readme

**MCUXpresso SDK: corePKCS11 Library** This repository is a fork of PKCS #11 key management library (<https://github.com/FreeRTOS/corePKCS11/tree/v3.5.0>)(v3.5.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable corepkcs11 repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**corePKCS11 Library** PKCS #11 is a standardized and widely used API for manipulating common cryptographic objects. It is important because the functions it specifies allow application software to use, create, modify, and delete cryptographic objects, without ever exposing those objects to the application's memory. For example, FreeRTOS AWS reference integrations use a small subset of the PKCS #11 API to, among other things, access the secret (private) key necessary to create a network connection that is authenticated and secured by the [Transport Layer Security \(TLS\)](#) protocol – without the application ever ‘seeing’ the key.

The Cryptoki or PKCS #11 standard defines a platform-independent API to manage and use cryptographic tokens. The name, “PKCS #11”, is used interchangeably to refer to the API itself and the standard which defines it.

This repository contains a software based mock implementation of the PKCS #11 interface (API) that uses the cryptographic functionality provided by Mbed TLS. Using a software mock enables rapid development and flexibility, but it is expected that the mock be replaced by an implementation specific to your chosen secure key storage in production devices.

Only a subset of the PKCS #11 standard is implemented, with a focus on operations involving asymmetric keys, random number generation, and hashing.

The targeted use cases include certificate and key management for TLS authentication and code-sign signature verification, on small embedded devices.

corePKCS11 is implemented on PKCS #11 v2.4.0, the full PKCS #11 standard can be found on the [oasis website](#).

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8, and checks against deviations from mandatory rules in the [MISRA coding standard](#). Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from [Coverity static analysis](#) and validation of memory safety through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**corePKCS11 v3.5.0 source code is part of the FreeRTOS 202210.00 LTS release.**

**corePKCS11 v3.0.0 source code is part of the FreeRTOS 202012.00 LTS release.**

**Purpose** Generally vendors for secure cryptoprocessors such as Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element, or any other type of secure hardware enclave, distribute a PKCS #11 implementation with the hardware. The purpose of the corePKCS11 software only mock library is therefore to provide a non hardware specific PKCS #11 implementation that allows for rapid prototyping and development before switching to a cryptoprocessor specific PKCS #11 implementation in production devices.

Since the PKCS #11 interface is defined as part of the PKCS #11 [specification](#) replacing this library with another implementation should require little porting effort, as the interface will not change. The system tests distributed in this repository can be leveraged to verify the behavior of a different implementation is similar to corePKCS11.

**corePKCS11 Configuration** The corePKCS11 library exposes preprocessor macros which must be defined prior to building the library. A list of all the configurations and their default values are defined in the doxygen documentation for this library.

### Build Prerequisites

**Library Usage** For building the library the following are required:

- **A C99 compiler**
- **mbedcrypto** library from [mbedtls](#) version 2.x or 3.x.
- **pkcs11 API header(s)** available from [OASIS](#) or [OpenSC](#)

Optionally, variables from the `pkcsFilePaths.cmake` file may be referenced if your project uses `cmake`.

**Integration and Unit Tests** In order to run the integration and unit test suites the following are dependencies are necessary:

- **C Compiler**
- **CMake 3.13.0 or later**
- **Ruby 2.0.0 or later** required by CMock.
- **Python 3** required for configuring mbedtls.
- **git** required for fetching dependencies.
- **GNU Make** or **Ninja**

The *mbedtls*, *CMock*, and *Unity* libraries are downloaded and built automatically using the `cmake` `FetchContent` feature.

**Coverage Measurement and Instrumentation** The following software is required to run the coverage target:

- Linux, MacOS, or another POSIX-like environment.
- A recent version of **GCC** or **Clang** with support for gcov-like coverage instrumentation.
- **gcov** binary corresponding to your chosen compiler
- **lcov** from the [Linux Test Project](#)
- **perl** needed to run the `lcov` utility.

Coverage builds are validated on recent versions of Ubuntu Linux.

## Running the Integration and Unit Tests

1. Navigate to the root directory of this repository in your shell.
2. Run **cmake** to construct a build tree: `cmake -S test -B build`
  - You may specify your preferred build tool by appending `-G'Unix Makefiles'` or `-GNinja` to the command above.
  - You may append `-DUNIT_TESTS=0` or `-DSYSTEM_TESTS=0` to disable Unit Tests or Integration Tests respectively.
3. Build the test binaries: `cmake --build ./build --target all`
4. Run `ctest --test-dir ./build` or `cmake --build ./build --target test` to run the tests without capturing coverage.
5. Run `cmake --build ./build --target coverage` to run the tests and capture coverage data.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Reference examples** The FreeRTOS-Labs repository contains demos using the PKCS #11 library [here](#) using FreeRTOS on the Windows simulator platform. These can be used as reference examples for the library API.

**Porting Guide** Documentation for porting corePKCS11 to a new platform can be found on the AWS [docs](#) web page.

corePKCS11 is not meant to be ported to projects that have a TPM, HSM, or other hardware for offloading crypto-processing. This library is specifically meant to be used for development and prototyping.

**Related Example Implementations** These projects implement the PKCS #11 interface on real hardware and have similar behavior to corePKCS11. It is preferred to use these, over corePKCS11, as they allow for offloading Cryptography to separate hardware.

- ARM's [Platform Security Architecture](#).
- Microchip's [cryptoauthlib](#).
- Infineon's [Optiga Trust X](#).

## Documentation

**Existing Documentation** For pre-generated documentation, please see the documentation linked in the locations below:

| Location                                                       |
|----------------------------------------------------------------|
| <a href="#">AWS IoT Device SDK for Embedded C FreeRTOS.org</a> |

Note that the latest included version of corePKCS11 may differ across repositories.



**Generating Documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

**Security** See *CONTRIBUTING* for more information.

**License** This library is licensed under the MIT-0 License. See the LICENSE file.

### 4.1.9 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

#### Readme

**MCUXpresso SDK: FreeRTOS-Plus-TCP Library** This repository is a fork of FreeRTOS-Plus-TCP library (<https://github.com/FreeRTOS/freertos-plus-tcp>)(4.0.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable FreeRTOS-Plus-TCP repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**Introduction** This branch contains unified IPv4 and IPv6 functionalities. Refer to the Getting started Guide (found [here](#)) for more details.

**FreeRTOS-Plus-TCP Library** FreeRTOS-Plus-TCP is a lightweight TCP/IP stack for FreeRTOS. It provides a familiar Berkeley sockets interface, making it as simple to use and learn as possible. FreeRTOS-Plus-TCP's features and RAM footprint are fully scalable, making FreeRTOS-Plus-TCP equally applicable to smaller lower throughput microcontrollers as well as larger higher throughput microprocessors.

This library has undergone static code analysis and checks for compliance with the [MISRA coding standard](#). Any deviations from the MISRA C:2012 guidelines are documented under [MISRA Deviations](#). The library is validated for memory safety and data structure invariance through the [CBMC automated reasoning tool](#) for the functions that parse data originating from the network. The library is also protocol tested using Maxwell protocol tester for both IPv4 and IPv6.

**Getting started** The easiest way to use the 4.0.0 version of FreeRTOS-Plus-TCP is to refer the Getting started Guide (found [here](#)) Another way is to start with the pre-configured demo application project (found in [this directory](#)). That way you will have the correct FreeRTOS source files included, and the correct include paths configured. Once a demo application is building and executing you can remove the demo application files, and start to add in your own application source files. See the [FreeRTOS Kernel Quick Start Guide](#) for detailed instructions and other useful links.

Additionally, for FreeRTOS-Plus-TCP source code organization refer to the [Documentation](#), and [API Reference](#).



**Getting help** If you have any questions or need assistance troubleshooting your FreeRTOS project, we have an active community that can help on the [FreeRTOS Community Support Forum](#). Please also refer to [FAQ](#) for frequently asked questions.

Also see the [Submitting a bugs/feature request](#) section of CONTRIBUTING.md for more details.

**Note:** All the remaining sections are generic and applies to all the versions from V3.0.0 onwards.

**Upgrading to V3.0.0 and V3.1.0** In version 3.0.0 or 3.1.0, the folder structure of FreeRTOS-Plus-TCP has changed and the files have been broken down into smaller logically separated modules. This change makes the code more modular and conducive to unit-tests. FreeRTOS-Plus-TCP V3.0.0 improves the robustness, security, and modularity of the library. Version 3.0.0 adds comprehensive unit test coverage for all lines and branches of code and has undergone protocol testing, and penetration testing by AWS Security to reduce the exposure to security vulnerabilities. Additionally, the source files have been moved to a `source` directory. This change requires modification of any existing project(s) to include the modified source files and directories. There are examples on how to use the new files and directory structure. For an example based on the Xilinx Zynq-7000, use the code in this [branch](#) and follow these [instructions](#) to build and run the demo.

**FreeRTOS-Plus-TCP V3.1.0 source code(.c .h) is part of the FreeRTOS 202210.00 LTS release.**

**Generating pre V3.0.0 folder structure for backward compatibility:** If you wish to continue using a version earlier than V3.0.0 i.e. continue to use your existing source code organization, a script is provided to generate the folder structure similar to [this](#).

**Note:** After running the script, while the `.c` files will have same names as the pre V3.0.0 source, the files in the `include` directory will have different names and the number of files will differ as well. This should, however, not pose any problems to most projects as projects generally include all files in a given directory.

Running the script to generate pre V3.0.0 folder structure: For running the script, you will need Python version > 3.7. You can download/install it from [here](#).

Once python is downloaded and installed, you can verify the version from your terminal/command window by typing `python --version`.

To run the script, you should switch to the FreeRTOS-Plus-TCP directory that was created using the *Cloning this repository* step above. And then run `python <Path/to/the/script>/GenerateOriginalFiles.py`.

## To consume FreeRTOS+TCP

**Consume with CMake** If using CMake, it is recommended to use this repository using FetchContent. Add the following into your project's main or a subdirectory's CMakeLists.txt:

- Define the source and version/tag you want to use:

```
FetchContent_Declare(freertos_plus_tcp
 GIT_REPOSITORY https://github.com/FreeRTOS/FreeRTOS-Plus-TCP.git
 GIT_TAG master #Note: Best practice to use specific git-hash or tagged version
 GIT_SUBMODULES "" # Don't grab any submodules since not latest
)
```

- Configure the FreeRTOS-Kernel and make it available
  - this particular example supports a native and cross-compiled build option.

```

set(FREERTOS_PLUS_FAT_DEV_SUPPORT OFF CACHE BOOL "" FORCE)
Select the native compile PORT
set(FREERTOS_PLUS_FAT_PORT "POSIX" CACHE STRING "" FORCE)
Select the cross-compile PORT
if (CMAKE_CROSSCOMPILING)
 # Eg. Zynq 2019_3 version of port
 set(FREERTOS_PLUS_FAT_PORT "ZYNQ_2019_3" CACHE STRING "" FORCE)
endif()

FetchContent_MakeAvailable(freertos_plus_tcp)

```

**Consuming stand-alone** This repository uses [Git Submodules](#) to bring in dependent components.

Note: If you download the ZIP file provided by GitHub UI, you will not get the contents of the submodules. (The ZIP file is also not a valid Git repository)

To clone using HTTPS:

```

git clone https://github.com/FreeRTOS/FreeRTOS-Plus-TCP.git ./FreeRTOS-Plus-TCP
cd ./FreeRTOS-Plus-TCP
git submodule update --checkout --init --recursive tools/CMock test/FreeRTOS-Kernel

```

Using SSH:

```

git clone git@github.com:FreeRTOS/FreeRTOS-Plus-TCP.git ./FreeRTOS-Plus-TCP
cd ./FreeRTOS-Plus-TCP
git submodule update --checkout --init --recursive tools/CMock test/FreeRTOS-Kernel

```

**Porting** The porting guide is available on [this page](#).

**Repository structure** This repository contains the FreeRTOS-Plus-TCP repository and a number of supplementary libraries for testing/PR Checks. Below is the breakdown of what each directory contains:

- tools
  - This directory contains the tools and related files (CMock/uncrustify) required to run tests/checks on the TCP source code.
- tests
  - This directory contains all the tests (unit tests and CBMC) and the dependencies ([FreeRTOS-Kernel/Litani-port](#)) the tests require.
- source/portable
  - This directory contains the portable files required to compile the FreeRTOS-Plus-TCP source code for different hardware/compilers.
- source/include
  - The include directory has all the ‘core’ header files of FreeRTOS-Plus-TCP source.
- source
  - This directory contains all the [.c] source files.

**Note** At this time it is recommended to use `BufferAllocation_2.c` in which case it is essential to use the `heap_4.c` memory allocation scheme. See [memory management](#).

**Kernel sources** The FreeRTOS Kernel Source is in [FreeRTOS/FreeRTOS-Kernel repository](#), and it is consumed by testing/PR checks as a submodule in this repository.

The version of the FreeRTOS Kernel Source in use could be accessed at `./test/FreeRTOS-Kernel` directory.

**CBMC** The `test/cbmc/proofs` directory contains CBMC proofs.

To learn more about CBMC and proofs specifically, review the training material [here](#).

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).