MCUXpresso SDK Documentation

Release 25.12.00

NXP
Dec 18, 2025

Table of contents

1 EVK-MIMX8MNDDR3L **3**
- 1.1 Overview . . . 3
- 1.2 Getting Started with MCUXpresso SDK Package . . . 3
  - 1.2.1 Getting Started with MCUXpresso for EVK-MIMX8MNDDR3L . . . 3
- 1.3 Getting Started with MCUXpresso SDK GitHub . . . 19
  - 1.3.1 Getting Started with MCUXpresso SDK Repository . . . 19
- 1.4 Release Notes . . . 43
  - 1.4.1 MCUXpresso SDK Release Notes . . . 43
- 1.5 ChangeLog . . . 46
  - 1.5.1 MCUXpresso SDK Changelog . . . 46
- 1.6 Driver API Reference Manual . . . 83
- 1.7 Middleware Documentation . . . 83
  - 1.7.1 Multicore . . . 83
  - 1.7.2 FreeMASTER . . . 83
  - 1.7.3 FreeRTOS . . . 83

2 MIMX8MN6 **85**
- 2.1 ASRC: Asynchronous sample rate converter . . . 85
- 2.2 ASRC Driver . . . 85
- 2.3 ASRC SDMA Driver . . . 98
- 2.4 CACHE: ARMV7-M7 CACHE Memory Controller . . . 102
- 2.5 Clock . . . 105
- 2.6 ECSPI: Enhanced Configurable Serial Peripheral Interface Driver . . . 137
- 2.7 ECSPI Driver . . . 137
- 2.8 ECSPI SDMA Driver . . . 151
- 2.9 ENET: Ethernet MAC Driver . . . 153
- 2.10 GPC: General Power Controller Driver . . . 184
- 2.11 GPIO: General-Purpose Input/Output Driver . . . 188
- 2.12 GPT: General Purpose Timer . . . 193
- 2.13 I2C: Inter-Integrated Circuit Driver . . . 200
- 2.14 I2C Driver . . . 200
- 2.15 Iomuxc_driver . . . 212
- 2.16 ISI: Image Sensing Interface . . . 232
- 2.17 Common Driver . . . 248
- 2.18 LCDIF: LCD interface . . . 260
- 2.19 MU: Messaging Unit . . . 272
- 2.20 OCOTP: On Chip One-Time Programmable controller. . . . 280
- 2.21 PDM: Microphone Interface . . . 283
- 2.22 PDM Driver . . . 283
- 2.23 PDM SDMA Driver . . . 301
- 2.24 PWM: Pulse Width Modulation Driver . . . 303
- 2.25 RDC: Resource Domain Controller . . . 310
- 2.26 RDC_SEMA42: Hardware Semaphores Driver . . . 316
- 2.27 SAI: Serial Audio Interface . . . 318
- 2.28 SAI Driver . . . 318
- 2.29 SAI SDMA Driver . . . 344

2.30 SDMA: Smart Direct Memory Access (SDMA) Controller Driver . . . 347
2.31 SEMA4: Hardware Semaphores Driver . . . 365
2.32 SNVS: Secure Non-Volatile Storage . . . 369
2.33 Secure Non-Volatile Storage High-Power . . . 369
2.34 Secure Non-Volatile Storage Low-Power . . . 377
2.35 SPDIF: Sony/Philips Digital Interface . . . 384
2.36 SRC: System Reset Controller Driver . . . 397
2.37 TMU: Thermal Management Unit Driver . . . 400
2.38 UART: Universal Asynchronous Receiver/Transmitter Driver . . . 406
2.39 UART Driver . . . 406
2.40 UART FreeRTOS Driver . . . 422
2.41 UART SDMA Driver . . . 422
2.42 WDOG: Watchdog Timer Driver . . . 424

3 Middleware **429**
3.1 Motor Control . . . 429
3.1.1 FreeMASTER . . . 429
3.2 MultiCore . . . 466
3.2.1 Multicore SDK . . . 466

4 RTOS **565**
4.1 FreeRTOS . . . 565
4.1.1 FreeRTOS kernel . . . 565
4.1.2 FreeRTOS drivers . . . 565
4.1.3 backoffalgorithm . . . 565
4.1.4 corehttp . . . 565
4.1.5 corejson . . . 565
4.1.6 coremqtt . . . 566
4.1.7 corepkcs11 . . . 566
4.1.8 freertos-plus-tcp . . . 566

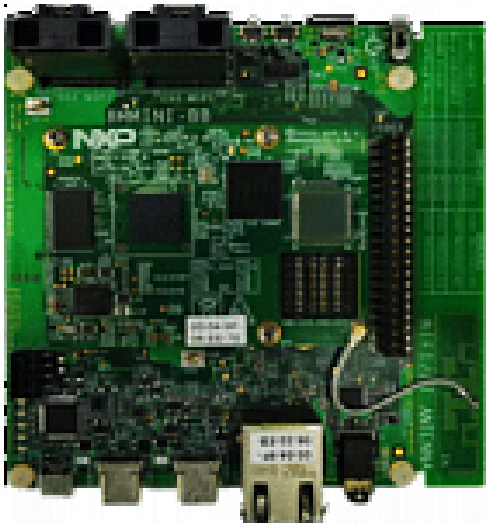
This documentation contains information specific to the evkmimx8mnddr3l board.

Chapter 1

EVK-MIMX8MNDDR3L

1.1 Overview

The i.MX 8M Nano family of boards provides a powerful and flexible development system for NXP's Cortex-M7 MCUs.



MCU device and part on board is shown below:

- Device: MIMX8MN6
- PartNumber: MIMX8MN6DVTJZ

1.2 Getting Started with MCUXpresso SDK Package

1.2.1 Getting Started with MCUXpresso for EVK-MIMX8MNDDR3L

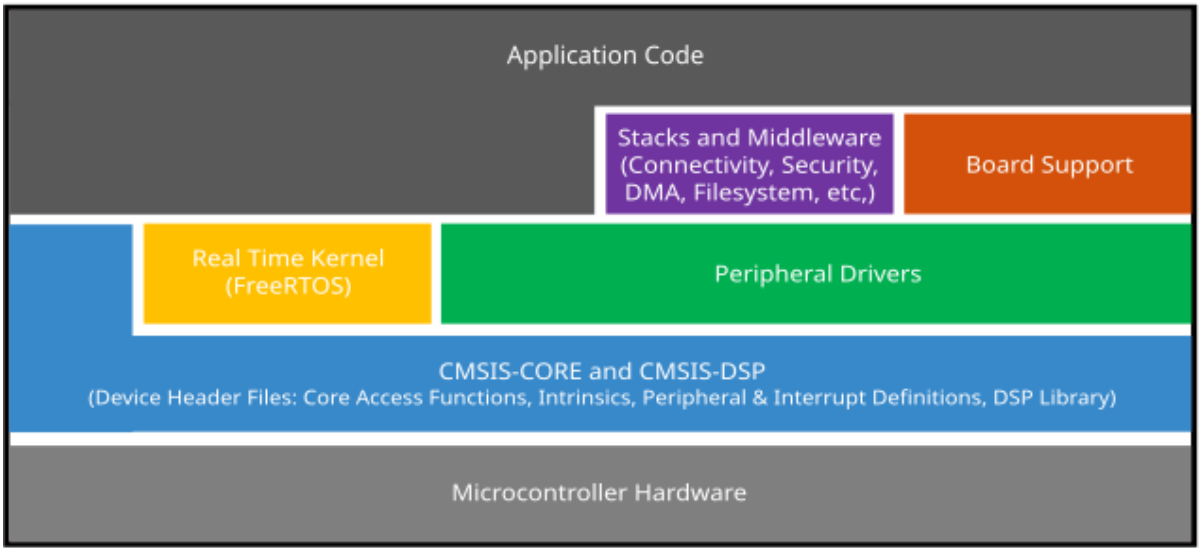
Overview

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 demo applications. The MCUXpresso SDK also contains optional RTOS integrations such as FreeRTOS and Azure RTOS, and device stack to support rapid development on devices.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for i.MX 8M Nano* (document MCUXSDKIMX8MNRN).

For the latest version of this and other MCUXpresso SDK documents, see the MCUXpresso SDK homepage MCUXpresso-SDK: Software Development Kit for MCUXpresso.



MCUXpresso SDK board support folders

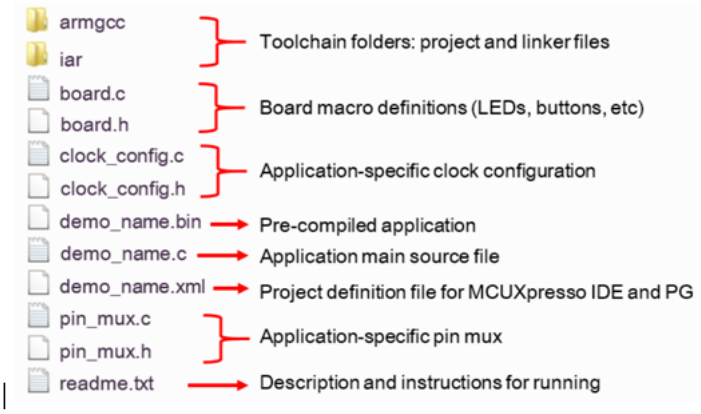
MCUXpresso SDK board support provides example applications for NXP development and evaluation boards for Arm Cortex-M cores. Board support packages are found inside of the top level boards folder, and each supported board has its own folder (MCUXpresso SDK package can support multiple boards). Within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

- cmsis_driver_examples: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- demo_apps: Full-featured applications intended to 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 intended to concisely illustrate how to use the MCUXpresso SDK's peripheral drivers for a single use case.
- rtos_examples: Basic FreeRTOS OS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers
- multicore_examples: Simple applications intended to concisely illustrate how to use middleware/multicore stack.

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

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

In the hello_world application folder you see the following contents:



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

Parent topic:*MCUXpresso SDK board support folders*

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

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

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

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

Parent topic:*MCUXpresso SDK board support folders*

Toolchain introduction

The MCUXpresso SDK release for i.MX 8M Nano includes the build system to be used with some toolchains. In this chapter, the toolchain support is presented and detailed.

Compiler/Debugger The MCUXpresso SDK i.MX 8M Nano release supports building and debugging with the toolchains listed in Table 1.

The user can choose the appropriate one for development.

- Arm GCC + SEGGER J-Link GDB Server. This is a command line tool option and it supports both Windows OS and Linux OS.
- IAR Embedded Workbench for Arm and SEGGER J-Link software. The IAR Embedded Workbench is an IDE integrated with editor, compiler, debugger, and other components. The SEGGER J-Link software provides the driver for the J-Link Plus debugger probe and supports the device to attach, debug, and download.

Compiler/Debugger	Supported host OS	Debug probe	Tool website
ArmGCC/J-Link GDB server	Windows OS/Linux OS	J-Link Plus	developer.arm.com/open-source/gnu-toolchain/gnu-rm

www.segger.com

| |IAR/J-Link|Windows OS|J-Link Plus|www.iar.com

www.segger.com

|

Download the corresponding tools for the specific host OS from the website.

Note: To support i.MX 8M Nano, the patch for IAR should be installed. The patch named iar_support_patch_imx8mn.zip can be used with MCUXpresso SDK. See the readme.txt in the patch for additional information about patch installation.

Parent topic:*Toolchain introduction*

Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the i.MX 8MN EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

Build an example application Before using IAR, get the IAR patch, iar_support_patch_imx8mn.zip. Install the i.MX8MN support patch following the guides in readme.txt located in the archive.

Do the following steps to build the hello_world example application.

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

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

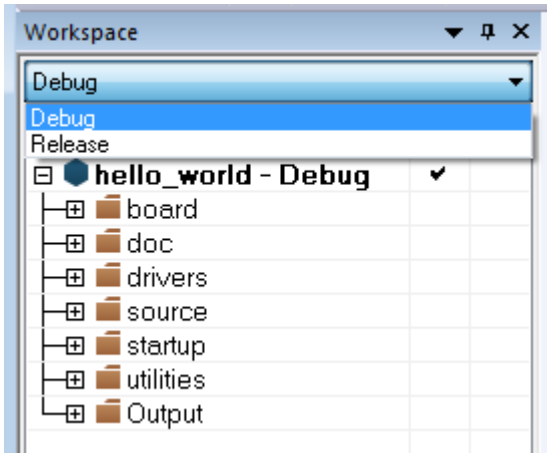
   Using the i.MX 8MN EVK hardware platform as an example, the hello_world workspace is located in:

   ```
   <install_dir>/boards/evkmimx8mn/demo_apps/hello_world/iar/hello_world.eww
   ```

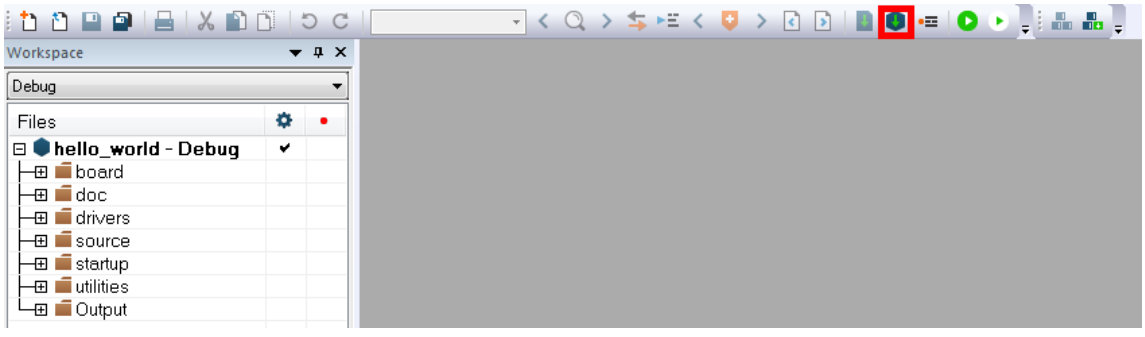
   Other example applications may have additional folders in their path.

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

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



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

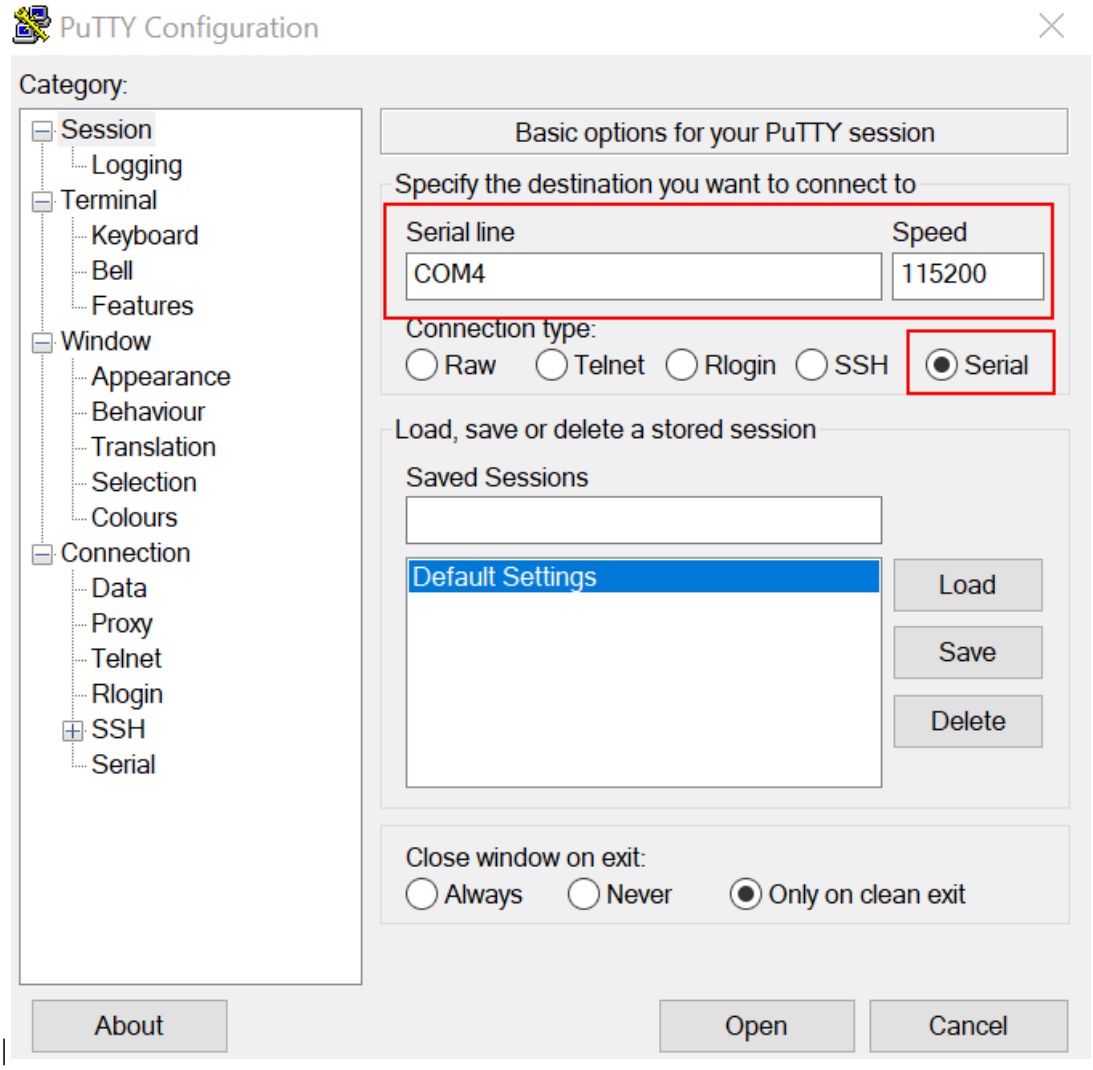


4. The build completes without errors.

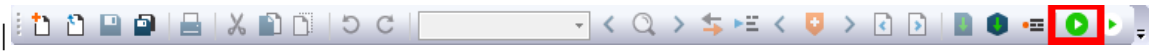
Parent topic:*Run a demo application using IAR*

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

1. This board supports the J-Link PLUS debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from http://www.segger.com/downloads/jlink/.
2. Connect the development platform to your PC via USB cable between the USB-UART MICRO USB connector and the PC USB connector, then connect 12 V power supply and J-Link Plus to the device.
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 baud rate
   2. No parity
   3. 8 data bits
   4. 1 stop bit



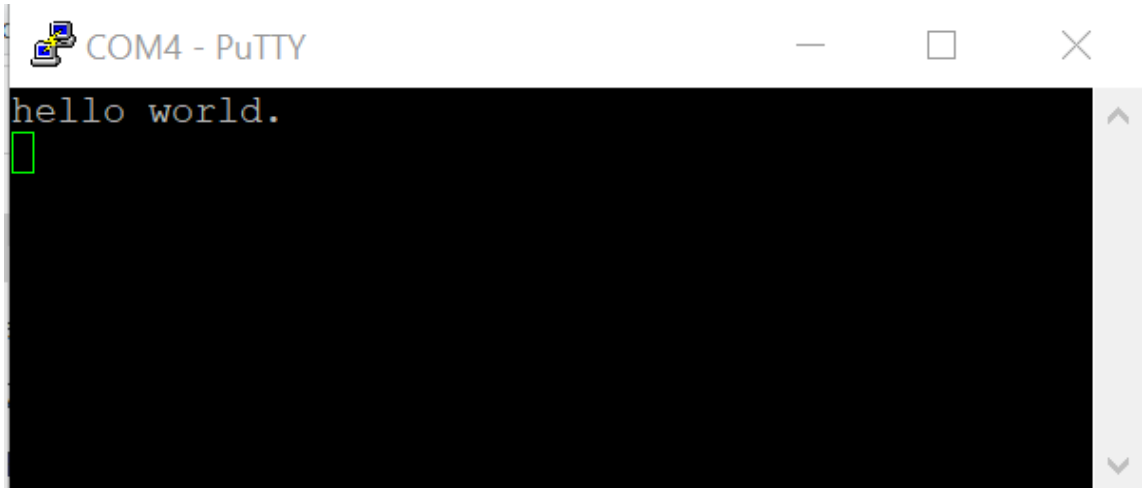
4. In IAR, click **Download and Debug** to download the application to the target.



5. The application then downloads to the target and automatically runs to the $\mathrm{main}()$ function.

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

7. The hello_world application now runs and a banner appears on the terminal. If this does not occur, check your terminal settings and connections.



Note: For downloading the DDR target application, insert one TF card with U-Boot code. This is required both on IAR and GCC.

Note: For converting the DDR target elf to bin, run the following commands (take git bash console as example).
1. For the elf file built by Arm GCC:
 $ <ARMGCC PATH>/bin/arm-none-eabi-objcopy.exe -Obinary --remove-section=.stacktop_and_pc <hello_world_<mcore type>.elf> <hello_world_<mcore type>.bin>

2. For the elf/out file built by IAR:
 $ <ARMGCC PATH>/bin/arm-none-eabi-objcopy.exe --remove-section=.stacktop_and_pc hello_world_<mcore type>.elf hello_world_<mcore type>_stripped.elf
 $ <IAR PATH>/arm/bin/ielftool.exe --bin hello_world_<mcore type>_stripped.elf hello_world_<mcore type>.bin

Parent topic:*Run a demo application using IAR*

Run a demo using ARMGCC / VSCODE

This section describes the steps to run an example application from the SDK archive using the ARMGCC / VSCODE toolchain.

Refer to the *running a demo using MCUXpresso VSC* section for detailed instructions on setting up and configuring your project in Visual Studio Code.

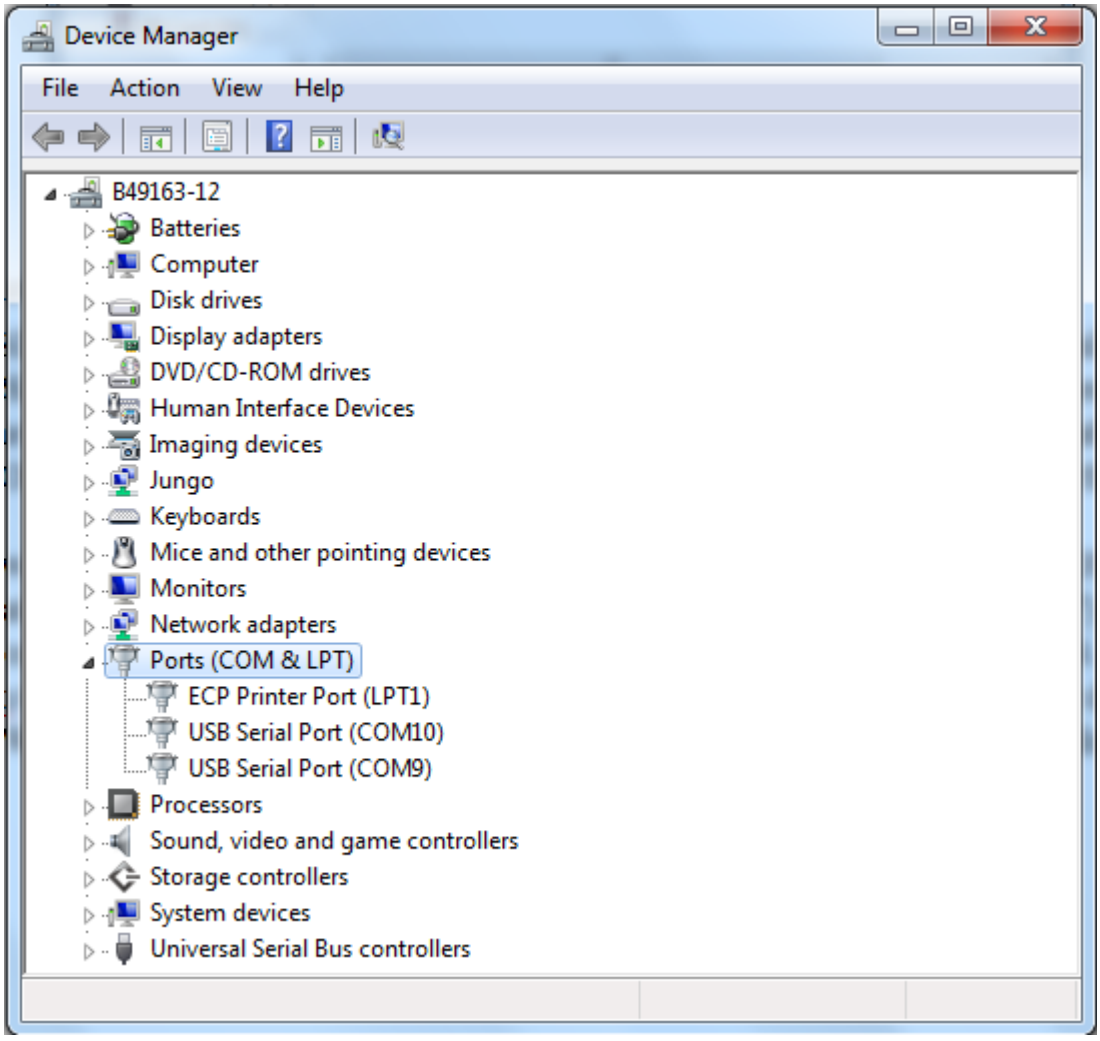
Refer to the *CLI* section for detailed instructions on building and running your project from the command line.

Running an application by U-Boot

This section describes the steps to write a bootable SDK bin file to TCM or DRAM with the prebuilt U-Boot image for the i.MX processor. The following steps describe how to use the U-Boot:

1. Connect the **DEBUG UART** slot on the board to your PC through the USB cable. The Windows OS installs the USB driver automatically, and the Ubuntu OS finds the serial devices as well.

2. On Windows OS, open the device manager, find **USB serial Port** in **Ports (COM and LPT)**. Assume that the ports are COM9 and COM10. One port is for the debug message from the Cortex-A53 and the other is for the Cortex-M7. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name /dev/ttyUSB* to determine your debug port. Similar to Windows OS, opening both is beneficial for development.



3. Build the application (for example, hello_world) to get the bin file (hello_world.bin).
4. Prepare an SD card with the prebuilt U-Boot image and copy bin file (hello_world.bin) into the SD card. Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the dipswitch configuration.
5. Open your preferred serial terminals for the serial devices, setting the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity, then power on the board.
6. Power on the board and hit any key to stop autoboot in the terminals, then enter to U-Boot command line mode. You can then write the image and run it from TCM or DRAM with the following commands:
    1. If the hello_world.bin is made from the debug/release target, which means the binary file will run at TCM, use the following commands to boot:
        - fatload mmc 1:1 0x48000000 hello_world.bin
        - cp.b 0x48000000 0x7e0000 0x20000

- bootaux 0x7e0000

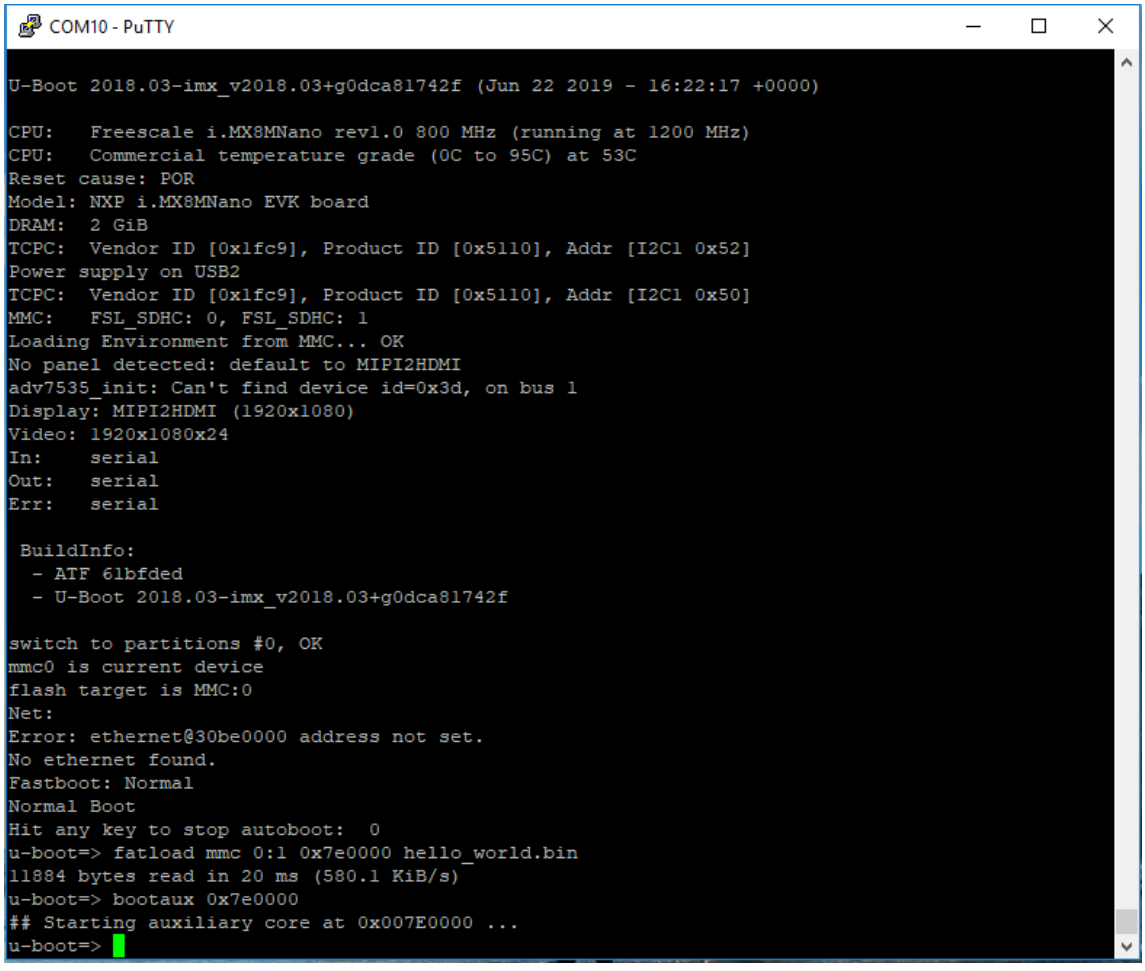
2. If the hello_world.bin is made from the ddr_debug/ddr_release target, which means the binary file runs at DRAM, use the following commands:
   - fatload mmc 1:1 0x80000000 hello_world.bin
   - dcache flush
   - bootaux 0x80000000
3. If the hello_world.bin is made from the flash_debug/flash_releasetarget, which means the binary file runs at nor_flash, use the following commands:
   - sf probe
   - sf read 0x80000000 0 4
   - fatload mmc 1:1 0x80000000 flash.bin
   - dcache flush
   - sf erase 0 0x20000
   - sf write 0x80000000 0 0x20000
   - bootaux 0x8000000 **Note:** If the Linux OS kernel runs together with M7, make sure the correct dtb file is used. This dtb file reserves resources used by M7 and avoids the Linux kernel from configuring them. Use the following command in U-Boot before running the kernel:

- DDR4 board

```
setenv fdtfile 'fsl-imx8mn-ddr4-evk-rpmsg.dtb'
save
```

- LPDDR4 board

```
setenv fdtfile 'fsl-imx8mn-evk-rpmsg.dtb'
save
```

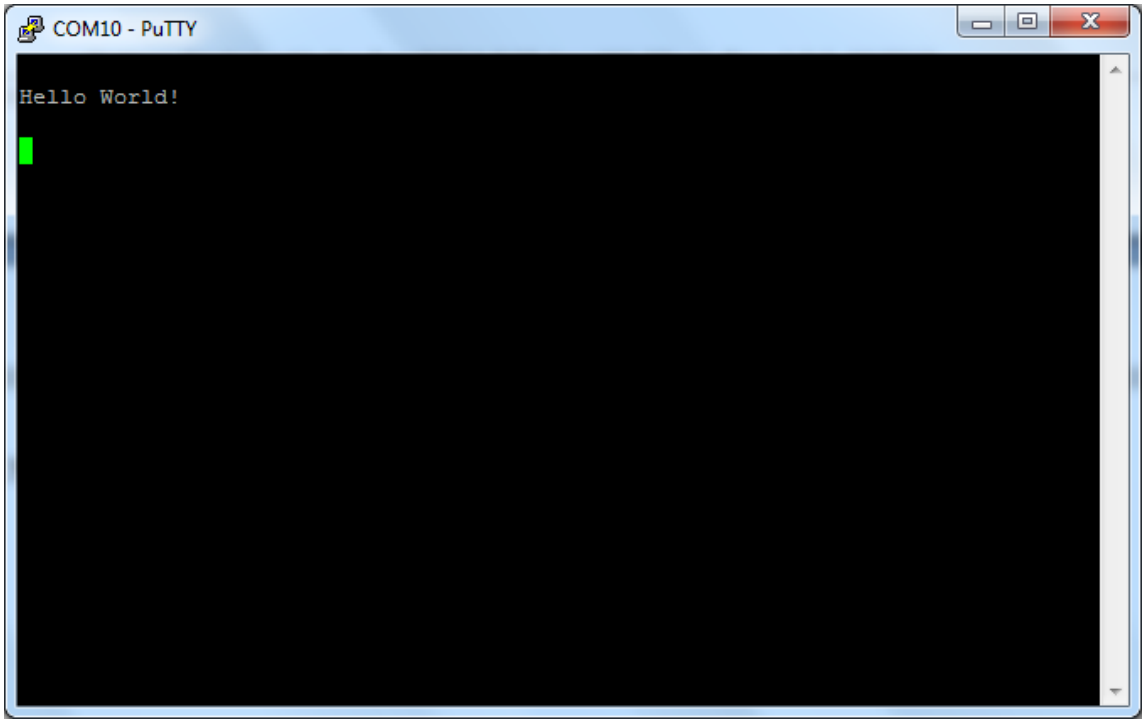


|

|

|

7. Open another terminal application on the PC, such as PuTTY 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:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit
8. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



Run a flash target demo by UUU

This section describes the steps to use the UUU to build and run example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the i.MX 8M Nano EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

Set up environment This section contains the steps to install the necessary components required to build and run a MCUXpresso SDK demo application, as supported by the MCUXpresso SDK.

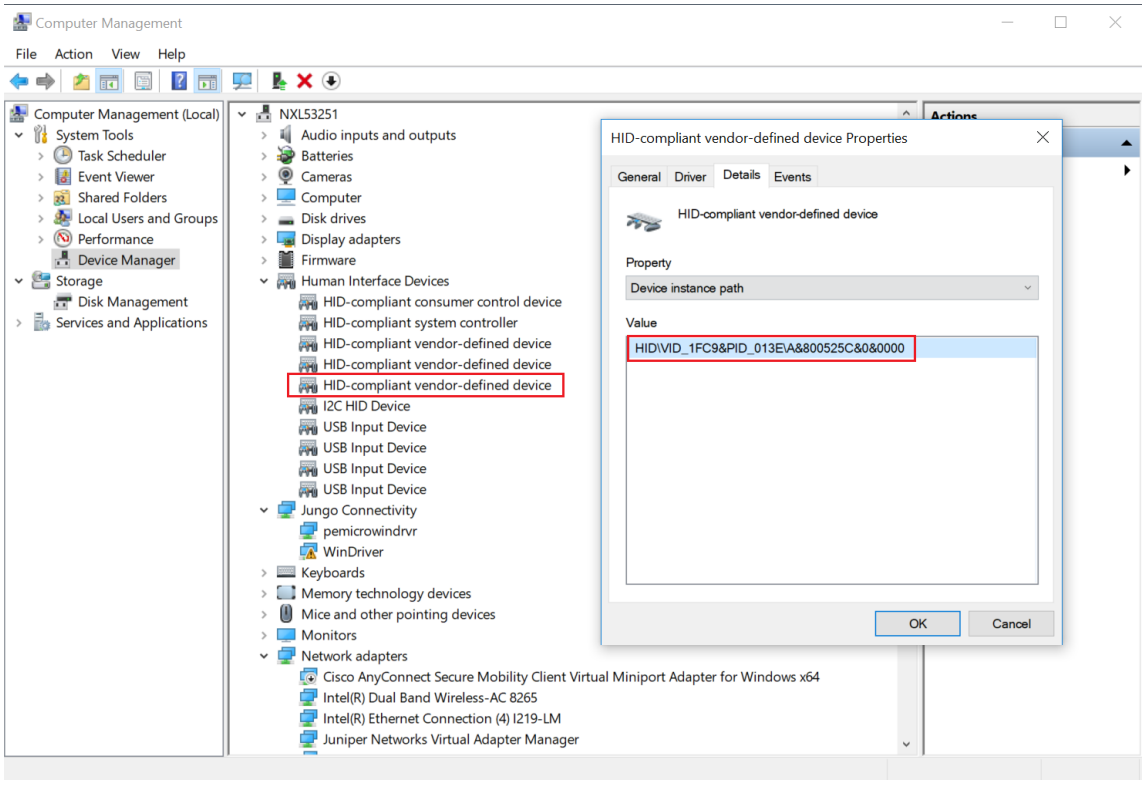
Download the MfgTool The Universal Upgrade Utility (UUU) is an upgraded version of MfgTool. It is a command line tool that aims at installing the bootloader to various storage including SD, QSPI, and so on, for i.MX series devices with ease.

The tool can be downloaded from github. Use version 1.2.135 or higher for full support for the M7 image. Download uuu.exe for Windows OS, or download UUU for Linux. Configure the path so that the executable can later be called anywhere in the command line.

Parent topic: Set up environment

Switch to Download Mode The board needs to be in Download Mode mode for UUU to download images:

1. Set the board boot mode to Download Mode [SW1101:1000[1-4]].
2. Connect the development platform to your PC via USB cable between the SERIAL port and the PC USB connector. The SERIAL port is J301 on the base board.
3. The PC recognizes the i.MX 8M Nano device as (VID:PID)=(1FC9:013E), as shown in Figure 1.



Parent topic: Set up environment

Parent topic:*Run a flash target demo by UUU*

Build an example application The following steps guide you through opening the hello_world example application. These steps may change slightly for other example applications, as some of these applications may have additional layers of folders in their paths.

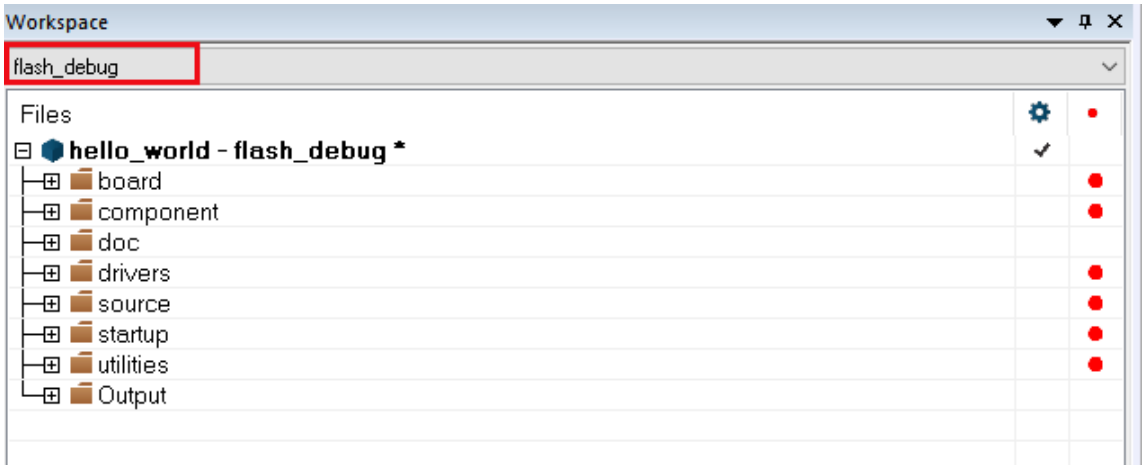
1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

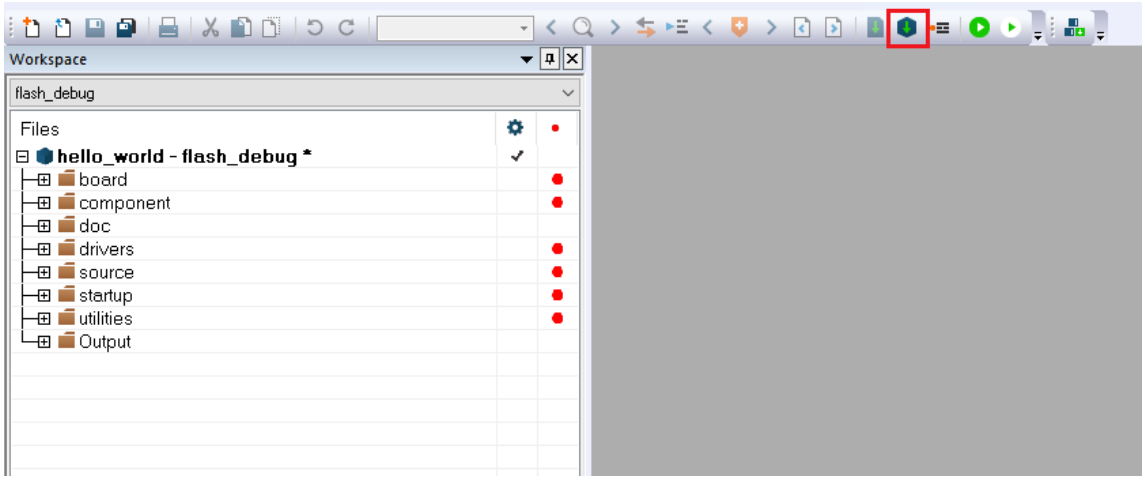
   Using the i.MX 8M Nano EVK board as an example, the workspace is located in:

   ```
   <install_dir>/boards/evkmimx8mn/demo_apps/hello_world/iar/hello_world.eww
   ```

2. Select the desired build target from the drop-down. For this example, select **hello_world – flash_debug**.



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

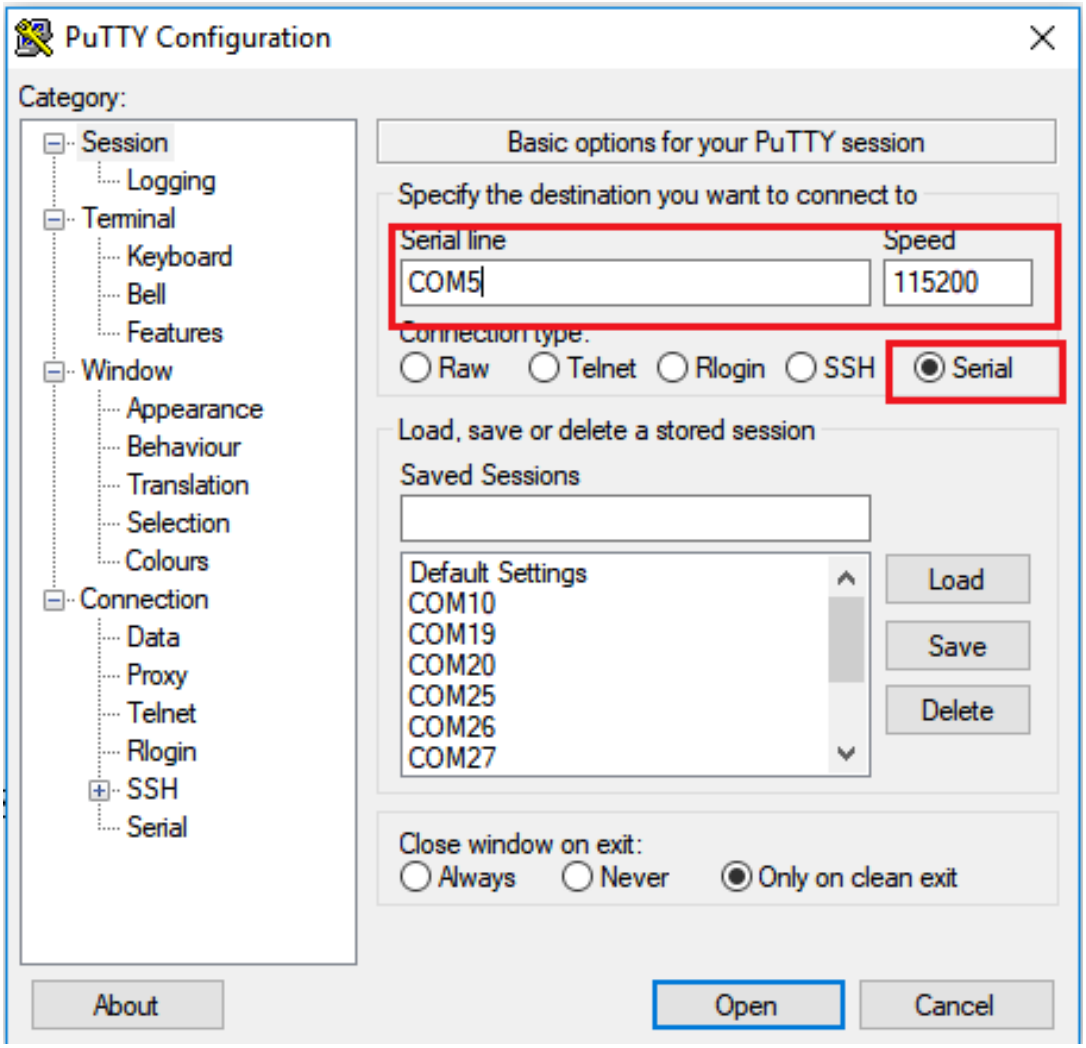


4. The build completes without errors.
5. Rename the generated hello_world.bin to m7_flash.bin, then copy it to the uuu tool directory.

Parent topic:*Run a flash target demo by UUU*

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

1. Connect the development platform to your PC via USB cable between the J901 USB DEBUG connector and the PC. It provides console output while using UUU.
2. Connect the J301 USB Type-C connector and the PC. It provides the data path for UUU.
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 baud rate
    2. No parity
    3. 8 data bits

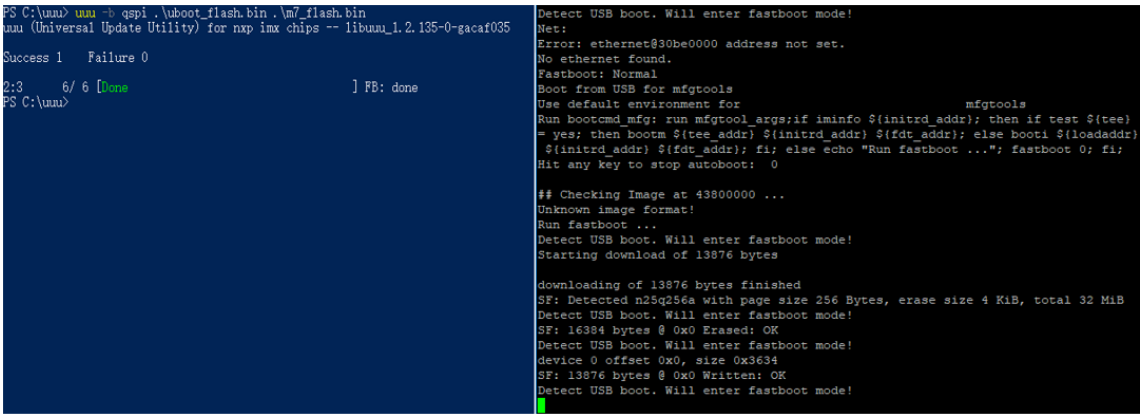


4. 1 stop bit |

|

4. Get the fspi version U-Boot image from release package and rename it to uboot_flash.bin.
5. In the command line, execute uuu with the *-b qspi* switch: uuu -b qspi uboot_flash.bin m7_flash.bin .

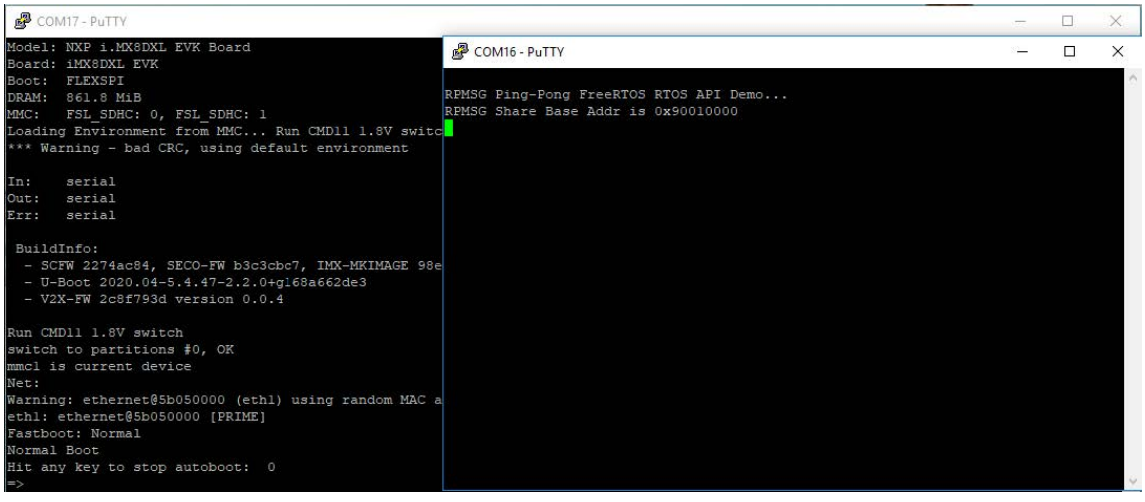
   The UUU puts the platform into fast boot mode and automatically flashes the target boot-loader to QSPI. The command line and fast boot console is as shown in Figure 2.



|

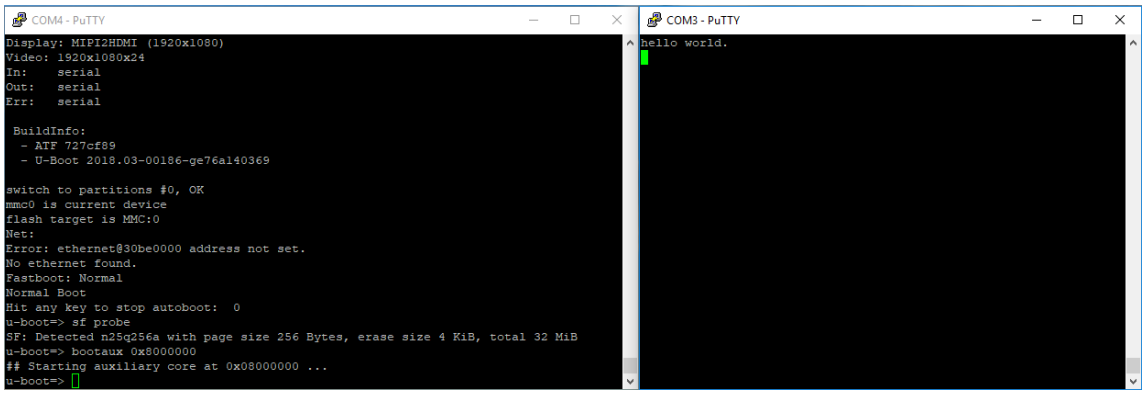
|

6. Then, power off the board and change the boot mode to MicroSD Mode [SW1101:1100[1-4]] , and power on the board again.



7. Use following command in U-Boot to kickoff m7:

```
sf probe
sf read ${loadaddr} 0 4
bootaux 0x8000000
```

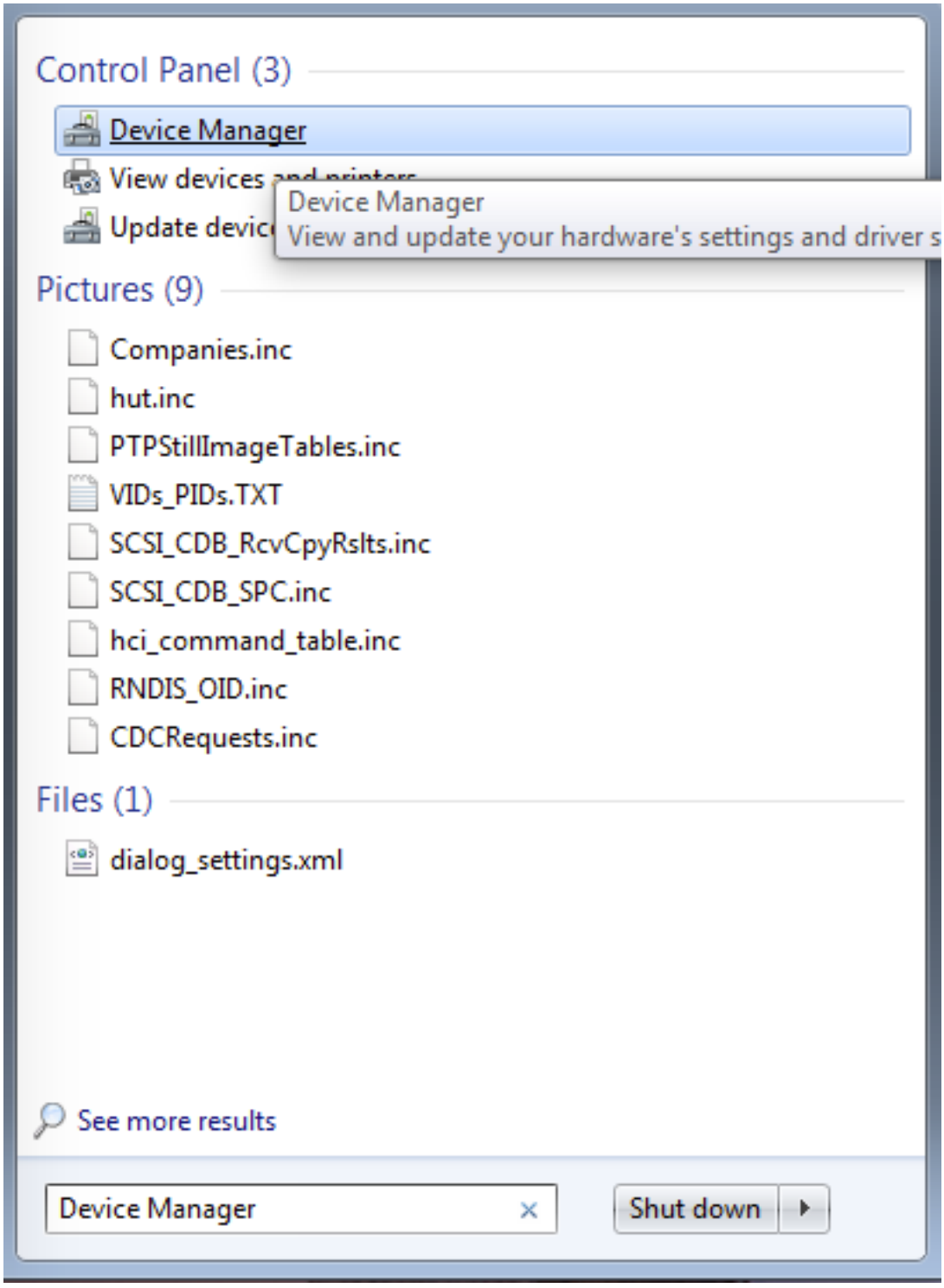


Parent topic:*Run a flash target demo by UUU*

How to determine COM port

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

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



2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports. Depending on the NXP board you're using, the COM port can be named differently.
    1. **USB-UART** interface

|

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

Welcome to the **GitHub Repository SDK Guide**. This documentation provides instructions for setting up and working with the MCUXpresso SDK distributed in a **multi-repository model**. The SDK is distributed across multiple GitHub repositories and managed using the **Zephyr West** tool, enabling modular development and streamlined workflows.

Overview

The GitHub Repository SDK approach offers:

- **Modular Structure**: Multiple repositories for flexibility and scalability.
- **Zephyr West Integration**: Simplified repository management and synchronization.
- **Cross-Platform Support**: Designed for MCUXpresso SDK development environments.

Benefits of the Multi-Repository Approach

- **Scalability**: Easily add or update components without impacting the entire SDK.
- **Collaboration**: Enables distributed development across teams and repositories.
- **Version Control**: Independent versioning for components ensures better stability.
- **Automation**: Zephyr West simplifies dependency handling and repository synchronization.

Setup and Configuration

Follow these steps to prepare your development environment:

Development Tools Installation This guide explains how to install the essential tools for development with the MCUXpresso SDK.

Quick Start: Automated Installation (Recommended) The **MCUXpresso Installer** is the fastest way to get started. It automatically installs all the basic tools you need.

1. **Download the MCUXpresso Installer** from: Dependency-Installation
2. **Run the installer** and select **"MCUXpresso SDK Developer"** from the menu
3. **Click Install** and let it handle everything automatically

Manual Installation If you prefer to install tools manually or need specific versions, follow these steps:

Essential Tools

Git - Version Control **What it does**: Manages code versions and downloads SDK repositories from GitHub.

Installation:

- Visit git-scm.com
- Download for your operating system
- Run installer with default settings
- **Important**: Make sure "Add Git to PATH" is selected during installation

Setup:

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

Python - Scripting Environment **What it does**: Runs build scripts and SDK tools.

Installation:

- Install Python **3.10 or newer** from python.org
- **Important**: Check "Add Python to PATH" during installation

West - SDK Management Tool **What it does**: Manages SDK repositories and provides build commands. The west tool is developed by the Zephyr project for managing multiple repositories.

Installation:

```
pip install -U west
```

Minimum version: 1.2.0 or newer

Build System Tools

CMake - Build Configuration **What it does**: Configures how your projects are built.

Recommended version: 3.30.0 or newer

Installation:

- **Windows**: Download .msi installer from cmake.org/download
- **Linux**: Use package manager or download from cmake.org
- **macOS**: Use Homebrew (brew install cmake) or download from cmake.org

Ninja - Fast Build System **What it does**: Compiles your code quickly.

Minimum version: 1.12.1 or newer

Installation:

- **Windows**: Usually included, or download from ninja-build.org
- **Linux**: sudo apt install ninja-build or download binary
- **macOS**: brew install ninja or download binary

Ruby - IDE Project Generation (Optional) **What it does**: Generates project files for IDEs like IAR and Keil.

When needed: Only if you want to use traditional IDEs instead of VS Code.

Installation: Follow the Ruby environment setup guide

Compiler Toolchains Choose and install the compiler toolchain you want to use:

Toolchain	Best For	Download Link	Environment Variable
ARM GCC (Recommended)	Most users, free	ARM GNU Toolchain	ARMGCC_DIR
IAR EWARM	Professional development	IAR Systems	IAR_DIR
Keil MDK	ARM ecosystem	ARM Developer	MDK_DIR
ARM Compiler	Advanced optimization	ARM Developer	ARMCLANG_DIR

Setting Up Environment Variables After toolchain installation, set an environment variable so the build system locates it:

Windows:

```
# Example for ARM GCC installed in C:\armgcc
setx ARMGCC_DIR "C:\armgcc"
```

Linux/macOS:

```
# Add to ~/.bashrc or ~/.zshrc
export ARMGCC_DIR="/usr"  # or your installation path
```

Verify Your Installation After installation, verify everything works by opening a terminal/command prompt and running these commands:

```
# Check each tool - you should see version numbers
git --version
python --version
west --version
cmake --version
ninja --version
arm-none-eabi-gcc --version  # (if using ARM GCC)
```

Troubleshooting Installation Issues **"Command not found" errors**:

- The tool isn't in your system PATH
- **Solution**: Add the installation directory to your PATH environment variable

Python/pip issues:

- Try using `python3` and `pip3` instead of `python` and `pip`
- On Windows, run the Command Prompt as an Administrator

Slow downloads:

- Add timeout option: `pip install -U west --default-timeout=1000`
- Use alternative mirror: `pip install -U west -i https://pypi.tuna.tsinghua.edu.cn/simple`

GitHub Repository Setup This guide explains how to initialize your MCUXpresso SDK workspace from GitHub repositories using the west tool. The GitHub Repository SDK uses multiple repositories hosted on GitHub to provide modular, flexible development.

Prerequisites Verify the requirements:

System Requirements:

- Python 3.8 or later
- Git 2.25 or later
- CMake 3.20 or later
- Build tools for your target platform

Verification Commands:

```
python --version    # Should show 3.8+
git --version       # Should show 2.25+
cmake --version     # Should show 3.20+
west --version      # Should show west tool installation
```

Workspace Initialization The GitHub Repository SDK uses the Zephyr west tool to manage multiple repositories containing different SDK components.

Step 1: Initialize Workspace Create and initialize your SDK workspace from GitHub:

Get the latest SDK from main branch:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk
```

Get SDK at specific revision:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk --mr {revision}
```

Note: Replace {revision} *with the desired release tag, such as* v25.09.00

Step 2: Choose Your Repository Update Strategy Navigate to the SDK workspace:

```
cd mcuxpresso-sdk
```

The west tool manages multiple GitHub repositories containing different SDK components. You have two options for downloading:

Option A: Download All Repositories (Complete SDK) Download all SDK repositories for comprehensive development:

```
west update
```

This command downloads all the repositories defined in the manifest from GitHub. Initial download takes several minutes and requires ~7 GB of disk space.

Best for:

- Exploring the complete SDK
- Multi-board development projects
- Comprehensive middleware evaluation

Option B: Targeted Repository Download (Recommended) Download only repositories needed for your specific board or device to save time and disk space:

```
# For specific board development
west update_board --set board your_board_name

# For specific device family development
west update_board --set device your_device_name

# List available repositories before downloading
west update_board --set board your_board_name --list-repo
```

Best for:

- Single board development
- Faster setup and reduced disk usage
- Focused development workflows

Examples:

```
# Update only repositories for FRDM-MCXW23 board
west update_board --set board frdmmcxw23

# Update only repositories for MCXW23 device family
west update_board --set device mcxw23
```

Step 3: Verify Installation Confirm successful setup:

```
# Verify workspace structure
ls -la
# Should show: manifests/ and mcuxsdk/ directories

# Test build system
west list_project -p examples/demo_apps/hello_world
# Should display available build configurations
```

Advanced Repository Management The west extension command update_board provides advanced repository management capabilities for optimized workspace setup with GitHub repositories.

Board-Specific Setup Update only repositories required for a specific board:

```
# Update only repositories for specific board, e.g., frdmmcxw23
west update_board --set board frdmmcxw23

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

Device-Specific Setup Update only repositories required for a specific device family:

```
# Update only repositories for specific device, e.g., MCXW235
west update_board --set device mcxw23

# List available repositories for the device family
west update_board --set device mcxw23 --list-repo
```

Custom Configuration For advanced users who want to create custom repository combinations:

```
# Use custom configuration file
west update_board --set custom path/to/custom-config.yml

# Generate custom configuration template
cp manifests/boards/custom.yml.template my-custom-config.yml
```

Benefits of Targeted Setup **Reduced Download Size**

- Download only components needed for your target board or device
- Significantly faster initial setup for focused development
- Typical reduction from 7 GB to 2GB

Optimized Workspace

- Cleaner workspace with relevant components only
- Reduced disk space usage
- Faster repository operations

Flexible Development

- Switch between different board configurations easily
- Maintain separate workspaces for different projects

- Include optional components as needed

Repository Information Before setting up your workspace, you can explore what repositories are available:

```
# Display repository information in console
west update_board --set board frdmmcxw23 --list-repo

# Export repository information to YAML file for reference
west update_board --set board frdmmcxw23 --list-repo -o board-repos.yml
```

This command lists all the available repositories with descriptions and outlines the included components in the workspace.

Package Generation (Optional) The update_board command can also generate ZIP packages for offline distribution:

```
# Generate board-specific SDK package
west update_board --set board frdmmcxw23 -o frdmmcxw23-sdk.zip
```

Note: Package generation is primarily intended for creating custom SDK distributions. For regular development, use the workspace update commands without the -o option.

Workspace Management

Updating Your Workspace Keep your SDK current with latest updates from GitHub:

For Complete SDK Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update all component repositories
cd ..
west update
```

For Targeted Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update board-specific repositories
cd ..
west update_board --set board your_board_name
```

Workspace Status Check workspace synchronization status:

```
# Show status of all repositories
west status

# Show detailed information about repositories
west list
```

Troubleshooting **Network Issues:**

- Use `west update --keep-descendants` for partial failures
- Configure Git credentials for private repositories
- Check firewall settings for Git protocol access

Permission Issues:

- Ensure write permissions in workspace directory
- Run commands without sudo/administrator privileges
- Verify Git SSH key configuration for authenticated access

Disk Space:

- Full SDK workspace requires approximately 7-8 GB
- Targeted workspace typically requires 1-2 GB
- Use board-specific setup to reduce workspace size

Repository Management Issues:

- Verify board/device names match available configurations
- Check that custom YAML files follow the correct template format
- Use `--list-repo` to verify available repositories before setup

Next Steps With your workspace initialized:

1. Review *Workspace Structure* to understand the layout
2. Build your first project with *First Build Guide*
3. Explore *Development Workflows MCUXPresso VSCode* or *Development Workflows Command Line* for the details on project setup and execution

For advanced repository management, see the west tool documentation.

Explore SDK Structure and Content

Learn about the organization of the SDK and its components:

SDK Architecture Overview The MCUXpresso SDK uses a modular architecture where software components are distributed across multiple repositories hosted on GitHub and managed through the west tool. This approach provides flexibility, maintainability, and enables selective component inclusion.

Repository Organization Based on the manifest structure, the SDK consists of four main repository categories:

Manifest Repository The manifest repo (mcuxsdk-manifests) contains the west.yml manifest file that tracks all other repositories in the SDK.

Base Repositories Recorded in submanifests/base.yml and loaded in the root west.yml manifest file. These are the foundational repositories that build the SDK:

- **Devices**: MCU-specific support packages
- **Examples**: Demonstration applications and code samples
- **Boards**: Board support packages

Middleware Repositories Recorded in the submanifests/middleware subdirectory, categorized according to functionality:

- **Connectivity**: Networking stacks, USB, and communication protocols
- **Security**: Cryptographic libraries and secure boot components
- **Wireless**: Bluetooth, IEEE 802.15.4, and other wireless protocols
- **Graphics**: Display drivers and UI frameworks
- **Audio**: Audio processing and voice recognition libraries
- **Machine Learning**: AI inference engines and neural network libraries
- **Safety**: IEC60730B safety libraries
- **Motor Control**: Motor control and real-time control libraries

Internal Repositories Recorded in submanifests/internal.yml and grouped into the "bifrost" group. These are only visible to NXP internal developers and hosted on NXP internal git servers.

Repository Hosting Public repositories are hosted on GitHub under these organizations:

- nxp-mcuxpresso
- NXP
- nxp-zephyr

Internal repositories are hosted on NXP's private Git infrastructure.

Benefits of This Architecture **Selective Integration**: Projects include only required components, reducing memory footprint and build complexity.

Independent Versioning: Each component maintains its own release cycle and version control.

Community Collaboration: Public repositories accept community contributions through standard Git workflows.

Scalable Maintenance: Component owners can update their repositories without affecting the entire SDK.

Workspace Management The west tool manages repository synchronization, version tracking, and workspace updates. All repositories are checked out under the mcuxsdk/ directory with their designated paths defined in the manifest files.

Workspace Structure After you initialize your SDK workspace, it creates a specific directory structure that organizes all SDK components. This structure is identical for both GitHub Repository SDK and Repository-Layout SDK Package.

Top-Level Organization

```
your-sdk-workspace/
  manifests/        # West manifest repository
  mcuxsdk/          # Main SDK content
```

The mcuxsdk/ directory serves as your primary working directory and contains all the SDK components.

SDK Component Layout Based on the actual SDK structure, the main directories include:

Directory	Contents	Purpose
arch/	Architecture-specific files	ARM CMSIS, build configurations
cmake	Build system modules	CMake configuration and build rules
compo	Software components	Reusable software libraries and utilities
devices	Device support packages	MCU-specific headers, startup code, linker scripts
drivers	Peripheral drivers	Hardware abstraction layer for MCU peripherals
examp	Sample applications	Demonstration code and reference implementations
middle	Optional software stacks	Networking, graphics, security, and other libraries
rtos/	Operating system support	FreeRTOS integration
scripts	Build and utility scripts	West extensions and development tools
svd	Svd files for devices, this is optional because of large size. Customers run west manifest config group.filter +optional and west update mcux-soc-svd to get this folder.	

Example Organization Examples follow a two-tier structure separating common code from board-specific implementations:

Common Example Files

```
examples/demo_apps/hello_world/
  CMakeLists.txt     # Build configuration
  example.yml        # Example metadata
  hello_world.c      # Application source code
  Kconfig            # Configuration options
  readme.md          # General documentation
```

Board-Specific Files

```
examples/_boards/your_board/demo_apps/hello_world/
   app.h                    # Board specific application header
   example_board_readme.md    # Board specific documentation
   hardware_init.c            # Board specific hardware initialization
   pin_mux.c                  # Pin multiplexing configuration
   pin_mux.h                  # Pin multiplexing header definitions
   hello_world.bin            # Pre-built binary for quick testing
   hello_world.mex             # MCUXpresso Config Tools project file
   prj.conf                 # Board specific Kconfig configuration
   reconfig.cmake             # Board specific cmake configuration overrides
```

Device Support Structure Device support is organized hierarchically by MCU family:

```
devices/
   MCX/                  # MCU portfolio
      MCXW/               # MCU family
         MCXW235/       # Specific device
            MCXW235.h          # Device register definitions
            drivers/           # Device-specific drivers
            gcc/               # GNU toolchain files
            iar/               # IAR toolchain files
            mcuxpresso/        # MCUXpresso IDE files
            startup_MCXW235.c # Startup and vector table
            system_MCXW235.c  # System initialization
```

Middleware Organization Middleware components are categorized by functionality and maintained in separate repositories. Based on the manifest files, common middleware categories include:

- **Connectivity**: USB, TCP/IP, industrial protocols
- **Security**: Cryptographic libraries, secure boot
- **Wireless**: Bluetooth, IEEE 802.15.4, Wi-Fi
- **Graphics**: Display drivers, UI frameworks
- **Audio**: Processing libraries, voice recognition
- **Machine Learning**: Inference engines, neural networks
- **Safety**: IEC60730B safety libraries
- **Motor Control**: Motor control and real-time control libraries

Documentation Structure SDK documentation is distributed across multiple locations:

- docs/ - Core SDK documentation and build infrastructure
- Component repositories - API documentation and integration guides
- Board directories - Hardware-specific setup instructions

For complete documentation, refer to the online documentation.

Understanding Example Structure Each example has **two README files**:

1. **General README:** examples/demo_apps/hello_world/readme.md
   - What the example does
   - General functionality description
   - Common usage information

2. **Board-Specific README:** examples/_boards/{board_name}/demo_apps/hello_world/example_board_readme.md
   - Board-specific setup instructions
   - Hardware connections required
   - Board-specific behavior notes

Tip: Always check both readme files - start with the general one, then read the board-specific one for detailed setup.

Development Workflows

Get started with building and running projects:

Building Your First Project This guide explains how to build and run your first SDK example project using the west build system. This applies to both GitHub Repository SDK and Repository-Layout SDK Package.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development board connected via USB
- Build tools installed per *Installation Guide*

Understanding Board Support Use the west extension to discover available examples for your board:

```
west list_project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

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

Basic Build Process

Simple Build Build the hello_world example with default settings:

```
west build -b your_board examples/demo_apps/hello_world
```

The default toolchain is armgcc, and the build system will select the first debug target as default if no config is specified.

Specifying Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release

# Debug build (default)
west build -b your_board examples/demo_apps/hello_world --config debug
```

Alternative Toolchains

```
# IAR toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar

# Other toolchains as supported by the example
```

Multicore Applications For multicore devices, specify the core ID:

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

For multicore projects using sysbuild:

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

Flash an Application Flash the built application to your board:

```
west flash -r linkserver
```

Debug Start a debug session:

```
west debug -r linkserver
```

Common Build Options

Clean Build Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run See the commands that get executed without running them:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device DEVICE_PART_NUMBER --config␣
↪release
```

Project Configuration

CMake Configuration Only Run configuration without building:

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

Interactive Configuration Launch the configuration GUI:

```
west build -t guiconfig
```

Troubleshooting

Build Failures Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Getting Help View the help information for west build:

```
west build -h
```

Check Supported Configurations To see available configuration options and board targets for an example, refer to the below command:

```
west list_project -p examples/demo_apps/hello_world
```

Next Steps

- Explore other examples in the SDK
- Learn about *Command Line Development* for advanced options
- Try *VS Code Development* for integrated development
- Refer *Workspace Structure* to understand the SDK layout

MCUXpresso for VS Code Development This guide covers using MCUXpresso for VS Code extension to build, debug, and develop SDK applications with an integrated development environment.

Prerequisites

- SDK workspace initialized (GitHub Repository SDK or Repository-Layout SDK Package)
- Development tools installed per *Installation Guide*
- Visual Studio Code installed
- MCUXpresso for VS Code extension installed

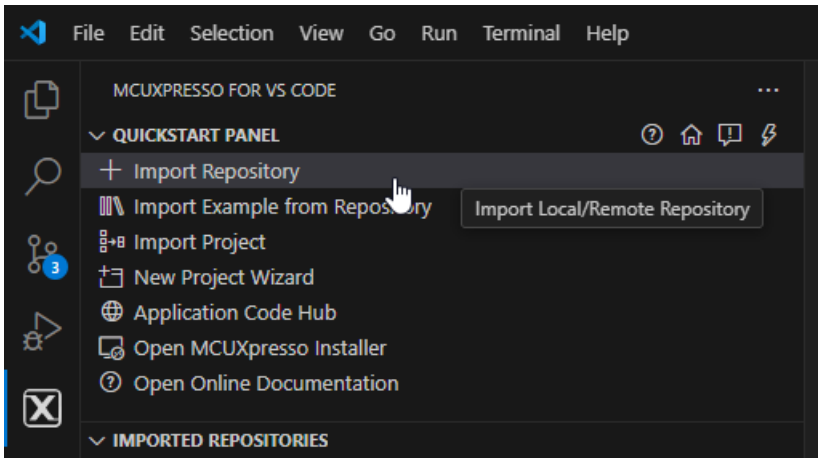
Extension Installation

Install MCUXpresso for VS Code The MCUXpresso for VS Code extension provides integrated development capabilities for MCUXpresso SDK projects. Refer to the MCUXpresso for VS Code Wiki for detailed installation and setup instructions.

SDK Import and Setup

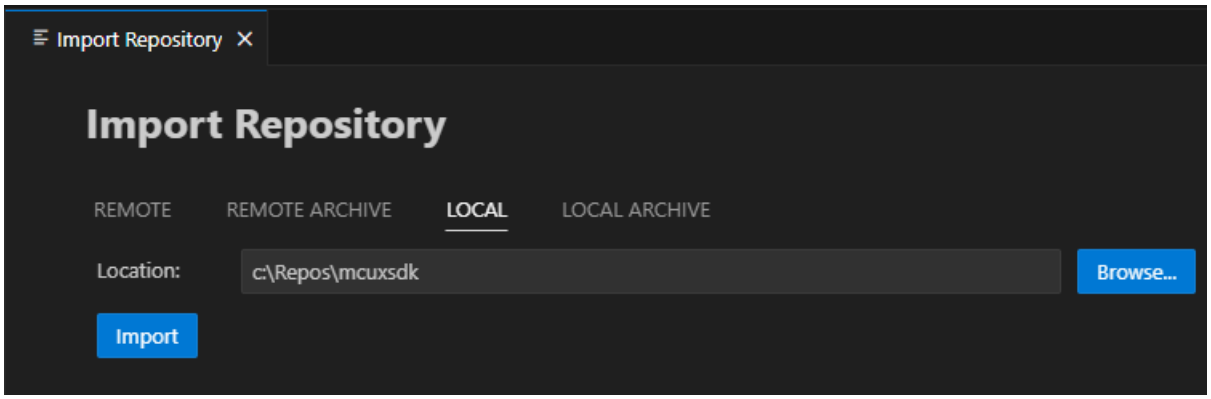
Import Methods The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

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

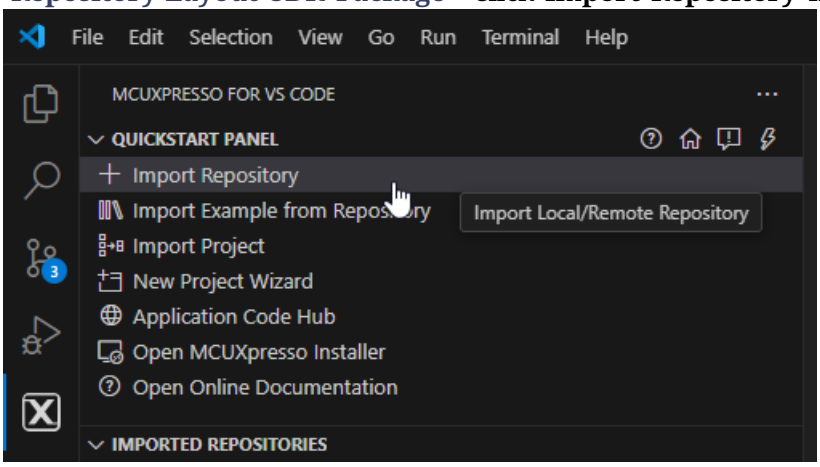


Note: You can import the SDK in several ways. Refer to MCUXpresso for VS Code Wiki for details.

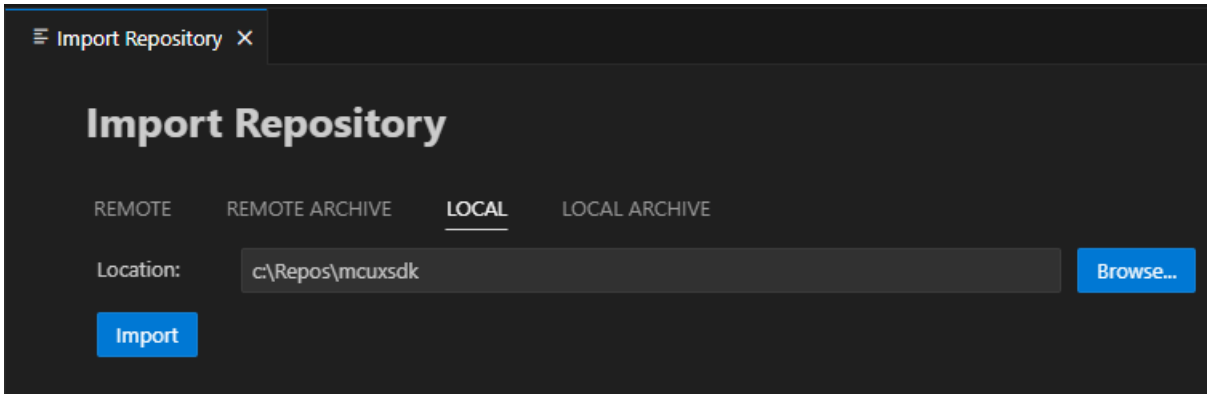
Select **Local** if you've already obtained the SDK according to *setting up the repo*. Select your location and click **Import**.



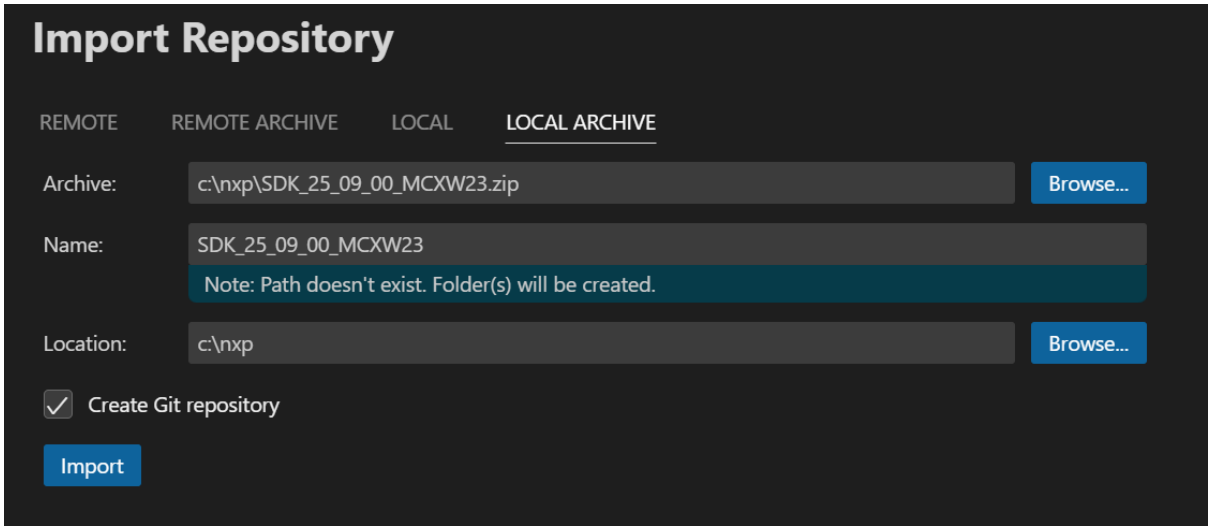
Import Repository-Layout SDK Package Click **Import Repository** from the **QUICKSTART PANEL**



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



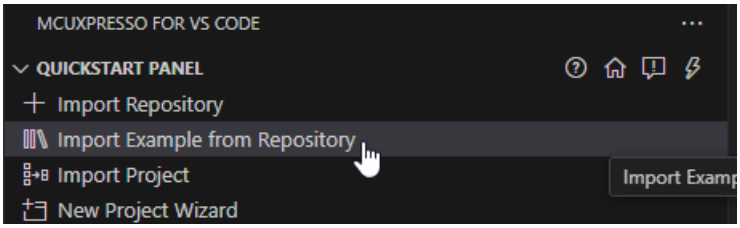
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



Building Example Applications

Import Example Project

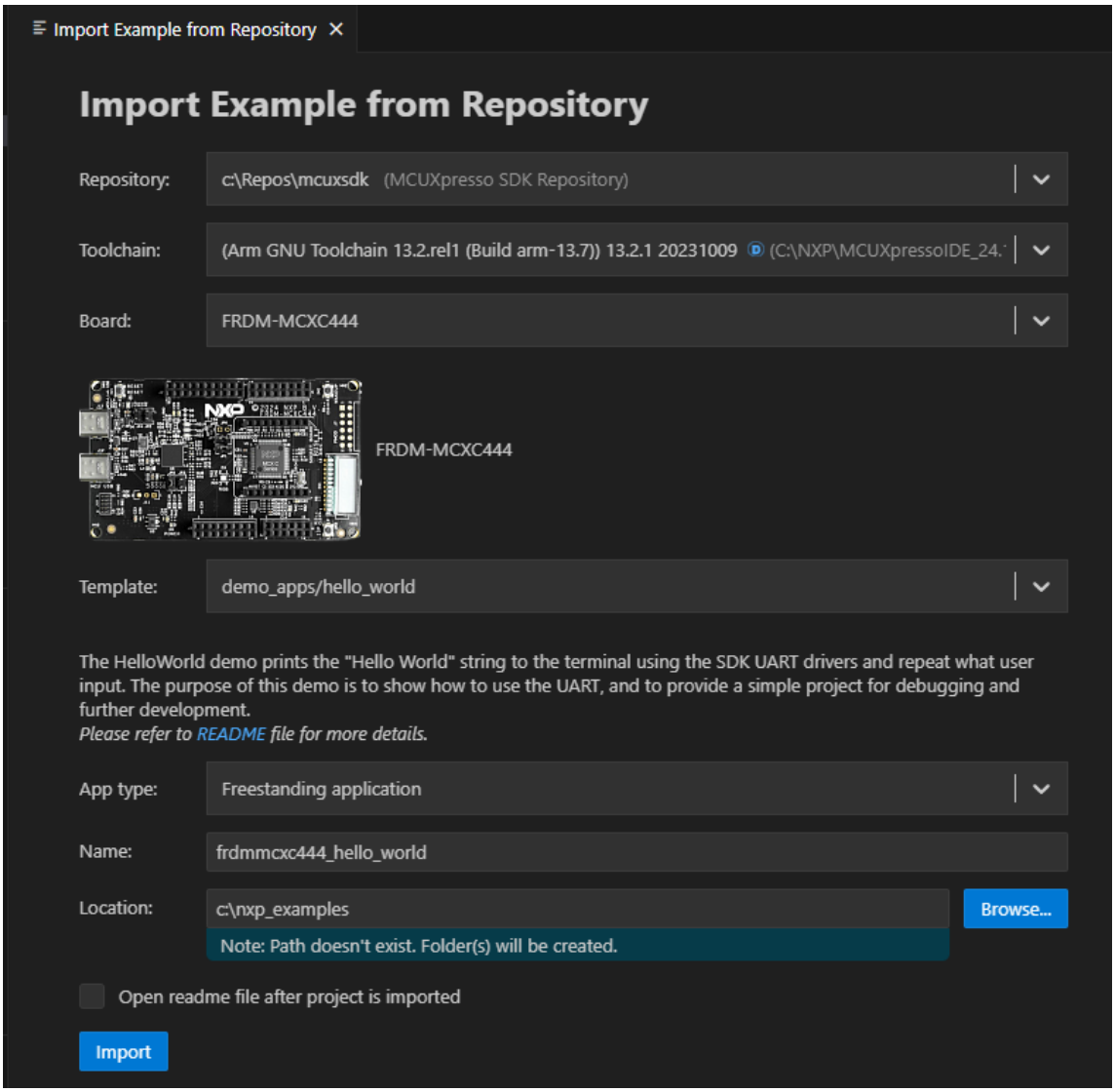
1. Click **Import Example from Repository** from the **QUICKSTART PANEL**



2. Configure project settings:
   - **MCUXpresso SDK**: Select your imported SDK
   - **Arm GNU Toolchain**: Choose toolchain
   - **Board**: Select your target development board
   - **Template**: Choose example category

- **Application**: Select specific example (e.g., hello_world)
- **App type**: Choose between Repository applications or Freestanding applications

3. Click **Import**



Application Types **Repository Applications:**

- Located inside the MCUXpresso SDK
- Integrated with SDK workspace

Freestanding Applications:

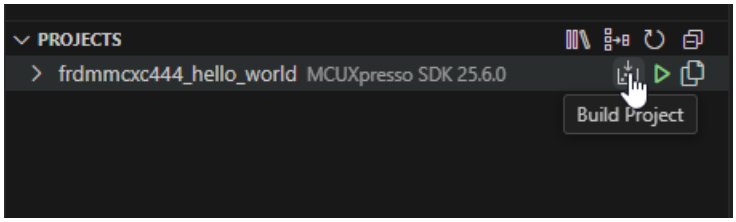
- Imported to user-defined location
- Independent of SDK location

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

Building Projects

Build Process

1. Navigate to **PROJECTS** view
2. Find your project
3. Click the **Build Project** icon

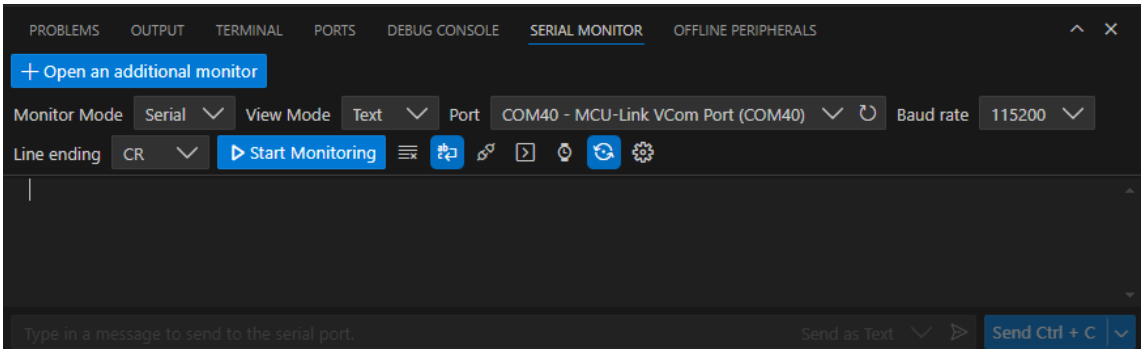


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

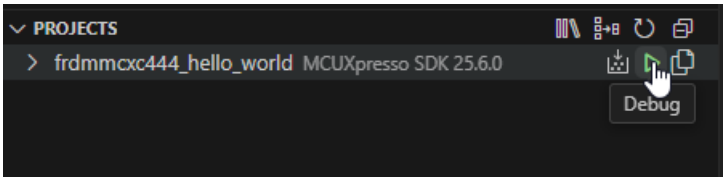
1. Open **Serial Monitor** from VS Code's integrated terminal



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

Debug Session

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

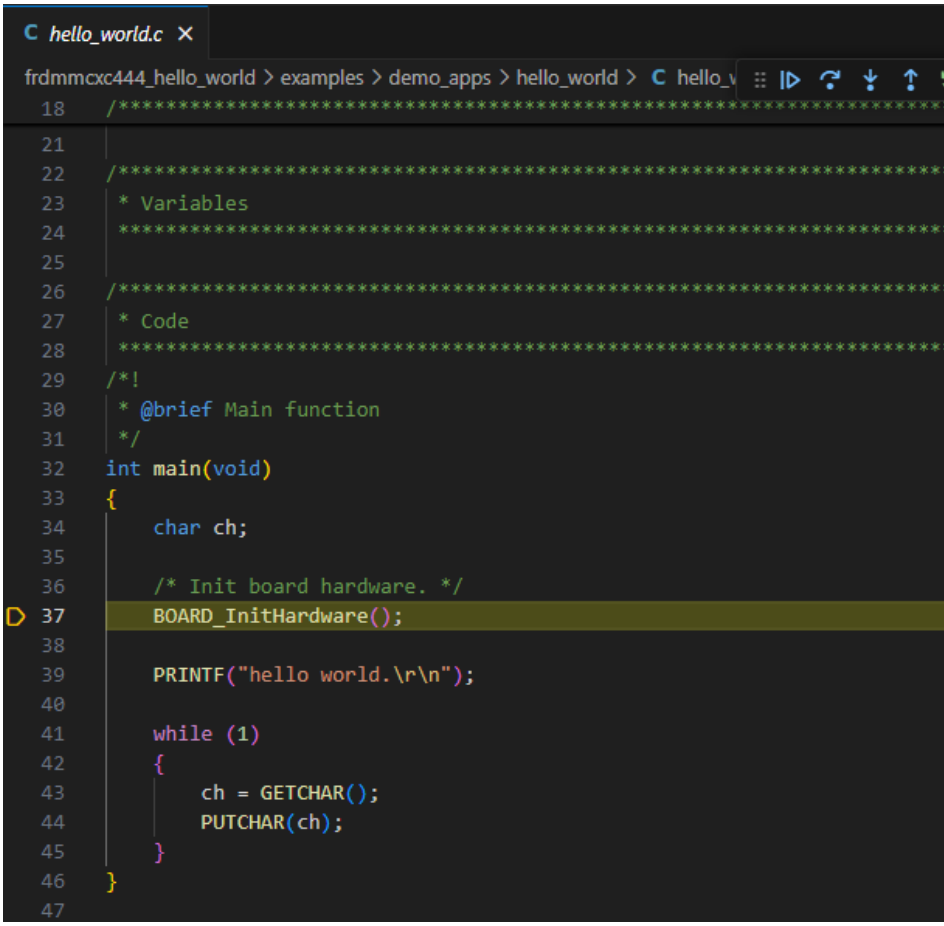


The debug session will begin with debug controls initially at the top of the interface.

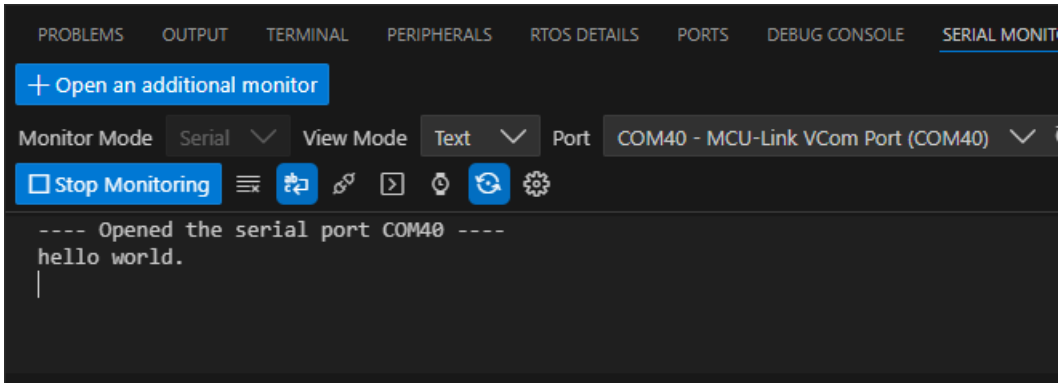
Debug Controls Use the debug controls to manage execution:

- **Continue**: Resume code execution
- **Step controls**: Navigate through code

- **Stop**: Terminate debug session



Monitor Output Observe application output in the **Serial Monitor** to verify correct operation.



Debug Probe Support For comprehensive information on debug probe support and configuration, refer to the MCUXpresso for VS Code Wiki DebugK section.

Project Configuration

Workspace Management The extension integrates with the MCUXpresso SDK workspace structure, providing access to:

- Example applications
- Board configurations

- Middleware components
- Build system integration

Multi-Project Support The PROJECTS view allows management of multiple imported projects within the same workspace.

Troubleshooting

Import Issues **SDK not detected:**

- Verify SDK workspace is properly initialized
- Ensure all required repositories are updated
- Check SDK manifest files are present

Project import failures:

- Confirm board support exists for selected example
- Verify toolchain installation
- Check example compatibility with selected board

Build Problems **Build failures:**

- Check integrated terminal for error messages
- Verify all dependencies are installed
- Ensure toolchain is properly configured

Debug Issues **Debug session fails:**

- Verify board connection via USB
- Check debug probe drivers are installed
- Confirm build completed successfully

Serial monitor problems:

- Verify correct VCom port selection
- Check baud rate configuration (115200)
- Ensure board drivers are installed

Integration with Command Line MCUXpresso for VS Code integrates with the underlying west build system, allowing seamless integration with command line workflows described in *Command Line Development*.

Advanced Features

Project Types The extension supports both repository-based and freestanding project types, providing flexibility in project organization and SDK integration.

Build System Integration The extension leverages the MCUXpresso SDK build system, providing access to all build configurations and options available through command line tools.

Next Steps

- Explore additional examples in the SDK
- Review *Command Line Development* for advanced build options
- Refer MCUXpresso for VS Code Wiki for detailed documentation
- Learn about *SDK Architecture* for better understanding of the development environment

Command Line Development This guide covers developing with the MCUXpresso SDK using command line tools and the west build system. This workflow applies to both GitHub Repository SDK and Repository-Layout SDK Package distributions.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development tools installed per *Installation Guide*
- Target board connected via USB

Understanding Board Support Use the west extension to discover available examples for your board:

```
west list_project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

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

Basic Build Commands

Standard Build Process Build with default settings (armgcc toolchain, first debug config):

```
west build -b your_board examples/demo_apps/hello_world
```

Specifying Build Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release

# Debug build with specific toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar --config debug
```

Multicore Applications For multicore devices, specify the core ID:

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

For multicore projects using sysbuild:

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

Shield Support For boards with shields:

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

Advanced Build Options

Clean Builds Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run See what commands would be executed:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device MK22F12810 --config release
```

Project Configuration

CMake Configuration Only Run configuration without building:

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

Interactive Configuration Launch the configuration GUI:

```
west build -t guiconfig
```

Flashing and Debugging

Flash Application Flash the built application to your board:

```
west flash -r linkserver
```

Debug Session Start a debugging session:

```
west debug -r linkserver
```

IDE Project Generation Generate IDE project files for traditional IDEs:

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

IDE project files are generated in mcuxsdk/build/<toolchain> folder.

Note: Ruby installation is required for IDE project generation. See *Installation Guide* for setup instructions.

Troubleshooting

Build Failures Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Toolchain Issues Verify environment variables are set correctly:

```
# Check ARM GCC
echo $ARMGCC_DIR
arm-none-eabi-gcc --version

# Check IAR (if using)
echo $IAR_DIR
```

Getting Help Display help information:

```
west build -h
west flash -h
west debug -h
```

Check Supported Configurations If unsure about supported options for an example:

```
west list_project -p examples/demo_apps/hello_world
```

Best Practices

Project Organization

- Keep custom projects outside the SDK tree
- Use version control for your application code
- Document any SDK modifications

Build Efficiency

- Use -p always for clean builds when troubleshooting
- Leverage --dry-run to understand build processes
- Use specific configs and toolchains to reduce build time

Development Workflow

1. Start with existing examples closest to your requirements
2. Copy and modify rather than building from scratch
3. Test with hello_world before moving to complex examples
4. Use configuration tools for pin muxing and clock setup

Next Steps

- Explore *VS Code Development* for integrated development experience
- Review *Workspace Structure* to understand SDK organization
- Refer build system documentation for advanced configurations

Using MCUXpresso Config Tools MCUXpresso Config tools provide a user-friendly way to configure hardware initialization of your projects. This guide explains the basic workflow with the MCUXpresso SDK west build system and the Config Tools.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- MCUXpresso Config Tools standalone installed (version 25.09 or above)
- MCUXpresso SDK Project that can be successfully built

Board Files MCUXpresso Config Tools generate source files for the board. These files include pin_mux.c/h and clock_config.c/h. The files contain initialization code functions that reflect the hardware configuration in the Config Tools. Within the SDK codebase, these files are specific for the board and either shared by multiple example projects or specific for one example. Open or import the configuration from the SDK project in the Config Tools and customize the settings to match the custom board or specific project use case and regenerate the code. See *User Guide for MCUXpresso Config Tools (Desktop)* (document GSMCUXCTUG) for details.

Note: When opening the configuration for SDK example projects, the board files may be shared across multiple examples. To ensure a separate copy of the board configuration files exists, create a freestanding project with copied board files.

Visual Studio Code To open the configuration in Visual Studio Code, use the context menu for the project to access Config Tools. See MCUXpresso Extension Documentation for details. Otherwise, use the manual workflow described in detail in the following section.

Manual Workflow Use the following steps:

1. Before using Config Tools, run the west command to get the project information for Config Tools from the SDK project files, for example:

```
west cfg_project_info -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_
↪id=cm33_core0
```

   This results in the creation of the project information json file that is searched by the config tools when the configuration is created. The parameters of the command should match the build parameters that will be used for the project.

2. Launch the MCUXpresso Config Tools and in the **Start development** wizard, select **Create a new configuration based on the existing IDE/Toolchain project**. Select the created "cfg_tools" subfolder as a project folder (for example: ...mcuxsdk/examples/demo_apps/hello_world/cfg_tools/).

Updating the SDK West project **Note:** Updating project is supported with Config Tools V25.12 or newer only.

Changes in the Config tools generated source code modules may require adjustments to the toolchain project to ensure a successful build. These changes may mean, for example, adding the newly generated files, adding include paths, required drivers, or other SDK components. This section describes how to manually resolve the changes needed in the project within the toolchain projects based on the SDK project managed by the West tool.

After the configuration in the Config Tools is finished, write updated files to the disk using the 'Update Code' command. The written files include a json file with the required changes for the toolchain project.

To resolve the changes in the project in the terminal, launch the west command that updates the project. For example:

```
west cfg_resolve -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_id=cm33_core0
```

This command updates the appropriate cmake and kconfig files to address the changes. After this, the application can be built.

Note: The cfg_resolve command supports additional arguments. Launch the *west cfg_resolve -h* command to get the list and description.

1.4 Release Notes

1.4.1 MCUXpresso SDK Release Notes

Overview

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

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

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

MCUXpresso SDK

As part of the MCUXpresso software and tools, MCUXpresso SDK is the evolution of Kinetis SDK, includes support for LPC, DSC,PN76, and i.MX System-on-Chip (SoC). The same drivers, APIs, and

middleware are still available with support for Kinetis, LPC, DSC, and i.MX silicon. The MCUXpresso SDK adds support for the MCUXpresso IDE, an Eclipse-based toolchain that works with all MCUXpresso SDKs. Easily import your SDK into the new toolchain to access to all of the available components, examples, and demos for your target silicon. In addition to the MCUXpresso IDE, support for the MCUXpresso Config Tools allows easy cloning of existing SDK examples and demos, allowing users to leverage the existing software examples provided by the SDK for their own projects.

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

Development tools

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

- IAR Embedded Workbench for Arm, version is 9.60.4
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

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

Development boards	MCU devices			
EVK-MIMX8	MIMX8MN6CVTIZ, MIMX8MN1DVPIZ, MIMX8MN3CVPIZ, MIMX8MN4CVTIZ, MIMX8MN5DVPIZ,	**MIMX8MN6DVTJZ**, MIMX8MN1DVTJZ, MIMX8MN3CVTIZ, MIMX8MN4DVTJZ, MIMX8MN5DVTJZ	MIMX8MN1CVPIZ, MIMX8MN2CVTIZ, MIMX8MN3DVPIZ, MIMX8MN5CVPIZ,	MIMX8MN1CVTIZ, MIMX8MN2DVTJZ, MIMX8MN3DVTJZ, MIMX8MN5CVTIZ,

MCUXpresso SDK release package

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

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

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

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

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Middleware

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

TinyCBOR Concise Binary Object Representation (CBOR) Library

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

Multicore Multicore Software Development Kit

llhttp HTTP parser llhttp

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

Release contents

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

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

Known issues

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

Cannot add SDK components into FreeRTOS projects

It is not possible to add any SDK components into FreeRTOS project using the MCUXpresso IDE New Project wizard.

The freertos_lpuart example does not complete successfully

The example hangs after console output 'FreeRTOS LPUART driver example'.

Examples: freertos_lpuart

Affected toolchains: All

The example does not perform as expected (Ticks do not printed on the console or the application does not wake up from the sleep mode).

Examples: freertos_tickless

Affected toolchains: All

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

---

eASRC

[2.0.7]

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

[2.0.6]

- Improvements
  - Updated use of definitions in the function ASRC_SetSlotConfig and ASRC_EnableContextSlot.

[2.0.5]

- Improvements
  - Fixed the typo and multichannel configuration issues in asrc driver.
  - Updated the asrc firmware table for better THD performance.

[2.0.4]

- Improvements
  - Corrected the data width support by IEC60958,
  - Updated the default stage1 result to float to avoid convert result distortion.

[2.0.3]

- Bug Fixes
  - Corrected the btis shift index used in function ASRC_TransferBlocking.
  - Ensured the resampler and prefilter are not in bypass mode when the convertion require the function.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 14.3, 14.4, 16.3, 16.1, 16.8, 17.7.

[2.0.1]

- Bug Fixes
  - Fixed the context id hard code issue in the function ASRC_TransferInCreateHandleSDMA/ASRC_TransferOutCreateHandleSDMA.
- Improvements
  - Added support for the data size bigger than 64K in sdma driver.

[2.0.0]

- Initial version.

ASRC_SDMA

[2.0.4]

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

[2.0.3]

- Refer asrc driver change log 2.0.0 to 2.0.3

CACHE ARMv7-M7

[2.0.5]

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

[2.0.4]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

COMMON

[2.6.3]

- Bug Fixes
  - Fixed build issue of CMSIS PACK BSP example caused by CMSIS 6.1 issue.

[2.6.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule for implicit conversions in boolean contexts

[2.6.1]

- Improvements
  - Support Cortex M23.

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.3]

- New Features
  - Added NETC into status group.

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

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

[2.3.0]

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

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Other Change
  - Added MECC status group definition.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.4]

- New Features
  - Added OTFAD into status group.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

ECSPI

[2.3.3]

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

[2.3.2]

- Improvements
  - Changed ECSPI_DUMMYDATA to 0x00.

[2.3.1]

- Bug Fixes
  - Fixed ECSPI_GetInstance potential issue that return wrong instance number.

[2.3.0]

- Bug Fixes
  - Fixed burst length issue,the burst length range shall range from 1-4096 bits, so the width shall be uint8_t rather than uint16_t.

[2.2.0]

- Bug Fixes
  - Removed the useless channel configuration of waveform, since the waveform can not be configured when not using the exchange bit(ECSPIx_CONREG[XCH]) for the transfer.
  - Fixed violations of MISRA C-2012 rules: 10.1, 11.9, 8.4.

[2.1.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.3, 10.4, 11.9, 14.4, 15.7, 17.7.

[2.1.0]

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.

[2.0.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.3, 10.4

[2.0.1]

- Bug Fixes
  - Memset local variable SDMA transfer configuration structure to make sure unused members in structure are cleared.
  - Fixed sign-compare warning in ECSPI_SendTransfer.

[2.0.0]

- Initial version.

ENET

[2.11.0]

- New Features
  - Added function ENET_Ptp1588JumpTimer which adjusts the ENET PTP 1588 timer by jumping a relative time difference. Compared to ENET_Ptp1588SetTimer, this function yields more accurate results when the relative time difference between the PTP clock and the target clock is known.

[2.10.1]

- Bug Fixes
  - Fixed WAKEUP interrupt not being handled.

[2.10.0]

- New Features
  - Added function ENET_Ptp1588GetChannelCaptureValue to read last captured time from PTP 1588 timer.

[2.9.3]

- Bug Fixes
  - Fixed ENET_Ptp1588GetTimer incorrect timestamps when timer wraps occur after nanosecond capture:
    - Now only increments second field when nanosecond value is less than half a second

[2.9.2]

- Bug Fixes
  - RGMII mode is (temporarily) disabled before selecting between 10/100-Mbit/s and 1000-Mbit/s modes of operation. The bit RGMII_EN of RCR register must not be set while changing ECR register's speed bit, otherwise there is a possibility of ENET IP ending in an incorrect state.

[2.9.1]

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

[2.9.0]

- Bug Fixes
  - Enabled collection of transfer statistics, so the function ENET_GetStatistics does not always return zeroes.
- New Features
  - Added new function ENET_EnableStatistics to enable/disable collection of transfer statistics.
  - Added new function ENET_ResetStatistics to reset transfer statistics.
- Improvements
  - Renamed the function ENET_ResetHareware to ENET_ResetHardware.

[2.8.0]

- New Features
  - Added the function to reset hardware on certain devices.

[2.7.1]

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

[2.7.0]

- Improvements
  - Deleted deprecated zero copy Tx/Rx functions and set callback function which can be configured in ENET_Init.
  - Moved the Rx zero copy buffer allocation to Rx BD initialization function to reduce unnecessary looping code.
- Bug Fixes
  - Fixed the issue that predefined Rx buffers which should not be used when enabling Rx zero copy are still be handled by cache operation, it causes hardfault on some platforms.
  - Fixed the issue that zero-copy Rx function doesn't check Rx length of 0 in the BD with EMPTY bit is 0, it may occur in the corner case reported by customer. Not sure how it turns out, consider it as an ENET IP issue and drop this abnormal BD.

[2.6.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 11.6.

[2.6.2]

- Improvements
  - Changed ENET1_MAC0_Rx_Tx_Done0_DriverIRQHandler/ENET1_MAC0_Rx_Tx_Done1_DriverIRQHan to ENET1_MAC0_Rx_Tx_Done1_DriverIRQHandler/ENET1_MAC0_Rx_Tx_Done2_DriverIRQHandler which represent ring 1 and ring 2.

[2.6.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 10.7, 11.6, 11.8.

[2.6.0]

- Improvements
  - Added MDIO access wrapper APIs for ease of use.
  - Fixed the build warning introduced by 64-bit compatibility patch.

[2.5.4]

- Improvements
  - Made the driver compatible with 64-bit platforms.

[2.5.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 11.6.

[2.5.2]

- Improvements
  - Updated the TXIC/RXIC register handling code according to the new header file.

[2.5.1]

- Bug Fixes
  - Fixed document typo.

[2.5.0]

- Bug Fixes
  - Fixed the SendFrame/SendFrameZeroCopy functions issue with scattered buffers.
  - Updated the formula of MDC calculation.
  - Used a feature macro to distinguish the old IP design from the new design, because old IP design always reads a value zero from ATCR->CAPTURE bit. For old IP, driver caculates and wait the necessary delay cycles after setting ATCR->CAPTURE then gets the timestamp value.
- New Features
  - Added new zero copy Tx/Rx function.

  - New zero copy Tx function combines scattered and contiguous Tx buffer in one API, it also supports more Tx featrues which buffer descriptor supports but previous Tx function doesn't support.
  - New zero copy Rx function use dynamic buffer mechanism and simpler interface.
- Improvements
  - Corrected the interrupt handler for PTP timestamp IRQ and PTP1588 event IRQ since platform difference.
  - Added missing IRQ handlers for PTP1588 events on some platforms.
  - Corrected the max Tx frame length verification, it will not depend on a fixed macro. The ENET_FRAME_MAX_FRAMELEN is only an default value for driver, application can configure it. Driver caculates the limitation with the max frame length in register which may takes extended 4 or 8 bytes VLAN tag if VLAN/SVLAN enables.
  - Deleted deprecated Clause 45 read/write legacy APIs.

[2.4.3]

- Improvements
  - Aligned the IRQ handler name with header file.

[2.4.2]

- Bug Fixes
  - Fixed the MISRA issue of speculative out-of-bounds access.

[2.4.1]

- Bug Fixes
  - Fixed the PTP time capture issue.

[2.4.0]

- Improvements
  - Exposed API ENET_ReclaimTxDescriptor for user application to relaim tx descriptors in their application.
  - Added counter to record multicast hash conflict in struct _enet_handle, improved the situation that one multicast group could be left by other conflict multicast address left operation.
  - Improved concurrent usage of relaim and send frame operation.

[2.3.4]

- Bug Fixes
  - Fixed the issue that interrupt handler only checks the interrupt event flag but not checks interrupt mask flag.

[2.3.3]

- Bug Fixes
  - Fixed the issue that some compilers may choose the memcpy with 4-bit aligned address limitation due to the type of address pointer is 'unsigned int *', the data address doesn't have to be 4-bit aligned.

[2.3.2]

- New Features
  - Added the feature that ENET driver can be used in the platform which integrates both 10/100M and 1G ENET IP.
  - Deleted duplicated code about ARM errata 838869 in first/second level IRQ handler.

[2.3.1]

- Improvements
  - Added function pointer checking in IRQ handler to make sure code can be used even it runs into the interrupt when the second level interupt handler is NULL.

[2.3.0]

- Bug Fixes
  - Fixed the issue that clause 45 MDIO read/write API doesn't check the transmission over status between two transmissions.
  - Fixed violations of the MISRA C-2012 rules 2.2,10.3,10.4,10.7,11.6,11.8,13.5,14.4,15.7,17.7.
- New Features
  - Added APIs to support send/receive frame with Zero-Copy.
- Improvements
  - Separated the clock configuration from module configuration when init and deinit.
  - Added functions to set second level interrupt handler.
  - Provided new function to get 1588 timer count without disabling interrupt.
  - Improved timestamp controlling, deleted all old timestamp management APIs and data structures.
  - Merged the single/multiple ring(s) APIs, now these APIs can handle both.
  - Used base and index to control buffer descriptor, aligned with qos and lpc enet driver.

[2.2.6]

- Bug Fixes
  - Updated MII speed formula referring to the manual.

[2.2.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.7, 11.6, 11.9, 13.5, 14.4, 16.4, 17.7, 21.15, 3.1, 8.4.
  - Changed to use ARRAY_SIZE(s_enetBases) as the array size for s_ENETHandle, fixed the hardfault issue for using some ENET instance when ARRAY_SIZE(s_enetBases) is not same as FSL_FEATURE_SOC_ENET_COUNT.

[2.2.4]

- Improvements
  - Added call to Data Synchronization Barrier instruction before activating Tx/Rx buffer descriptor to ensure previous data update is completed.
  - Improved ENET_TransmitIRQHandler to store timestamps for multiple transmit buffer descriptors.
  - Bug Fixes
  - Fixed the issue that ENET_Ptp1588GetTimer did not handle the timer wrap situation.

[2.2.3]

- Improvements
  - Improved data buffer cache maintenance in the ENET driver.

[2.2.2]

- New Features
  - Added APIs for extended multi-ring support.
  - Added the AVB configure API for extended AVB feature support.

[2.2.1]

- Improvements
  - Changed the input data pointer attribute to const in ENET_SendFrame().

[2.1.1]

- New Features
  - Added the extended MDIO IEEE802.3 Clause 45 MDIO format SMI command APIs.
  - Added the extended interrupt coalescing feature.
- Improvements
  - Combined all storage operations in the ENET_Init to ENET_SetHandler API.

[2.0.1]

- Bug Fixes
  - Used direct transmit busy check when doing data transmit.
- Miscellaneous Changes
  - Updated IRQ handler work flow.
  - Changed the TX/RX interrupt macro from kENET_RxByteInterrupt to kENET_RxBufferInterrupt, from kENET_TxByteInterrupt to kENET_TxBufferInterrupt.
  - Deleted unnecessary parameters in ENET handler.

[2.0.0]

- Initial version.

GPC

[2.2.0]

- Improvements
  - Optimized the exited APIs to support extended IRQs and slots.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules.

[2.1.0]

- Improvements
  - Unified the register or bit fields' name which contains specific cortex M core information.
- Bug Fixes
  - Fixed Coverity Out-of-bounds issue.

[2.0.1]

- Improvements
  - Added parameters to enable/disable the WFI and DSM mask.

[2.0.0]

- Initial version.

GPIO

[2.0.7]

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

[2.0.6]

- Bug Fixes
  - Fixed compile warning: 'GPIO_GetInstance' defined but not used when macro FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is defined.

[2.0.5]

- Bug Fixes
  - Fixed MISRA C-2012 issue: rule-17.7.

[2.0.4]

- Improvements
  - Updated the GPIO_PinWrite to use atomic operation if possible.
- Bug Fixes
  - Fixed GPIO_PortToggle bug with platforms don't have register DR_TOGGLE.

[2.0.3]

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

[2.0.2]

- Bug Fixes
  - Fixed the bug of enabling wrong GPIO clock gate in initial API. Since some GPIO instances may not have a clock gate enabled, it checks the clock gate number and makes sure the clock gate is valid.

[2.0.1]

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

[2.0.0]

- Initial version.

GPT

[2.0.6]

- Bug Fixes
  - Fix CERT INT30-C issues.

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.8, 17.7.

[2.0.0]

- Initial version.

I2C

[2.0.7]

- Bug Fixes
  - Fixed MISRA issues.
    - Fixed rules 8.4, 8.5.

[2.0.6]

- Bug Fixes
  - Fixed the bug that, in I2C_MasterStop after the stop command is issued, the IBB flag should be cleared rather than set.
  - Fixed the bug that to clear kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag, their bits should be written '0' rather than '1'.

[2.0.5]

- Bug Fixes
  - Fixed Coverity issue of unchecked return value in I2C_RTOS_Transfer.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 11.9, 14.4, 15.7, 16.4, 17.7.
- Improvements
  - Updated the I2C_WAIT_TIMEOUT macro to unified name I2C_RETRY_TIMES.

[2.0.4]

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

[2.0.3]

- Improvements
  - Improved code readability, added new static API I2C_WaitForStatusReady for the status flag wait, and changed to call I2C_WaitForStatusReady instead of polling flags with reading register.

[2.0.2]

- Improvements
  - Added I2C_WATI_TIMEOUT macro to allow users to specify the timeout times for waiting flags in functional API and blocking transfer API.

[2.0.1]

- Bug Fixes
  - Added a proper handle for transfer config flag kI2C_TransferNoStartFlag to support transmit with kI2C_TransferNoStartFlag flag. Only supports write only or write+read with no start flag; does not support read only with no start flag.

[2.0.0]

- Initial version.

MU

[2.3.1]

- Bug Fixes
  - Fixed FSL_FEATURE_MU_HAS_RESET_DEASSERT_INT macro use.

[2.3.0]

- New Features
  - Added MU_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
  - Added timeout mechanism to all polling loops in MU driver code.
  - Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
- Improvements
  - Updated function signatures to return status codes for better error handling:
    - Changed MU_ResetBothSides to return status_t instead of void
    - Updated MU_SendMsg to return status_t for timeout indication
    - Updated MU_ReceiveMsg to use MU_TIMEOUT_VALUE (0xFFFFFFFF) as a special return value to indicate timeout
  - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.2.0]

- New Features
  - Added API MU_GetRxStatusFlags.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
  - Fixed general interrupt comment typo.

[2.1.0]

- Improvements
  - Added new enum mu_msg_reg_index_t.

[2.0.7]

- Bug Fixes
  - Fixed MU_GetInterruptsPending bug that can not get general interrupt status.

[2.0.6]

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

[2.0.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 14.4, 15.5.

[2.0.4]

- Improvements
  - Improved for the platforms which don't support reset assert interrupt and get the other core power mode.

[2.0.3]

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

[2.0.2]

- Improvements
  - Added support for MIMX8MQx.

[2.0.1]

- Improvements
  - Added support for MCIMX7Ux_M4.

[2.0.0]

- Initial version.

PDM

[2.9.3]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
  - Added new features FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ/FSL_FEATURE_PDM_HAS_DC_OUT

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - Added software buffer queue for transactional API.

[2.0.1]

- Improvements
  - Improved HWVAD feature.

[2.0.0]

- Initial version.

PDM_SDMA

[2.7.1]

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

[2.7.0]

- Improvements
  - Added new api PDM_TransferTerminateReceiveSDMA to reset all PDM SDMA internal state machine.

[2.6.0]

- Bug Fixes
  - Fixed burst length overflow issue when using multi fifo.
  - Added alignment check between transfer size and the fifo width.

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.3.1

PWM

[2.0.1]

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

[2.0.0]

- Initial version.

RDC

[2.2.0]

- New Features
  - Added APIs to get memory region or peripheral access policy for specific domain.

[2.1.1]

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

[2.1.0]

- Improvements
  - Enhanced to support memory region larger than 32-bit address.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 11.3, 11.8, 17.7.

[2.0.1]

- Bug Fixes:
  - Added __DSB after new configuration is set to ensure the new configuration takes effect.

[2.0.0]

- Initial version.

RDC_SEMA42

[2.0.5]

- Bug Fixes
  - Fixed CERT INT30-C issues.

[2.0.4]

- Improvements
  - Changed to implement RDC_SEMAPHORE_Lock base on RDC_SEMAPHORE_TryLock.

[2.0.3]

- Improvements:
  - Supported the RDC_SEMAPHORE_Type structure whose gate registers are defined as an array.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 10.8, 14.3, 14.4, 18.1.

[2.0.1]

- Improvements:
  - Added support for the platforms that don't have dedicated RDC_SEMA42 clock gate.

[2.0.0]

- Initial version.

SAI

[2.4.10]

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

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

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

[2.4.7]

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

[2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

[2.4.5]

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

[2.4.4]

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

[2.4.3]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
  - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

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

[2.3.6]

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

[2.3.5]

- Improvements
  - Make driver to be aarch64 compatible.

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

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

[2.0.0]

- Initial version.

SAI_SDMA

[2.6.0]

- Improvements
  - Removed deprecated API (SAI_TransferTxSetFormatSDMA,SAI_TransferRxSetFormatSDMA) and replaced them with new API (SAI_TransferRxSetConfigSDMA,SAI_TransferRxSetConfigSDMA)

[2.5.3]

- Bug Fixes
  - Fixed the internal state of bd pool can not be reset in SAI_TransferTerminateSendSDMA.

[2.5.2]

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

[2.5.1]

- Bug Fixes
  - Fixed the incorrect internal queue state after abort the transfer in callback.

[2.5.0]

- Bug Fixes
  - Fixed the incorrect driver queue state after calling abort function.
- Improvements
  - Added new api SAI_TransferTerminateSendSDMA/SAI_TransferTerminateReceiveSDMA.

[2.4.1]

- Bug Fixes
  - Fixed the handler array s_sdmaPrivateHandle overflow issue.

[2.4.0]

- Bug Fixes
  - Fixed burst length overflow issue when using multi fifo.
  - Added alignment check between transfer size and the fifo width.
  - Fixed violations of MISRA C-2012 rule 10.6.

[2.3.1]

- Refer SAI driver change log 2.1.0 to 2.3.1

SDMA

[2.4.3]

- Bug Fixes
  - Fixed coverity issues with CERT INT30-C, CERT INT31-C compliance.

[2.4.2]

- Bug Fixes
  - Add global variable s_ramScriptLoaded to fix compatibility issues.

[2.4.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4, 11.8.

[2.4.0]

- Improvements
  - Added load ram script automatically according to the peripheral type.
  - Updated sdma ram script to version I.MX7D_4_6.
  - Added api SDMA_GetRamScriptVersion to support to SDMA ram script version.
- Bug Fixes
  - Removed the SDMA_StartChannelEvents from SDMA_StartTransfer as sdma scheduler will handle it automatically

[2.3.6]

- Bug Fixes
  - Cleared the SDMAARM_CONFIG_CSM initial value before write value 0 using OR.

[2.3.5]

- Bug Fixes
  - Added transfer size validation to aovid overflow.

[2.3.4]

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

[2.3.3]

- Improvements
  - Improved sdma driver comments and parameters validation.

[2.3.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.6.

[2.3.1]

- Improvements
  - Removed clear all channel interrupt status in SDMA_HandleIRQ to avoid the possibility of lossing interrupt.

[2.3.0]

- Improvements
  - Added peripheral-to-peripheral support in SDMA driver.
  - Added 24bit data width support in sdma driver.
- Bug Fixes
  - Fixed Coverity issue: left shift may overflow issue.
  - Fixed MISRA2004 issue: the operand of underlying type 'unsigned char' or 'unsigned short' caused the result cast to the underlying type.
  - Fixed violations of MISRA C-2012 rule 10.3, 11.9, 10.4, 17.7, 20.7, 14.4, 11.6, 12.2, 16.4.

[2.2.1]

- Bug Fixes
  - Fixed MISRA 2004 issue in sdma driver.

[2.2.0]

- Improvements
  - Added fsl_sdma_script.h to define the sdma script address and firmware.
  - Updated the format of generic register R7 to align with newest firmware.

[2.1.1]

- Improvements
  - Added SDMA_SetDoneConfig to support hardware/software done configuration.
  - Marked SDMC_EnableSwDone as deprecated.
- Bug Fixes
  - Fixed logical dead code issue in function SDMA_SetDoneConfig.

[2.1.0]

- Improvements
  - Added SDMA_SetMultiFifoConfig API to support multi fifo feature.
  - Added SDMA_EnableSwDone API to support software done feature.
  - Added SDMA_LoadScript API to support load script to SDMA program memory.
  - Added SDMA_DumpScript API to support dump script from SDMA program memory.
  - Added SDMA3 IRQ handler.

[2.0.0]

- Initial version.

SEMA4

[2.2.2]

- Improvements
  - Updated SEMA4_TryLock function to avoid unsigned integer operations wrap issue.

[2.2.1]

- Bug Fixes
  - Fixed violations of the CERT INT31-C, MISRA C-2012 rules 10.3, 10.4.

[2.2.0]

- New Features
  - Added SEMA4_BUSY_POLL_COUNT parameter to prevent infinite polling loops in SEMA4 operations.
  - Added timeout mechanism to all polling loops in SEMA4 driver code.
- Improvements
  - Updated SEMA4_Lock function to return status_t instead of void for better error handling.
  - Enhanced documentation to clarify timeout behavior and return values.

[2.1.0]

- Improvements
  - Changed mask parameter type in SEMA4_EnableGateNotifyInterrupt() and SEMA4_DisableGateNotifyInterrupt() functions to avoid casting from unsigned long to unsigned short in the code when modifying the 16bits CPINE register.

[2.0.3]

- Improvements
  - Changed to implement SEMA4_Lock base on SEMA4_TryLock.

[2.0.2]

- Improvements:
  - Supported the SEMA4_Type structure whose gate registers are defined as an array.

[2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 15.5, 18.1, 18.4.

[2.0.0]

- Initial version.

TMU

[2.1.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.4, 15.5.

[2.1.0]

- Other Changes:
  - Removed the gain setting and reference voltage setting of amplifier.

[2.0.1]

- Bug Fixes
  - Fixed missing right pair definition for extern C.

[2.0.0]

- Initial version.
- This module was first developed on i.MX 8MM.

UART

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

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

[2.3.0]

- Bug Fixes
  - Fixed DMA transfer blocking issue by enabling tx idle interrupt after DMA transmission finishes.

[2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 rule 10.4 violation.

[2.2.0]

- New Features
  - Modified uart_config_t, UART_Init and UART_GetDefaultConfig APIs so that the RTS and CTS used for hardware flow control can be enabled during module initialization.
  - Added API UART_SetRxRTSWatermark so that the water mark level of RTS deassertion can be configured.

[2.1.1]

- Bug Fixes
  - Fixed MISRA 8.5 violation.

[2.1.0]

- Improvements
  - Added timeout mechanism when waiting for certain states in transfer driver.

[2.0.2]

- Improvements
  - Added check for transmission complete in UART_WriteBlocking, UART_TransferHandleIRQ and UART_SendSDMACallback to ensure all the data would be sent out to bus.
  - Modified UART_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.
- Bug Fixes
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 11.9, 14.4.

[2.0.1]

- Bug Fixes
  - Memset local variable SDMA transfer configuration structure to make sure unused members in structure are cleared.

[2.0.0]

- Initial version.

UART_SDMA

[2.3.0]

- Refer UART driver change log 2.0.0 to 2.1.1, 2.3.0.

WDOG

[2.2.0]

- Bug Fixes
  - Fixed the wrong behavior of workMode.enableWait, workMode.enableStop, workMode.enableDebug in configuration structure wdog_config_t. When set the items to true, WDOG will continues working in those modes.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 10.6, 10.7 and 11.9.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    - WDOG_Init
    - WDOG_Refresh

[2.1.0]

- New Features
  - Added new API “WDOG_TriggerSystemSoftwareReset()” to allow users to reset the system by software.
  - Added new API “WDOG_TriggerSoftwareSignal()” to allow users to trigger a WDOG_B signal by software.
  - Removed the parameter “softwareAssertion” and “softwareResetSignal” out of the wdog_config_t structure.
  - Added new parameter “enableTimeOutAssert” to the wdog_config_t structure. With this parameter enabled, when the WDOG timeout occurs, a WDOG_B signal will be asserted. This signal can be routed to external pin of the chip. Note that WDOG_B signal remains asserted until a power-on reset (POR) occurs.

[2.0.1]

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

[2.0.0]

- Initial version.

1.6 Driver API Reference Manual

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

MIMX8MN6

1.7 Middleware Documentation

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

1.7.1 Multicore

Multicore SDK

1.7.2 FreeMASTER

freemaster

1.7.3 FreeRTOS

FreeRTOS

Chapter 2

MIMX8MN6

2.1 ASRC: Asynchronous sample rate converter

2.2 ASRC Driver

uint32_t ASRC_GetInstance(ASRC_Type *base)

Get instance number of the ASRC peripheral.

Parameters

- base – ASRC base pointer.

void ASRC_Init(ASRC_Type *base)

brief Initializes the asrc peripheral.

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

param base asrc base pointer.

void ASRC_Deinit(ASRC_Type *base)

De-initializes the ASRC peripheral.

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

Parameters

- base – ASRC base pointer.

void ASRC_GetContextDefaultConfig(*asrc_context_config_t* *config, uint32_t channels, uint32_t inSampleRate, uint32_t outSampleRate)

ASRC get context default configuration.

Parameters

- config – ASRC context configuration pointer.
- channels – input audio data channel numbers.
- inSampleRate – input sample rate.
- outSampleRate – output sample rate.

status_t ASRC_SetContextConfig(ASRC_Type *base, *asrc_context_t* context, *asrc_context_config_t* *config)

ASRC configure context.

Parameters

- base – ASRC base pointer.
- context – index of asrc context, reference asrc_context_t.
- config – ASRC context configuration pointer.

Return values

kStatus_InvalidArgument – invalid parameters. kStatus_ASRCConfigureFailed context configure failed. kStatus_Success context configure success.

status_t ASRC_SetContextOutputConfig(ASRC_Type *base, *asrc_context_t* context, *asrc_context_output_config_t* *config)

ASRC configure context output.

Parameters

- base – ASRC base pointer.
- context – index of asrc context, reference asrc_context_t.
- config – ASRC context output configuration pointer.

status_t ASRC_SetContextInputConfig(ASRC_Type *base, *asrc_context_t* context, *asrc_context_input_config_t* *config)

ASRC configure context input.

Parameters

- base – ASRC base pointer.
- context – index of asrc context, reference asrc_context_t.
- config – ASRC context input configuration pointer.

static inline void ASRC_EnableContextRun(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context enable run. All control fileds associated with a context must be stable prior to setting context run enable.

Parameters

- base – ASRC base pointer.
- context – ASRC context index.
- enable – true is enable, inform the datapath begin processing sample data for the context. false is disable, data processing will halt immediately.

static inline void ASRC_EnableContextRunStop(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context enable run stop. This function used to flush the ASRC pipeline and completely end processing for a context.

Parameters

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

static inline void ASRC_EnableContextInDMA(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context input DMA request enable.

Parameters

- base – ASRC base pointer.

- context – ASRC context index.
- enable – true is enable, false is disable.

static inline void ASRC_EnableContextOutDMA(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context output DMA request enable.

Parameters

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

static inline void ASRC_EnablePreFilterBypass(ASRC_Type *base, *asrc_context_t* context, bool bypass)

ASRC prefilter bypass mode This function enable or disable the ASRC prefilter bypass mode.

Parameters

- base – ASRC peripheral base address.
- context – context processor number.
- bypass – true is bypass, false is normal mode.

static inline void ASRC_EnableResamplerBypass(ASRC_Type *base, *asrc_context_t* context, bool bypass)

ASRC resampler bypass mode This function enable or disable the ASRC resampler bypass mode.

Parameters

- base – ASRC peripheral base address.
- context – context processor number.
- bypass – true is bypass, false is normal mode.

static inline void ASRC_SetContextChannelNumber(ASRC_Type *base, *asrc_context_t* context, uint32_t channels)

ASRC set context channel number. Note: The maximum channel number in one context can not exceed 32.

Parameters

- base – ASRC peripheral base address.
- context – context number.
- channels – channel number, should <= 32.

uint32_t ASRC_GetContextOutSampleSize(uint32_t inSampleRate, uint32_t inSamplesSize, uint32_t inWidth, uint32_t outSampleRate, uint32_t outWidth)

ASRC get output sample count.

Parameters

- inSampleRate – output sample rate.
- inSamplesSize – input sample rate.
- inWidth – input samples buffer size, the size of buffer should be converted to align with 4 byte .
- outSampleRate – input sample width.

- outWidth – Output width.

Return values
output – samples size.

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

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

Parameters

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

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

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

Parameters

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

static inline uint32_t ASRC_GetInterruptStatus(ASRC_Type *base)

Gets the ASRC interrupt status flag state.

Parameters

- base – ASRC base pointer

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

static inline void ASRC_ClearInterruptStatus(ASRC_Type *base, uint32_t status)

clear the ASRC interrupt status flag state.

Parameters

- base – ASRC base pointer
- status – status flag to be cleared.

static inline uint32_t ASRC_GetFifoStatus(ASRC_Type *base, *asrc_context_t* context)

Gets the ASRC fifo status flag.

Parameters

- base – ASRC base pointer
- context – context id

static inline void ASRC_WriteContextFifo(ASRC_Type *base, *asrc_context_t* context, uint32_t data)

write the ASRC context fifo.

Parameters

- base – ASRC base pointer.
- context – context id.
- data – data to write.

static inline uint32_t ASRC_ReadContextFifo(ASRC_Type *base, *asrc_context_t* context)

read the ASRC context fifo.

Parameters

- base – ASRC base pointer.
- context – context id.

Return values
read – data.

static inline uint32_t ASRC_GetWriteContextFifoAddr(ASRC_Type *base, *asrc_context_t* context)

Get ASRC write fifo address.

Parameters

- base – ASRC base pointer.
- context – context id.

Return values
write – fifo address.

static inline uint32_t ASRC_GetReadContextFifoAddr(ASRC_Type *base, *asrc_context_t* context)

Get the ASRC read context fifo address.

Parameters

- base – ASRC base pointer.
- context – context id.

Return values
read – fifo address.

uint32_t ASRC_ReadFIFORemainedSample(ASRC_Type *base, *asrc_context_t* context, uint32_t *outAddr, uint32_t outWidth, uint32_t sampleCount)

Get the ASRC read fifo remained samples. Since the DMA request will be triggered only when the sample group in read fifo is bigger then the watermark, so when the data size cannot be divisible by the (watermark + 1), then part of sample will left in read fifo, application should call this api to get the left samples.

Parameters

- base – ASRC base pointer.
- context – context id.
- outAddr – address to receive remained sample in read fifo.
- outWidth – output data width.
- sampleCount – specify the read sample count.

Return values
sample – counts actual read from output fifo.

status_t ASRC_TransferBlocking(ASRC_Type *base, *asrc_context_t* context, *asrc_transfer_t* *xfer)

ASRC blocking convert audio sample rate. This function depends on the configuration of input and output, so it should be called after the ASRC_SetContextConfig. The data format it supports: 1.16bit 16bit per sample in input buffer, input buffer size should be calculate as: samples 2U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the parameter outWidth should be 2 2.20bit 24bit per sample in input buffer, input buffer size should be calculate as: samples 3U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the outWidth should be 3. 3.24bit 24bit per sample in input buffer, input buffer size should be calculate as: samples * 3U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the outWidth should be 3. 4.32bit 32bit per sample in input buffer, input buffer size should be calculate as: samples * 4U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the outWidth should be 4.

Parameters

- base – ASRC base pointer.
- context – context id.

- xfer – .xfer configuration.

Return values
kStatus_Success. –

FSL_ASRC_DRIVER_VERSION
Version 2.0.7

ASRC return status, _asrc_status.

Values:

enumerator kStatus_ASRCIdle
ASRC is idle.

enumerator kStatus_ASRCBusy
ASRC is busy.

enumerator kStatus_ASRCInvalidArgument
ASRC invalid argument.

enumerator kStatus_ASRCConfigureFailed
ASRC configure failed

enumerator kStatus_ASRCConvertError
ASRC convert error failed

enumerator kStatus_ASRCNotSupport
ASRC not support

enumerator kStatus_ASRCQueueFull
ASRC queue full

enumerator kStatus_ASRCQueueIdle
ASRC quue idle

enumerator kStatus_ASRCLoadFirmwareFailed
ASRC load firmware failed

enumerator kStatus_ASRCResamplerConfigureFailed
ASRC resampler configured failed

enumerator kStatus_ASRCPrefilterConfigureFailed
ASRC prefilter configured failed

enum _asrc_context
asrc context id

Values:

enumerator kASRC_Context0
Context 0 value

enumerator kASRC_Context1
Context 1 value

enumerator kASRC_Context2
Context 2 value

enumerator kASRC_Context3
Context 3 value

The ASRC interrupt enable flag, _asrc_interrupt_mask.

Values:

enumerator kASRC_Context0InputFifoOverflow
context 0 input fifo overflow

enumerator kASRC_Context1InputFifoOverflow
context 1 input fifo overflow

enumerator kASRC_Context2InputFifoOverflow
context 2 input fifo overflow

enumerator kASRC_Context3InputFifoOverflow
context 3 input fifo overflow

enumerator kASRC_Context0OutFifoReadEmpty
context 0 out fifo read empty

enumerator kASRC_Context1OutFifoReadEmpty
context 1 out fifo read empty

enumerator kASRC_Context2OutFifoReadEmpty
context 2 out fifo read empty

enumerator kASRC_Context3OutFifoReadEmpty
context 3 out fifo read empty

enumerator kASRC_Context0RunStopDone
context 0 run stop done interrupt

enumerator kASRC_Context1RunStopDone
context 1 run stop done interrupt

enumerator kASRC_Context2RunStopDone
context 2 run stop done interrupt

enumerator kASRC_Context3RunStopDone
context 3 run stop done interrupt

enumerator kASRC_ContextAllInterruptStatus
all the context interrupt status

ASRC fifo status, _asrc_fifo_status.

Values:

enumerator kASRC_FifoStatusInputFifoWatermarkFlag
input water mark flag raised

enumerator kASRC_FifoStatusOutputFifoWatermarkFlag
output water mark flag raised

enum _asrc_data_endianness
arsc data endianness

Values:

enumerator kASRC_DataEndianLittle
context data little endian

enumerator kASRC_DataEndianBig
context data big endian

enum _asrc_data_width
data width

Values:

enumerator kASRC_DataWidth32Bit
data width 32bit

enumerator kASRC_DataWidth24Bit
data width 24bit

enumerator kASRC_DataWidth20Bit
data width 20bit

enumerator kASRC_DataWidth16Bit
data width 16bit

enum _asrc_data_type
data type

Values:

enumerator kASRC_DataTypeInteger
data type int

enumerator kASRC_DataTypeFloat
data type float, single precision floating point format

enum _asrc_data_sign
sign extension

Values:

enumerator kASRC_DataSigned
input data is signed

enumerator kASRC_DataUnsigned
input data is unsinged

enum _asrc_sampleBuffer_init_mode
asrc prefilter and resampler sample buffer init mode

Values:

enumerator kASRC_SampleBufferNoPreFillOnInit
do not pre-fill

enumerator kASRC_SampleBufferFillFirstSampleOnInit
replicate the first sample to fill the right half of the sample buffer

enumerator kASRC_SampleBufferFillZeroOnInit
zero fill the right half og the sample buffer

enum _asrc_sampleBuffer_stop_mode
asrc prefilter and resampler sample buffer stop mode

Values:

enumerator kASRC_SampleBufferFillLastSampleOnStop
replicate the last sample to fill the left half of the sample buffer

enumerator kASRC_SampleBufferFillZeroOnStop
zero fill the left half of the sample buffer

enum _asrc_prefilter_stage1_result
ASRC prefilter stage1 result format.

Values:

enumerator kASRC_PrefilterStage1ResultInt
prefilter stage1 results are stored in 32 bit int format

enumerator kASRC_PrefilterStage1ResultFloat
prefilter stage1 results are stored in 32 bit float format

enum _asrc_resampler_taps
ASRC resampler taps.

Values:

enumerator kASRC_ResamplerTaps_32
resampler taps 32

enumerator kASRC_ResamplerTaps_64
resampler taps 64

enumerator kASRC_ResamplerTaps_128
resampler taps 128

ASRC support sample rate, _asrc_sample_rate.

Values:

enumerator kASRC_SampleRate_8000
8K sample rate

enumerator kASRC_SampleRate_11025
11025 sample rate

enumerator kASRC_SampleRate_12000
12K sample rate

enumerator kASRC_SampleRate_16000
16K sample rate

enumerator kASRC_SampleRate_22050
22.05K sample rate

enumerator kASRC_SampleRate_24000
24K sample rate

enumerator kASRC_SampleRate_32000
32K sample rate

enumerator kASRC_SampleRate_44100
44.1K sample rate

enumerator kASRC_SampleRate_48000
48K sample rate

enumerator kASRC_SampleRate_64000
64K sample rate

enumerator kASRC_SampleRate_88200
88.2K sample rate

enumerator kASRC_SampleRate_96000
96K sample rate

enumerator kASRC_SampleRate_128000
128K sample rate

enumerator kASRC_SampleRate_176400
176K sample rate

enumerator kASRC_SampleRate_192000
256K sample rate

enumerator kASRC_SampleRate_256000
256K sample rate

enumerator kASRC_SampleRate_352800
352.8K sample rate

enumerator kASRC_SampleRate_384000
384K sample rate

enumerator kASRC_SampleRate_768000
768K sample rate

typedef enum *_asrc_context* asrc_context_t
asrc context id

typedef enum *_asrc_data_endianness* asrc_data_endianness_t
arsc data endianness

typedef enum *_asrc_data_width* asrc_data_width_t
data width

typedef enum *_asrc_data_type* asrc_data_type_t
data type

typedef enum *_asrc_data_sign* asrc_data_sign_t
sign extension

typedef enum *_asrc_sampleBuffer_init_mode* asrc_sampleBuffer_init_mode_t
asrc prefilter and resampler sample buffer init mode

typedef enum *_asrc_sampleBuffer_stop_mode* asrc_sampleBuffer_stop_mode_t
asrc prefilter and resampler sample buffer stop mode

typedef enum *_asrc_prefilter_stage1_result* asrc_prefilter_stage1_result_t
ASRC prefilter stage1 result format.

typedef enum *_asrc_resampler_taps* asrc_resampler_taps_t
ASRC resampler taps.

typedef struct *_asrc_data_format* asrc_data_format_t
asrc context data format

typedef struct *_asrc_access_ctrl* asrc_access_ctrl_t
asrc context access control The ASRC provides interleaving support in hardware to ensure that a variety of sample source can be internally combined tp confir with this format. The interleave patter is controlled using 3 register fields: GROUP_LENGTH, ACCESS_LENGTH,

ITERATIONIS. This is intended to support hardware configurations which distribute a single context across samples from multiple audio sources. Take a example as below: accessGroupLen = 6, the sample group length is 6 samples accessIterations = 2, the 2 sequential ACCESS_LENGTH read from single source accessLen = 2, the 2 samples fetch from one source.

typedef struct *_asrc_context_input_config* asrc_context_input_config_t
asrc context input configuration

typedef struct *_asrc_context_output_config* asrc_context_output_config_t
asrc context output configuration

typedef struct *_asrc_context_prefilter_config* asrc_context_prefilter_config_t
asrc context prefilter configuration

typedef struct *_asrc_context_resampler_config* asrc_context_resampler_config_t
asrc context resampler configuration

typedef struct *_asrc_context_config* asrc_context_config_t
asrc context configuration

typedef struct *_asrc_transfer* asrc_transfer_t
ASRC transfer.

FSL_ASRC_INPUT_FIFO_DEPTH
ASRC fifo depth.

FSL_ASRC_OUTPUT_FIFO_DEPTH

ASRC_SUPPORT_MAXIMUM_CONTEXT_PROCESSOR_NUMBER
ASRC support maximum channel number of context.

struct asrc_data_format
#include <fsl_asrc.h> asrc context data format

Public Members

uint8_t dataPosition
context input data sample position

asrc_data_endianness_t dataEndianness
context input data endianness

asrc_data_width_t dataWidth
context input data width

asrc_data_type_t dataType
context input data type

asrc_data_sign_t dataSign
context input data signed or unsigned

struct asrc_access_ctrl
#include <fsl_asrc.h> asrc context access control The ASRC provides interleaving support in hardware to ensure that a variety of sample source can be internally combined tp confir with this format. The interleave patter is controlled using 3 register fields: GROUP_LENGTH, ACCESS_LENGTH, ITERATIONIS. This is intended to support hardware configurations which distribute a single context across samples from multiple audio sources. Take a example as below: accessGroupLen = 6, the sample group length is 6 samples accessIterations = 2, the 2 sequential ACCESS_LENGTH read from single source accessLen = 2, the 2 samples fetch from one source.

Public Members

uint8_t accessIterations
: number of sequential fetches per source

uint8_t accessGroupLen
: number of channels in a context

uint8_t accessLen
: number of channels per source1

struct __asrc__context__input__config
: *#include <fsl_asrc.h>* asrc context input configuration

Public Members

uint32_t sampleRate
: input audio data sample rate

uint8_t watermark
: input water mark per samples

asrc_access_ctrl_t accessCtrl
: input access control

asrc_data_format_t dataFormat
: input data format

struct __asrc__context__output__config
: *#include <fsl_asrc.h>* asrc context output configuration

Public Members

uint32_t sampleRate
: output audio data sample rate

uint8_t watermark
: output water mark per samples

asrc_access_ctrl_t accessCtrl
: output access control

asrc_data_format_t dataFormat
: output data format

bool enableDither
: output path contains a TPDF dither function. The dither function support all fixed output modes(16, 20, 24, 32bits) dither is not supported in 32bit floating point output mode

bool enableIEC60958
: output IEC60958 bit field insertion enable

struct __asrc__context__prefilter__config
: *#include <fsl_asrc.h>* asrc context prefilter configuration

Public Members

asrc_sampleBuffer_init_mode_t initMode
: prefilter initial mode

asrc_sampleBuffer_stop_mode_t stopMode
: prefilter stop mode

asrc_prefilter_stage1_result_t stage1Result
: stage1 data store format

uint32_t filterSt1Taps
: prefilter stage1 taps

uint32_t filterSt2Taps
: prefilter stage2 taps

uint32_t filterSt1Exp
: prefilter stage1 expansion factor

const uint32_t *filterCoeffAddress
: prefilter coeff address

struct __asrc__context__resampler__config
: *#include <fsl_asrc.h>* asrc context resampler configuration

Public Members

asrc_sampleBuffer_init_mode_t initMode
: initial mode

asrc_sampleBuffer_stop_mode_t stopMode
: resampler stop mode

asrc_resampler_taps_t tap
: resampleer taps

uint32_t filterPhases
: interpolation phases

uint64_t filterCenterTap
: interpolation center tap

const uint32_t *filterCoeffAddress
: interpolation coeff address

struct __asrc__context__config
: *#include <fsl_asrc.h>* asrc context configuration

Public Members

uint8_t contextChannelNums
: context channel numbers

asrc_context_input_config_t contextInput
: context input configuration

asrc_context_output_config_t contextOutput
: context output configuration

asrc_context_prefilter_config_t contextPrefilter
context pre filter configuration

asrc_context_resampler_config_t contextResampler
context resampler configuration

struct __asrc__transfer
#include <fsl_asrc.h> ASRC transfer.

Public Members

uint32_t *inDataAddr
address of audio data to be converted

uint32_t inDataSize
size of the audio data

uint32_t *outDataAddr
address of audio data that is been converted

uint32_t outDataSize
size of the audio data

2.3 ASRC SDMA Driver

void ASRC_TransferInCreateHandleSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_sdma_callback_t* callback, *sdma_handle_t* *dmaHandle, uint32_t eventSource, *asrc_context_t* context, const *asrc_p2p_sdma_config_t* *periphConfig, void *userData)

Initializes the ASRC input SDMA handle.

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

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- base – ASRC peripheral base address.
- callback – Pointer to user callback function.
- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – ASRC input sdma event source.
- context – ASRC context number.
- periphConfig – peripheral configurations, used for case.
- userData – User parameter passed to the callback function.

void ASRC_TransferOutCreateHandleSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_sdma_callback_t* callback, *sdma_handle_t* *dmaHandle, uint32_t eventSource, *asrc_context_t* context, const *asrc_p2p_sdma_config_t* *periphConfig, void *userData)

Initializes the ASRC output SDMA handle.

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

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- callback – ASRC outcallback.
- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – ASRC output event source.
- context – ASRC context number.
- periphConfig – peripheral configurations, used for case.
- userData – User parameter passed to the callback function.

status_t ASRC_TransferSetContextConfigSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_context_config_t* *asrcConfig)

Configures the ASRC context.

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- asrcConfig – asrc context configurations.

status_t ASRC_TransferSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_transfer_t* *xfer)

Performs a non-blocking ASRC transfer using DMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- xfer – ASRC xfer configurations pointer.

Return values

- kStatus_Success – Start a ASRC SDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – ASRC is busy sending data.

void ASRC_TransferAbortInSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle)

Aborts a ASRC in transfer using SDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.

void ASRC_TransferAbortOutSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle)

brief Aborts a ASRC out transfer using SDMA.

param base ASRC base pointer. param handle ASRC SDMA handle pointer.

FSL_ASRC_SDMA_DRIVER_VERSION
Version 2.0.4

typedef struct *_asrc_sdma_handle* asrc_sdma_handle_t
ASRC sdma handle prototype.

typedef void (*asrc_sdma_callback_t)(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *status_t* status, void *userData)
ASRC SDMA transfer callback function for finish and error.

typedef void (*asrc_start_peripheral_t)(bool start)
ASRC trigger peripheral function pointer.

typedef struct *_asrc_p2p_sdma_config* asrc_p2p_sdma_config_t
destination peripheral configuration

typedef struct *_asrc_sdma_in_handle* asrc_sdma_in_handle_t
ASRC sdma in handle.

typedef struct *_asrc_sdma_out_handle* asrc_sdma_out_handle_t
ASRC sdma out handle.

ASRC_XFER_IN_QUEUE_SIZE
ASRC xfer queue size.

ASRC_XFER_OUT_QUEUE_SIZE

struct _asrc_p2p_sdma_config
#include <fsl_asrc_sdma.h> destination peripheral configuration

Public Members

uint32_t eventSource
peripheral event source

uint8_t watermark
peripheral watermark

uint8_t channel
peripheral channel number

uint8_t fifoWidth
peripheral fifo width

bool enableContinuous
true is the amount of samples to be transferred is unknown and script will keep on transferring as long as both events are detected and script must be stopped by application, false is The amount of samples to be transferred is equal to the count field of mode word

asrc_start_peripheral_t startPeripheral
trigger peripheral start

struct _asrc_sdma_in_handle
#include <fsl_asrc_sdma.h> ASRC sdma in handle.

Public Members

sdma_handle_t *sdmaHandle
DMA handler for ASRC

uint32_t eventSource
: ASRC event source number

asrc_sdma_callback_t callback
: Callback for users while transfer finish or error occurs

void *userData
: User callback parameter

sdma_buffer_descriptor_t bdPool[4U]
: BD pool for SDMA transfer.

uint8_t asrcInWatermark
: The transfer data count in a DMA request

uint8_t bytesPerSample
: Bytes in a sample

uint32_t *asrcQueue[4U]
: Transfer queue storing queued transfer.

size_t sdmaTransferSize[4U]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer.

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

const *asrc_p2p_sdma_config_t* *peripheralConfig
: peripheral configuration

uint32_t state
: Internal state for ASRC SDMA transfer

struct __asrc__sdma__out__handle
: *#include <fsl_asrc_sdma.h>* ASRC sdma out handle.

Public Members

sdma_handle_t *sdmaHandle
: DMA handler for ASRC

void *userData
: User callback parameter

uint32_t state
: Internal state for ASRC SDMA transfer

uint8_t bytesPerSample
: Bytes in a sample

uint32_t eventSource
: ASRC event source number

asrc_sdma_callback_t callback
: Callback for users while transfer finish or error occurs

uint8_t asrcOutWatermark
: The transfer data count in a DMA request

sdma_buffer_descriptor_t bdPool[(4U * 2U)]
BD pool for SDMA transfer.

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

size_t sdmaTransferSize[(4U * 2U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer.

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

const *asrc_p2p_sdma_config_t* *peripheralConfig
peripheral configuration

uint32_t nonAlignSize
non align size

void *nonAlignAddr
non align address

struct __asrc__sdma__handle
#include <fsl_asrc_sdma.h> ASRC DMA transfer handle, users should not touch the content of the handle.

Public Members

asrc_sdma_in_handle_t inDMAHandle
input dma handle

asrc_sdma_out_handle_t outDMAHandle
output dma handle

asrc_context_t context
ASRC context number

uint8_t dataChannels
ASRC process data channel number

2.4 CACHE: ARMV7-M7 CACHE Memory Controller

static inline void L1CACHE__EnableICache(void)
Enables cortex-m7 L1 instruction cache.

static inline void L1CACHE__DisableICache(void)
Disables cortex-m7 L1 instruction cache.

static inline void L1CACHE__InvalidateICache(void)
Invalidate cortex-m7 L1 instruction cache.

void L1CACHE__InvalidateICacheByRange(uint32_t address, uint32_t size_byte)
Invalidate cortex-m7 L1 instruction cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned. The startAddr here will be

forced to align to L1 I-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

static inline void L1CACHE_EnableDCache(void)

Enables cortex-m7 L1 data cache.

static inline void L1CACHE_DisableDCache(void)

Disables cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCache(void)

Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_CleanDCache(void)

Cleans cortex-m7 L1 data cache.

static inline void L1CACHE_CleanInvalidateDCache(void)

Cleans and Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cortex-m7 L1 data cache by range.

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

Parameters

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

static inline void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cortex-m7 L1 data cache by range.

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

Parameters

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

static inline void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

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

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

Parameters

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

void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all instruction caches by range.

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

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

Parameters

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

void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all data caches by range.

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

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

Parameters

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

void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Cleans all data caches by range.

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

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

Parameters

- address – The physical address.

- size_byte – size of the memory to be cleaned.

void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates all data caches by range.

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

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

Parameters

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

FSL_CACHE_DRIVER_VERSION

cache driver version 2.0.4.

2.5 Clock

enum _clock_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_CoreM7Clk

ARM M7 Core clock

enumerator kCLOCK_AxiClk

Main AXI bus clock.

enumerator kCLOCK_AhbClk

AHB bus clock.

enumerator kCLOCK_IpgClk

IPG bus clock.

enumerator kCLOCK_PerClk

Peripheral Clock.

enumerator kCLOCK_EnetIpgClk

ENET IPG Clock.

enumerator kCLOCK_Osc24MClk

OSC 24M clock.

enumerator kCLOCK_ArmPllClk

Arm PLL clock.

enumerator kCLOCK_DramPllClk

Dram PLL clock.

enumerator kCLOCK_SysPll1Clk

Sys PLL1 clock.

enumerator kCLOCK_SysPll1Div2Clk
Sys PLL1 clock divided by 2.

enumerator kCLOCK_SysPll1Div3Clk
Sys PLL1 clock divided by 3.

enumerator kCLOCK_SysPll1Div4Clk
Sys PLL1 clock divided by 4.

enumerator kCLOCK_SysPll1Div5Clk
Sys PLL1 clock divided by 5.

enumerator kCLOCK_SysPll1Div6Clk
Sys PLL1 clock divided by 6.

enumerator kCLOCK_SysPll1Div8Clk
Sys PLL1 clock divided by 8.

enumerator kCLOCK_SysPll1Div10Clk
Sys PLL1 clock divided by 10.

enumerator kCLOCK_SysPll1Div20Clk
Sys PLL1 clock divided by 20.

enumerator kCLOCK_SysPll2Clk
Sys PLL2 clock.

enumerator kCLOCK_SysPll2Div2Clk
Sys PLL2 clock divided by 2.

enumerator kCLOCK_SysPll2Div3Clk
Sys PLL2 clock divided by 3.

enumerator kCLOCK_SysPll2Div4Clk
Sys PLL2 clock divided by 4.

enumerator kCLOCK_SysPll2Div5Clk
Sys PLL2 clock divided by 5.

enumerator kCLOCK_SysPll2Div6Clk
Sys PLL2 clock divided by 6.

enumerator kCLOCK_SysPll2Div8Clk
Sys PLL2 clock divided by 8.

enumerator kCLOCK_SysPll2Div10Clk
Sys PLL2 clock divided by 10.

enumerator kCLOCK_SysPll2Div20Clk
Sys PLL2 clock divided by 20.

enumerator kCLOCK_SysPll3Clk
Sys PLL3 clock.

enumerator kCLOCK_AudioPll1Clk
Audio PLL1 clock.

enumerator kCLOCK_AudioPll2Clk
Audio PLL2 clock.

enumerator kCLOCK_VideoPll1Clk
Video PLL1 clock.

enumerator kCLOCK_ExtClk1
External clock1.

enumerator kCLOCK_ExtClk2
External clock2.

enumerator kCLOCK_ExtClk3
External clock3.

enumerator kCLOCK_ExtClk4
External clock4.

enumerator kCLOCK_NoneName
None Clock Name.

enum _clock_ip_name
CCM CCGR gate control.

Values:

enumerator kCLOCK_IpInvalid

enumerator kCLOCK_Debug
DEBUG Clock Gate.

enumerator kCLOCK_Dram
DRAM Clock Gate.

enumerator kCLOCK_Ecspi1
ECSPI1 Clock Gate.

enumerator kCLOCK_Ecspi2
ECSPI2 Clock Gate.

enumerator kCLOCK_Ecspi3
ECSPI3 Clock Gate.

enumerator kCLOCK_Enet1
ENET1 Clock Gate.

enumerator kCLOCK_Gpio1
GPIO1 Clock Gate.

enumerator kCLOCK_Gpio2
GPIO2 Clock Gate.

enumerator kCLOCK_Gpio3
GPIO3 Clock Gate.

enumerator kCLOCK_Gpio4
GPIO4 Clock Gate.

enumerator kCLOCK_Gpio5
GPIO5 Clock Gate.

enumerator kCLOCK_Gpt1
GPT1 Clock Gate.

enumerator kCLOCK_Gpt2
GPT2 Clock Gate.

enumerator kCLOCK_Gpt3
GPT3 Clock Gate.

enumerator kCLOCK_Gpt4
GPT4 Clock Gate.

enumerator kCLOCK_Gpt5
GPT5 Clock Gate.

enumerator kCLOCK_Gpt6
GPT6 Clock Gate.

enumerator kCLOCK_I2c1
I2C1 Clock Gate.

enumerator kCLOCK_I2c2
I2C2 Clock Gate.

enumerator kCLOCK_I2c3
I2C3 Clock Gate.

enumerator kCLOCK_I2c4
I2C4 Clock Gate.

enumerator kCLOCK_Iomux
IOMUX Clock Gate.

enumerator kCLOCK_Ipmux1
IPMUX1 Clock Gate.

enumerator kCLOCK_Ipmux2
IPMUX2 Clock Gate.

enumerator kCLOCK_Ipmux3
IPMUX3 Clock Gate.

enumerator kCLOCK_Ipmux4
IPMUX4 Clock Gate.

enumerator kCLOCK_Mu
MU Clock Gate.

enumerator kCLOCK_Ocram
OCRAM Clock Gate.

enumerator kCLOCK_OcramS
OCRAM S Clock Gate.

enumerator kCLOCK_Pwm1
PWM1 Clock Gate.

enumerator kCLOCK_Pwm2
PWM2 Clock Gate.

enumerator kCLOCK_Pwm3
PWM3 Clock Gate.

enumerator kCLOCK_Pwm4
PWM4 Clock Gate.

enumerator kCLOCK_Qspi
QSPI Clock Gate.

enumerator kCLOCK_Rdc
RDC Clock Gate.

enumerator kCLOCK_Sai2
SAI2 Clock Gate.

enumerator kCLOCK_Sai3
SAI3 Clock Gate.

enumerator kCLOCK_Sai5
SAI5 Clock Gate.

enumerator kCLOCK_Sai6
SAI6 Clock Gate.

enumerator kCLOCK_Sai7
SAI7 Clock Gate.

enumerator kCLOCK_Sdma1
SDMA1 Clock Gate.

enumerator kCLOCK_Sdma2
SDMA2 Clock Gate.

enumerator kCLOCK_Sec_Debug
SEC_DEBUG Clock Gate.

enumerator kCLOCK_Sema42_1
RDC SEMA42 Clock Gate.

enumerator kCLOCK_Sema42_2
RDC SEMA42 Clock Gate.

enumerator kCLOCK_Sim_display
SIM_Display Clock Gate.

enumerator kCLOCK_Sim_m
SIM_M Clock Gate.

enumerator kCLOCK_Sim_main
SIM_MAIN Clock Gate.

enumerator kCLOCK_Sim_s
SIM_S Clock Gate.

enumerator kCLOCK_Sim_wakeup
SIM_WAKEUP Clock Gate.

enumerator kCLOCK_Uart1
UART1 Clock Gate.

enumerator kCLOCK_Uart2
UART2 Clock Gate.

enumerator kCLOCK_Uart3
UART3 Clock Gate.

enumerator kCLOCK_Uart4
UART4 Clock Gate.

enumerator kCLOCK_Usdhc1
USDHC1 Clock Gate.

enumerator kCLOCK_Usdhc2
USDHC2 Clock Gate.

enumerator kCLOCK_Wdog1
WDOG1 Clock Gate.

enumerator kCLOCK_Wdog2
WDOG2 Clock Gate.

enumerator kCLOCK_Wdog3
WDOG3 Clock Gate.

enumerator kCLOCK_Asrc
ASRC Clock Gate.

enumerator kCLOCK_Pdm
PDM Clock Gate.

enumerator kCLOCK_Usdhc3
USDHC3 Clock Gate.

enumerator kCLOCK_Sdma3
SDMA3 Clock Gate.

enumerator kCLOCK_TempSensor
TempSensor Clock Gate.

enum _clock_root_control
ccm root name used to get clock frequency.

Values:

enumerator kCLOCK_RootM7
ARM Cortex-M7 Clock control name.

enumerator kCLOCK_RootAxi
AXI Clock control name.

enumerator kCLOCK_RootEnetAxi
ENET AXI Clock control name.

enumerator kCLOCK_RootNoc
NOC Clock control name.

enumerator kCLOCK_RootAhb
AHB Clock control name.

enumerator kCLOCK_RootIpg
IPG Clock control name.

enumerator kCLOCK_RootAudioAhb
Audio AHB Clock control name.

enumerator kCLOCK_RootAudioIpg
Audio IPG Clock control name.

enumerator kCLOCK_RootDramAlt
DRAM ALT Clock control name.

enumerator kCLOCK_RootSai2
SAI2 Clock control name.

enumerator kCLOCK_RootSai3
SAI3 Clock control name.

enumerator kCLOCK_RootSai5
SAI5 Clock control name.

enumerator kCLOCK_RootSai6
SAI6 Clock control name.

enumerator kCLOCK_RootSai7
SAI7 Clock control name.

enumerator kCLOCK_RootEnetRef
ENET Clock control name.

enumerator kCLOCK_RootEnetTimer
ENET TIMER Clock control name.

enumerator kCLOCK_RootEnetPhy
ENET PHY Clock control name.

enumerator kCLOCK_RootQspi
QSPI Clock control name.

enumerator kCLOCK_RootI2c1
I2C1 Clock control name.

enumerator kCLOCK_RootI2c2
I2C2 Clock control name.

enumerator kCLOCK_RootI2c3
I2C3 Clock control name.

enumerator kCLOCK_RootI2c4
I2C4 Clock control name.

enumerator kCLOCK_RootUart1
UART1 Clock control name.

enumerator kCLOCK_RootUart2
UART2 Clock control name.

enumerator kCLOCK_RootUart3
UART3 Clock control name.

enumerator kCLOCK_RootUart4
UART4 Clock control name.

enumerator kCLOCK_RootEcspi1
ECSPI1 Clock control name.

enumerator kCLOCK_RootEcspi2
ECSPI2 Clock control name.

enumerator kCLOCK_RootEcspi3
ECSPI3 Clock control name.

enumerator kCLOCK_RootPwm1
PWM1 Clock control name.

enumerator kCLOCK_RootPwm2
PWM2 Clock control name.

enumerator kCLOCK_RootPwm3
PWM3 Clock control name.

enumerator kCLOCK_RootPwm4
PWM4 Clock control name.

enumerator kCLOCK_RootGpt1
GPT1 Clock control name.

enumerator kCLOCK_RootGpt2
GPT2 Clock control name.

enumerator kCLOCK_RootGpt3
GPT3 Clock control name.

enumerator kCLOCK_RootGpt4
GPT4 Clock control name.

enumerator kCLOCK_RootGpt5
GPT5 Clock control name.

enumerator kCLOCK_RootGpt6
GPT6 Clock control name.

enumerator kCLOCK_RootWdog
WDOG Clock control name.

enumerator kCLOCK_RootPdm
PDM Clock control name.

enum _clock_root
ccm clock root index used to get clock frequency.

Values:

enumerator kCLOCK_M7ClkRoot
ARM Cortex-M7 Clock control name.

enumerator kCLOCK_AxiClkRoot
AXI Clock control name.

enumerator kCLOCK_NocClkRoot
NOC Clock control name.

enumerator kCLOCK_AhbClkRoot
AHB Clock control name.

enumerator kCLOCK_IpgClkRoot
IPG Clock control name.

enumerator kCLOCK_AudioAhbClkRoot
Audio AHB Clock control name.

enumerator kCLOCK_AudioIpgClkRoot
Audio IPG Clock control name.

enumerator kCLOCK_DramAltClkRoot
DRAM ALT Clock control name.

enumerator kCLOCK_Sai2ClkRoot
SAI2 Clock control name.

enumerator kCLOCK_Sai3ClkRoot
SAI3 Clock control name.

enumerator kCLOCK_Sai5ClkRoot
SAI5 Clock control name.

enumerator kCLOCK_Sai6ClkRoot
SAI6 Clock control name.

enumerator kCLOCK_Sai7ClkRoot
SAI7 Clock control name.

enumerator kCLOCK_QspiClkRoot
QSPI Clock control name.

enumerator kCLOCK_I2c1ClkRoot
I2C1 Clock control name.

enumerator kCLOCK_I2c2ClkRoot
I2C2 Clock control name.

enumerator kCLOCK_I2c3ClkRoot
I2C3 Clock control name.

enumerator kCLOCK_I2c4ClkRoot
I2C4 Clock control name.

enumerator kCLOCK_Uart1ClkRoot
UART1 Clock control name.

enumerator kCLOCK_Uart2ClkRoot
UART2 Clock control name.

enumerator kCLOCK_Uart3ClkRoot
UART3 Clock control name.

enumerator kCLOCK_Uart4ClkRoot
UART4 Clock control name.

enumerator kCLOCK_Ecspi1ClkRoot
ECSPI1 Clock control name.

enumerator kCLOCK_Ecspi2ClkRoot
ECSPI2 Clock control name.

enumerator kCLOCK_Ecspi3ClkRoot
ECSPI3 Clock control name.

enumerator kCLOCK_Pwm1ClkRoot
PWM1 Clock control name.

enumerator kCLOCK_Pwm2ClkRoot
PWM2 Clock control name.

enumerator kCLOCK_Pwm3ClkRoot
PWM3 Clock control name.

enumerator kCLOCK_Pwm4ClkRoot
PWM4 Clock control name.

enumerator kCLOCK_Gpt1ClkRoot
GPT1 Clock control name.

enumerator kCLOCK_Gpt2ClkRoot
GPT2 Clock control name.

enumerator kCLOCK_Gpt3ClkRoot
GPT3 Clock control name.

enumerator kCLOCK_Gpt4ClkRoot
GPT4 Clock control name.

enumerator kCLOCK_Gpt5ClkRoot
GPT5 Clock control name.

enumerator kCLOCK_Gpt6ClkRoot
GPT6 Clock control name.

enumerator kCLOCK_WdogClkRoot
WDOG Clock control name.

enumerator kCLOCK_PdmClkRoot
PDM Clock control name.

enum _clock_rootmux_m7_clk_sel
Root clock select enumeration for ARM Cortex-M7 core.

Values:

enumerator kCLOCK_M7RootmuxOsc24M
ARM Cortex-M7 Clock from OSC 24M.

enumerator kCLOCK_M7RootmuxSysPll2Div5
ARM Cortex-M7 Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_M7RootmuxSysPll2Div4
ARM Cortex-M7 Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_M7RootmuxSysPll1Div3
ARM Cortex-M7 Clock from SYSTEM PLL1 divided by 3.

enumerator kCLOCK_M7RootmuxSysPll1
ARM Cortex-M7 Clock from SYSTEM PLL1.

enumerator kCLOCK_M7RootmuxAudioPll1
ARM Cortex-M7 Clock from AUDIO PLL1.

enumerator kCLOCK_M7RootmuxVideoPll1
ARM Cortex-M7 Clock from VIDEO PLL1.

enumerator kCLOCK_M7RootmuxSysPll3
ARM Cortex-M7 Clock from SYSTEM PLL3.

enum _clock_rootmux_axi_clk_sel
Root clock select enumeration for AXI bus.

Values:

enumerator kCLOCK_AxiRootmuxOsc24M
ARM AXI Clock from OSC 24M.

enumerator kCLOCK_AxiRootmuxSysPll2Div3
ARM AXI Clock from SYSTEM PLL2 divided by 3.

enumerator kCLOCK_AxiRootmuxSysPll1
ARM AXI Clock from SYSTEM PLL1.

enumerator kCLOCK_AxiRootmuxSysPll2Div4
ARM AXI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_AxiRootmuxSysPll2
ARM AXI Clock from SYSTEM PLL2.

enumerator kCLOCK_AxiRootmuxAudioPll1
ARM AXI Clock from AUDIO PLL1.

enumerator kCLOCK_AxiRootmuxVideoPll1
ARM AXI Clock from VIDEO PLL1.

enumerator kCLOCK_AxiRootmuxSysPll1Div8
ARM AXI Clock from SYSTEM PLL1 divided by 8.

enum _clock_rootmux_ahb_clk_sel
Root clock select enumeration for AHB bus.

Values:

enumerator kCLOCK_AhbRootmuxOsc24M
ARM AHB Clock from OSC 24M.

enumerator kCLOCK_AhbRootmuxSysPll1Div6
ARM AHB Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK_AhbRootmuxSysPll1
ARM AHB Clock from SYSTEM PLL1.

enumerator kCLOCK_AhbRootmuxSysPll1Div2
ARM AHB Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK_AhbRootmuxSysPll2Div8
ARM AHB Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_AhbRootmuxSysPll3
ARM AHB Clock from SYSTEM PLL3.

enumerator kCLOCK_AhbRootmuxAudioPll1
ARM AHB Clock from AUDIO PLL1.

enumerator kCLOCK_AhbRootmuxVideoPll1
ARM AHB Clock from VIDEO PLL1.

enum _clock_rootmux_audio_ahb_clk_sel
Root clock select enumeration for Audio AHB bus.

Values:

enumerator kCLOCK_AudioAhbRootmuxOsc24M
ARM Audio AHB Clock from OSC 24M.

enumerator kCLOCK_AudioAhbRootmuxSysPll2Div2
ARM Audio AHB Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_AudioAhbRootmuxSysPll1
ARM Audio AHB Clock from SYSTEM PLL1.

enumerator kCLOCK_AudioAhbRootmuxSysPll2
ARM Audio AHB Clock from SYSTEM PLL2.

enumerator kCLOCK_AudioAhbRootmuxSysPll2Div6
ARM Audio AHB Clock from SYSTEM PLL2 divided by 6.

enumerator kCLOCK_AudioAhbRootmuxSysPll3
ARM Audio AHB Clock from SYSTEM PLL3.

enumerator kCLOCK_AudioAhbRootmuxAudioPll1
ARM Audio AHB Clock from AUDIO PLL1.

enumerator kCLOCK_AudioAhbRootmuxVideoPll1
ARM Audio AHB Clock from VIDEO PLL1.

enum _clock_rootmux_qspi_clk_sel
Root clock select enumeration for QSPI peripheral.

Values:

enumerator kCLOCK_QspiRootmuxOsc24M
ARM QSPI Clock from OSC 24M.

enumerator kCLOCK_QspiRootmuxSysPll1Div2
ARM QSPI Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK_QspiRootmuxSysPll2Div3
ARM QSPI Clock from SYSTEM PLL2 divided by 3.

enumerator kCLOCK_QspiRootmuxSysPll2Div2
ARM QSPI Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_QspiRootmuxAudioPll2
ARM QSPI Clock from AUDIO PLL2.

enumerator kCLOCK_QspiRootmuxSysPll1Div3
ARM QSPI Clock from SYSTEM PLL1 divided by 3

enumerator kCLOCK_QspiRootmuxSysPll3
ARM QSPI Clock from SYSTEM PLL3.

enumerator kCLOCK_QspiRootmuxSysPll1Div8
ARM QSPI Clock from SYSTEM PLL1 divided by 8.

enum _clock_rootmux_ecspi_clk_sel
Root clock select enumeration for ECSPI peripheral.

Values:

enumerator kCLOCK_EcspiRootmuxOsc24M
ECSPI Clock from OSC 24M.

enumerator kCLOCK_EcspiRootmuxSysPll2Div5
ECSPI Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_EcspiRootmuxSysPll1Div20
ECSPI Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_EcspiRootmuxSysPll1Div5
ECSPI Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_EcspiRootmuxSysPll1
ECSPI Clock from SYSTEM PLL1.

enumerator kCLOCK_EcspiRootmuxSysPll3
ECSPI Clock from SYSTEM PLL3.

enumerator kCLOCK_EcspiRootmuxSysPll2Div4
ECSPI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_EcspiRootmuxAudioPll2
ECSPI Clock from AUDIO PLL2.

enum _clock_rootmux_enet_axi_clk_sel

Root clock select enumeration for ENET AXI bus.

Values:

enumerator kCLOCK_EnetAxiRootmuxOsc24M

ENET AXI Clock from OSC 24M.

enumerator kCLOCK_EnetAxiRootmuxSysPll1Div3

ENET AXI Clock from SYSTEM PLL1 divided by 3.

enumerator kCLOCK_EnetAxiRootmuxSysPll1

ENET AXI Clock from SYSTEM PLL1.

enumerator kCLOCK_EnetAxiRootmuxSysPll2Div4

ENET AXI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_EnetAxiRootmuxSysPll2Div5

ENET AXI Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_EnetAxiRootmuxAudioPll1

ENET AXI Clock from AUDIO PLL1.

enumerator kCLOCK_EnetAxiRootmuxVideoPll1

ENET AXI Clock from VIDEO PLL1.

enumerator kCLOCK_EnetAxiRootmuxSysPll3

ENET AXI Clock from SYSTEM PLL3.

enum _clock_rootmux_enet_ref_clk_sel

Root clock select enumeration for ENET REF Clcok.

Values:

enumerator kCLOCK_EnetRefRootmuxOsc24M

ENET REF Clock from OSC 24M.

enumerator kCLOCK_EnetRefRootmuxSysPll2Div8

ENET REF Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_EnetRefRootmuxSysPll2Div20

ENET REF Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK_EnetRefRootmuxSysPll2Div10

ENET REF Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_EnetRefRootmuxSysPll1Div5

ENET REF Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_EnetRefRootmuxAudioPll1

ENET REF Clock from AUDIO PLL1.

enumerator kCLOCK_EnetRefRootmuxVideoPll1

ENET REF Clock from VIDEO PLL1.

enumerator kCLOCK_EnetRefRootmuxExtClk4

ENET REF Clock from External Clock 4.

enum _clock_rootmux_enet_timer_clk_sel

Root clock select enumeration for ENET TIMER Clcok.

Values:

enumerator kCLOCK_EnetTimerRootmuxOsc24M
ENET TIMER Clock from OSC 24M.

enumerator kCLOCK_EnetTimerRootmuxSysPll2Div10
ENET TIMER Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_EnetTimerRootmuxAudioPll1
ENET TIMER Clock from AUDIO PLL1.

enumerator kCLOCK_EnetTimerRootmuxExtClk1
ENET TIMER Clock from External Clock 1.

enumerator kCLOCK_EnetTimerRootmuxExtClk2
ENET TIMER Clock External Clock 2.

enumerator kCLOCK_EnetTimerRootmuxExtClk3
ENET TIMER Clock from External Clock 3.

enumerator kCLOCK_EnetTimerRootmuxExtClk4
ENET TIMER Clock from External Clock 4.

enumerator kCLOCK_EnetTimerRootmuxVideoPll1
ENET TIMER Clock from VIDEO PLL1.

enum __clock_rootmux_enet_phy_clk_sel
Root clock select enumeration for ENET PHY Clcok.

Values:

enumerator kCLOCK_EnetPhyRootmuxOsc24M
ENET PHY Clock from OSC 24M.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div20
ENET PHY Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div8
ENET PHY Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div5
ENET PHY Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div2
ENET PHY Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_EnetPhyRootmuxAudioPll1
ENET PHY Clock from AUDIO PLL1.

enumerator kCLOCK_EnetPhyRootmuxVideoPll1
ENET PHY Clock from VIDEO PLL1.

enumerator kCLOCK_EnetPhyRootmuxAudioPll2
ENET PHY Clock from AUDIO PLL2.

enum __clock_rootmux_i2c_clk_sel
Root clock select enumeration for I2C peripheral.

Values:

enumerator kCLOCK_I2cRootmuxOsc24M
I2C Clock from OSC 24M.

enumerator kCLOCK_I2cRootmuxSysPll1Div5
I2C Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_I2cRootmuxSysPll2Div20
I2C Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK_I2cRootmuxSysPll3
I2C Clock from SYSTEM PLL3 .

enumerator kCLOCK_I2cRootmuxAudioPll1
I2C Clock from AUDIO PLL1.

enumerator kCLOCK_I2cRootmuxVideoPll1
I2C Clock from VIDEO PLL1.

enumerator kCLOCK_I2cRootmuxAudioPll2
I2C Clock from AUDIO PLL2.

enumerator kCLOCK_I2cRootmuxSysPll1Div6
I2C Clock from SYSTEM PLL1 divided by 6.

enum __clock_rootmux_uart_clk_sel
Root clock select enumeration for UART peripheral.

Values:

enumerator kCLOCK_UartRootmuxOsc24M
UART Clock from OSC 24M.

enumerator kCLOCK_UartRootmuxSysPll1Div10
UART Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_UartRootmuxSysPll2Div5
UART Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_UartRootmuxSysPll2Div10
UART Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_UartRootmuxSysPll3
UART Clock from SYSTEM PLL3.

enumerator kCLOCK_UartRootmuxExtClk2
UART Clock from External Clock 2.

enumerator kCLOCK_UartRootmuxExtClk34
UART Clock from External Clock 3, External Clock 4.

enumerator kCLOCK_UartRootmuxAudioPll2
UART Clock from Audio PLL2.

enum __clock_rootmux_gpt
Root clock select enumeration for GPT peripheral.

Values:

enumerator kCLOCK_GptRootmuxOsc24M
GPT Clock from OSC 24M.

enumerator kCLOCK_GptRootmuxSystemPll2Div10
GPT Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_GptRootmuxSysPll1Div2
GPT Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK_GptRootmuxSysPll1Div20
GPT Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_GptRootmuxVideoPll1
GPT Clock from VIDEO PLL1.

enumerator kCLOCK_GptRootmuxSystemPll1Div10
GPT Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_GptRootmuxAudioPll1
GPT Clock from AUDIO PLL1.

enumerator kCLOCK_GptRootmuxExtClk123
GPT Clock from External Clock1, External Clock2, External Clock3.

enum _clock_rootmux_wdog_clk_sel
Root clock select enumeration for WDOG peripheral.

Values:

enumerator kCLOCK_WdogRootmuxOsc24M
WDOG Clock from OSC 24M.

enumerator kCLOCK_WdogRootmuxSysPll1Div6
WDOG Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK_WdogRootmuxSysPll1Div5
WDOG Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_WdogRootmuxVpuPll
WDOG Clock from VPU DLL.

enumerator kCLOCK_WdogRootmuxSystemPll2Div8
WDOG Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_WdogRootmuxSystemPll3
WDOG Clock from SYSTEM PLL3.

enumerator kCLOCK_WdogRootmuxSystemPll1Div10
WDOG Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_WdogRootmuxSystemPll2Div6
WDOG Clock from SYSTEM PLL2 divided by 6.

enum _clock_rootmux_pwm_clk_sel
Root clock select enumeration for PWM peripheral.

Values:

enumerator kCLOCK_PwmRootmuxOsc24M
PWM Clock from OSC 24M.

enumerator kCLOCK_PwmRootmuxSysPll2Div10
PWM Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_PwmRootmuxSysPll1Div5
PWM Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_PwmRootmuxSysPll1Div20
PWM Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_PwmRootmuxSystemPll3
PWM Clock from SYSTEM PLL3.

enumerator kCLOCK_PwmRootmuxExtClk12
PWM Clock from External Clock1, External Clock2.

enumerator kCLOCK_PwmRootmuxSystemPll1Div10
PWM Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_PwmRootmuxVideoPll1
PWM Clock from VIDEO PLL1.

enum _clock_rootmux_sai_clk_sel
Root clock select enumeration for SAI peripheral.

Values:

enumerator kCLOCK_SaiRootmuxOsc24M
SAI Clock from OSC 24M.

enumerator kCLOCK_SaiRootmuxAudioPll1
SAI Clock from AUDIO PLL1.

enumerator kCLOCK_SaiRootmuxAudioPll2
SAI Clock from AUDIO PLL2.

enumerator kCLOCK_SaiRootmuxVideoPll1
SAI Clock from VIDEO PLL1.

enumerator kCLOCK_SaiRootmuxSysPll1Div6
SAI Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK_SaiRootmuxOsc26m
SAI Clock from OSC HDMI 26M.

enumerator kCLOCK_SaiRootmuxExtClk1
SAI Clock from External Clock1, External Clock2, External Clock3.

enumerator kCLOCK_SaiRootmuxExtClk2
SAI Clock from External Clock2, External Clock3, External Clock4.

enum _clock_rootmux_pdm_clk_sel
Root clock select enumeration for PDM peripheral.

Values:

enumerator kCLOCK_PdmRootmuxOsc24M
GPT Clock from OSC 24M.

enumerator kCLOCK_PdmRootmuxSystemPll2
GPT Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_PdmRootmuxAudioPll1
GPT Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK_PdmRootmuxSysPll1
GPT Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_PdmRootmuxSysPll2
GPT Clock from VIDEO PLL1.

enumerator kCLOCK_PdmRootmuxSysPll3
GPT Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_PdmRootmuxExtClk3
GPT Clock from AUDIO PLL1.

enumerator kCLOCK_PdmRootmuxAudioPll2
GPT Clock from External Clock1, External Clock2, External Clock3.

enum _clock_rootmux_noc_clk_sel

Root clock select enumeration for NOC CLK.

Values:

enumerator kCLOCK_NocRootmuxOsc24M
NOC Clock from OSC 24M.

enumerator kCLOCK_NocRootmuxSysPll1
NOC Clock from SYSTEM PLL1.

enumerator kCLOCK_NocRootmuxSysPll3
NOC Clock from SYSTEM PLL3.

enumerator kCLOCK_NocRootmuxSysPll2
NOC Clock from SYSTEM PLL2.

enumerator kCLOCK_NocRootmuxSysPll2Div2
NOC Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_NocRootmuxAudioPll1
NOC Clock from AUDIO PLL1.

enumerator kCLOCK_NocRootmuxVideoPll1
NOC Clock from VIDEO PLL1.

enumerator kCLOCK_NocRootmuxAudioPll2
NOC Clock from AUDIO PLL2.

enum _clock_pll_gate

CCM PLL gate control.

Values:

enumerator kCLOCK_ArmPllGate
ARM PLL Gate.

enumerator kCLOCK_GpuPllGate
GPU PLL Gate.

enumerator kCLOCK_VpuPllGate
VPU PLL Gate.

enumerator kCLOCK_DramPllGate
DRAM PLL1 Gate.

enumerator kCLOCK_SysPll1Gate
SYSTEM PLL1 Gate.

enumerator kCLOCK_SysPll1Div2Gate
SYSTEM PLL1 Div2 Gate.

enumerator kCLOCK_SysPll1Div3Gate
SYSTEM PLL1 Div3 Gate.

enumerator kCLOCK_SysPll1Div4Gate
SYSTEM PLL1 Div4 Gate.

enumerator kCLOCK_SysPll1Div5Gate
SYSTEM PLL1 Div5 Gate.

enumerator kCLOCK_SysPll1Div6Gate
SYSTEM PLL1 Div6 Gate.

enumerator kCLOCK_SysPll1Div8Gate
SYSTEM PLL1 Div8 Gate.

enumerator kCLOCK_SysPll1Div10Gate
SYSTEM PLL1 Div10 Gate.

enumerator kCLOCK_SysPll1Div20Gate
SYSTEM PLL1 Div20 Gate.

enumerator kCLOCK_SysPll2Gate
SYSTEM PLL2 Gate.

enumerator kCLOCK_SysPll2Div2Gate
SYSTEM PLL2 Div2 Gate.

enumerator kCLOCK_SysPll2Div3Gate
SYSTEM PLL2 Div3 Gate.

enumerator kCLOCK_SysPll2Div4Gate
SYSTEM PLL2 Div4 Gate.

enumerator kCLOCK_SysPll2Div5Gate
SYSTEM PLL2 Div5 Gate.

enumerator kCLOCK_SysPll2Div6Gate
SYSTEM PLL2 Div6 Gate.

enumerator kCLOCK_SysPll2Div8Gate
SYSTEM PLL2 Div8 Gate.

enumerator kCLOCK_SysPll2Div10Gate
SYSTEM PLL2 Div10 Gate.

enumerator kCLOCK_SysPll2Div20Gate
SYSTEM PLL2 Div20 Gate.

enumerator kCLOCK_SysPll3Gate
SYSTEM PLL3 Gate.

enumerator kCLOCK_AudioPll1Gate
AUDIO PLL1 Gate.

enumerator kCLOCK_AudioPll2Gate
AUDIO PLL2 Gate.

enumerator kCLOCK_VideoPll1Gate
VIDEO PLL1 Gate.

enumerator kCLOCK_VideoPll2Gate
VIDEO PLL2 Gate.

enum __clock_gate_value
CCM gate control value.

Values:

enumerator kCLOCK_ClockNotNeeded
Clock always disabled.

enumerator kCLOCK_ClockNeededRun
Clock enabled when CPU is running.

enumerator kCLOCK_ClockNeededRunWait
Clock enabled when CPU is running or in WAIT mode.

enumerator kCLOCK_ClockNeededAll
Clock always enabled.

enum _clock_pll_bypass_ctrl
PLL control names for PLL bypass.

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: bypass bit shift.

Values:

enumerator kCLOCK_AudioPll1BypassCtrl
CCM Audio PLL1 bypass Control.

enumerator kCLOCK_AudioPll2BypassCtrl
CCM Audio PLL2 bypass Control.

enumerator kCLOCK_VideoPll1BypassCtrl
CCM Video Pll1 bypass Control.

enumerator kCLOCK_DramPllInternalPll1BypassCtrl
CCM DRAM PLL bypass Control.

enumerator kCLOCK_ArmPllPwrBypassCtrl
CCM Arm PLL bypass Control.

enumerator kCLOCK_SysPll1InternalPll1BypassCtrl
CCM System PLL1 bypass Control.

enumerator kCLOCK_SysPll2InternalPll1BypassCtrl
CCM System PLL2 bypass Control.

enumerator kCLOCK_SysPll3InternalPll1BypassCtrl
CCM System PLL3 bypass Control.

enum _ccm_analog_pll_clke
PLL clock names for clock enable/disable settings.

These constants define the PLL clock names for PLL clock enable/disable operations.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Clock enable bit shift.

Values:

enumerator kCLOCK_AudioPll1Clke
Audio pll1 clke

enumerator kCLOCK_AudioPll2Clke
Audio pll2 clke

enumerator kCLOCK_VideoPll1Clke
Video pll1 clke

enumerator kCLOCK_DramPllClke
Dram pll clke

enumerator kCLOCK_ArmPllClke
Arm pll clke

enumerator kCLOCK_SystemPll1Clke
System pll1 clke

enumerator kCLOCK_SystemPll1Div2Clke
System pll1 Div2 clke

enumerator kCLOCK_SystemPll1Div3Clke
System pll1 Div3 clke

enumerator kCLOCK_SystemPll1Div4Clke
System pll1 Div4 clke

enumerator kCLOCK_SystemPll1Div5Clke
System pll1 Div5 clke

enumerator kCLOCK_SystemPll1Div6Clke
System pll1 Div6 clke

enumerator kCLOCK_SystemPll1Div8Clke
System pll1 Div8 clke

enumerator kCLOCK_SystemPll1Div10Clke
System pll1 Div10 clke

enumerator kCLOCK_SystemPll1Div20Clke
System pll1 Div20 clke

enumerator kCLOCK_SystemPll2Clke
System pll2 clke

enumerator kCLOCK_SystemPll2Div2Clke
System pll2 Div2 clke

enumerator kCLOCK_SystemPll2Div3Clke
System pll2 Div3 clke

enumerator kCLOCK_SystemPll2Div4Clke
System pll2 Div4 clke

enumerator kCLOCK_SystemPll2Div5Clke
System pll2 Div5 clke

enumerator kCLOCK_SystemPll2Div6Clke
System pll2 Div6 clke

enumerator kCLOCK_SystemPll2Div8Clke
System pll2 Div8 clke

enumerator kCLOCK_SystemPll2Div10Clke
System pll2 Div10 clke

enumerator kCLOCK_SystemPll2Div20Clke
System pll2 Div20 clke

enumerator kCLOCK_SystemPll3Clke
System pll3 clke

enum _clock_pll_ctrl
ANALOG Power down override control.

Values:

enumerator kCLOCK_AudioPll1Ctrl

enumerator kCLOCK_AudioPll2Ctrl

enumerator kCLOCK_VideoPll1Ctrl

enumerator kCLOCK_DramPllCtrl

enumerator kCLOCK_ArmPllCtrl

enumerator kCLOCK_SystemPll1Ctrl

enumerator kCLOCK_SystemPll2Ctrl

enumerator kCLOCK_SystemPll3Ctrl

PLL reference clock select.

Values:

enumerator kANALOG_PllRefOsc24M
reference OSC 24M

enumerator kANALOG_PllPadClk
reference PAD CLK

typedef enum *_clock_name* clock_name_t
Clock name used to get clock frequency.

typedef enum *_clock_ip_name* clock_ip_name_t
CCM CCGR gate control.

typedef enum *_clock_root_control* clock_root_control_t
ccm root name used to get clock frequency.

typedef enum *_clock_root* clock_root_t
ccm clock root index used to get clock frequency.

typedef enum *_clock_rootmux_m7_clk_sel* clock_rootmux_m7_clk_sel_t
Root clock select enumeration for ARM Cortex-M7 core.

typedef enum *_clock_rootmux_axi_clk_sel* clock_rootmux_axi_clk_sel_t
Root clock select enumeration for AXI bus.

typedef enum *_clock_rootmux_ahb_clk_sel* clock_rootmux_ahb_clk_sel_t
Root clock select enumeration for AHB bus.

typedef enum *_clock_rootmux_audio_ahb_clk_sel* clock_rootmux_audio_ahb_clk_sel_t
Root clock select enumeration for Audio AHB bus.

typedef enum *_clock_rootmux_qspi_clk_sel* clock_rootmux_qspi_clk_sel_t
Root clock select enumeration for QSPI peripheral.

typedef enum *_clock_rootmux_ecspi_clk_sel* clock_rootmux_ecspi_clk_sel_t
Root clock select enumeration for ECSPI peripheral.

typedef enum *_clock_rootmux_enet_axi_clk_sel* clock_rootmux_enet_axi_clk_sel_t
Root clock select enumeration for ENET AXI bus.

typedef enum *_clock_rootmux_enet_ref_clk_sel* clock_rootmux_enet_ref_clk_sel_t
Root clock select enumeration for ENET REF Clcok.

typedef enum *_clock_rootmux_enet_timer_clk_sel* clock_rootmux_enet_timer_clk_sel_t
Root clock select enumeration for ENET TIMER Clcok.

typedef enum *_clock_rootmux_enet_phy_clk_sel* clock_rootmux_enet_phy_clk_sel_t
Root clock select enumeration for ENET PHY Clcok.

typedef enum *_clock_rootmux_i2c_clk_sel* clock_rootmux_i2c_clk_sel_t
Root clock select enumeration for I2C peripheral.

typedef enum *_clock_rootmux_uart_clk_sel* clock_rootmux_uart_clk_sel_t
Root clock select enumeration for UART peripheral.

typedef enum *_clock_rootmux_gpt* clock_rootmux_gpt_t
Root clock select enumeration for GPT peripheral.

typedef enum *_clock_rootmux_wdog_clk_sel* clock_rootmux_wdog_clk_sel_t
Root clock select enumeration for WDOG peripheral.

typedef enum *_clock_rootmux_pwm_clk_sel* clock_rootmux_Pwm_clk_sel_t
Root clock select enumeration for PWM peripheral.

typedef enum *_clock_rootmux_sai_clk_sel* clock_rootmux_sai_clk_sel_t
Root clock select enumeration for SAI peripheral.

typedef enum *_clock_rootmux_pdm_clk_sel* clock_rootmux_pdm_clk_sel_t
Root clock select enumeration for PDM peripheral.

typedef enum *_clock_rootmux_noc_clk_sel* clock_rootmux_noc_clk_sel_t
Root clock select enumeration for NOC CLK.

typedef enum *_clock_pll_gate* clock_pll_gate_t
CCM PLL gate control.

typedef enum *_clock_gate_value* clock_gate_value_t
CCM gate control value.

typedef enum *_clock_pll_bypass_ctrl* clock_pll_bypass_ctrl_t
PLL control names for PLL bypass.

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: bypass bit shift.

typedef enum *_ccm_analog_pll_clke* clock_pll_clke_t
PLL clock names for clock enable/disable settings.

These constants define the PLL clock names for PLL clock enable/disable operations.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Clock enable bit shift.

typedef enum *_clock_pll_ctrl* clock_pll_ctrl_t
ANALOG Power down override control.

typedef struct *_ccm_analog_frac_pll_config* ccm_analog_frac_pll_config_t
Fractional-N PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

typedef struct *_ccm_analog_integer_pll_config* ccm_analog_integer_pll_config_t
Integer PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

FSL_CLOCK_DRIVER_VERSION
CLOCK driver version 2.4.0.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

OSC24M_CLK_FREQ
: XTAL 24M clock frequency.

CLKPAD_FREQ
: pad clock frequency.

ECSPI_CLOCKS
: Clock ip name array for ECSPI.

ENET_CLOCKS
: Clock ip name array for ENET.

GPIO_CLOCKS
: Clock ip name array for GPIO.

GPT_CLOCKS
: Clock ip name array for GPT.

I2C_CLOCKS
: Clock ip name array for I2C.

IOMUX_CLOCKS
: Clock ip name array for IOMUX.

IPMUX_CLOCKS
: Clock ip name array for IPMUX.

PWM_CLOCKS
: Clock ip name array for PWM.

RDC_CLOCKS
: Clock ip name array for RDC.

SAI_CLOCKS
: Clock ip name array for SAI.

RDC_SEMA42_CLOCKS
: Clock ip name array for RDC SEMA42.

UART_CLOCKS
: Clock ip name array for UART.

USDHC_CLOCKS
: Clock ip name array for USDHC.

WDOG_CLOCKS
: Clock ip name array for WDOG.

TMU_CLOCKS
: Clock ip name array for TEMPSENSOR.

SDMA_CLOCKS
: Clock ip name array for SDMA.

MU_CLOCKS
: Clock ip name array for MU.

QSPI_CLOCKS
: Clock ip name array for QSPI.

PDM_CLOCKS
: Clock ip name array for PDM.

ASRC_CLOCKS
: Clock ip name array for ASRC.

CCM_BIT_FIELD_EXTRACTION(val, mask, shift)
: CCM reg macros to extract corresponding registers bit field.

CCM_REG_OFF(root, off)
: CCM reg macros to map corresponding registers.

CCM_REG(root)

CCM_REG_SET(root)

CCM_REG_CLR(root)

AUDIO_PLL1_GEN_CTRL_OFFSET
: CCM Analog registers offset.

AUDIO_PLL2_GEN_CTRL_OFFSET

VIDEO_PLL1_GEN_CTRL_OFFSET

GPU_PLL_GEN_CTRL_OFFSET

VPU_PLL_GEN_CTRL_OFFSET

ARM_PLL_GEN_CTRL_OFFSET

SYS_PLL1_GEN_CTRL_OFFSET

SYS_PLL2_GEN_CTRL_OFFSET

SYS_PLL3_GEN_CTRL_OFFSET

DRAM_PLL_GEN_CTRL_OFFSET

CCM_ANALOG_TUPLE(reg, shift)
: CCM ANALOG tuple macros to map corresponding registers and bit fields.

CCM_ANALOG_TUPLE_SHIFT(tuple)

CCM_ANALOG_TUPLE_REG_OFF(base, tuple, off)

CCM_ANALOG_TUPLE_REG(base, tuple)

CCM_TUPLE(ccgr, root)
: CCM CCGR and root tuple.

CCM_TUPLE_CCGR(tuple)

CCM_TUPLE_ROOT(tuple)

CLOCK_ROOT_SOURCE
: clock root source

CLOCK_ROOT_CONTROL_TUPLE

kCLOCK_CoreSysClk
: For compatible with other platforms without CCM.

CLOCK_GetCoreSysClkFreq
: For compatible with other platforms without CCM.

static inline void CLOCK_SetRootMux(*clock_root_control_t* rootClk, uint32_t mux)

Set clock root mux. User maybe need to set more than one mux ROOT according to the clock tree description in the reference manual.

Parameters

- rootClk – Root clock control (see clock_root_control_t enumeration).
- mux – Root mux value, refer to _ccm_rootmux_xxx enumeration.

static inline uint32_t CLOCK_GetRootMux(*clock_root_control_t* rootClk)

Get clock root mux. In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

- rootClk – Root clock control (see clock_root_control_t enumeration).

Returns

Root mux value, refer to _ccm_rootmux_xxx enumeration.

static inline void CLOCK_EnableRoot(*clock_root_control_t* rootClk)

Enable clock root.

Parameters

- rootClk – Root clock control (see clock_root_control_t enumeration)

static inline void CLOCK_DisableRoot(*clock_root_control_t* rootClk)

Disable clock root.

Parameters

- rootClk – Root control (see clock_root_control_t enumeration)

static inline bool CLOCK_IsRootEnabled(*clock_root_control_t* rootClk)

Check whether clock root is enabled.

Parameters

- rootClk – Root control (see clock_root_control_t enumeration)

Returns

CCM root enabled or not.

- true: Clock root is enabled.
- false: Clock root is disabled.

void CLOCK_UpdateRoot(*clock_root_control_t* ccmRootClk, uint32_t mux, uint32_t pre, uint32_t post)

Update clock root in one step, for dynamical clock switching Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.

Parameters

- ccmRootClk – Root control (see clock_root_control_t enumeration)
- mux – Root mux value, refer to _ccm_rootmux_xxx enumeration
- pre – Pre divider value (0-7, divider=n+1)
- post – Post divider value (0-63, divider=n+1)

void CLOCK_SetRootDivider(*clock_root_control_t* ccmRootClk, uint32_t pre, uint32_t post)

Set root clock divider Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.

Parameters

- ccmRootClk – Root control (see clock_root_control_t enumeration)
- pre – Pre divider value (1-8)
- post – Post divider value (1-64)

static inline uint32_t CLOCK_GetRootPreDivider(*clock_root_control_t* rootClk)

Get clock root PRE_PODF. In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

- rootClk – Root clock name (see clock_root_control_t enumeration).

Returns

Root Pre divider value.

static inline uint32_t CLOCK_GetRootPostDivider(*clock_root_control_t* rootClk)

Get clock root POST_PODF. In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

- rootClk – Root clock name (see clock_root_control_t enumeration).

Returns

Root Post divider value.

static inline void CLOCK_ControlGate(uintptr_t ccmGate, *clock_gate_value_t* control)

lockrief Set PLL or CCGR gate control

Parameters

- ccmGate – Gate control (see clock_pll_gate_t and clock_ip_name_t enumeration)
- control – Gate control value (see clock_gate_value_t)

void CLOCK_EnableClock(*clock_ip_name_t* ccmGate)

Enable CCGR clock gate and root clock gate for each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual. Take care of that one module may need to set more than one clock gate.

Parameters

- ccmGate – Gate control for each module (see clock_ip_name_t enumeration).

void CLOCK_DisableClock(*clock_ip_name_t* ccmGate)

Disable CCGR clock gate for the each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual. Take care of that one module may need to set more than one clock gate.

Parameters

- ccmGate – Gate control for each module (see clock_ip_name_t enumeration).

static inline void CLOCK_PowerUpPll(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* pllControl)

Power up PLL.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_ctrl_t enumeration)

static inline void CLOCK_PowerDownPll(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* pllControl)

Power down PLL.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_ctrl_t enumeration)

static inline void CLOCK_SetPllBypass(CCM_ANALOG_Type *base, *clock_pll_bypass_ctrl_t* pllControl, bool bypass)

PLL bypass setting.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name, refer to ccm_analog_pll_control_t enumeration
- bypass – Bypass the PLL.
  - true: Bypass the PLL.
  - false: Do not bypass the PLL.

static inline bool CLOCK_IsPllBypassed(CCM_ANALOG_Type *base, *clock_pll_bypass_ctrl_t* pllControl)

Check if PLL is bypassed.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name, refer to ccm_analog_pll_control_t enumeration

Returns

PLL bypass status.

- true: The PLL is bypassed.
- false: The PLL is not bypassed.

static inline bool CLOCK_IsPllLocked(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* pllControl)

Check if PLL clock is locked.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_ctrl_t enumeration)

Returns

PLL lock status.

- true: The PLL clock is locked.
- false: The PLL clock is not locked.

static inline void CLOCK_EnableAnalogClock(CCM_ANALOG_Type *base, *clock_pll_clke_t* pllClock)

Enable PLL clock.

Parameters

- base – CCM_ANALOG base pointer.
- pllClock – PLL clock name, refer to ccm_analog_pll_clock_t enumeration

static inline void CLOCK_DisableAnalogClock(CCM_ANALOG_Type *base, *clock_pll_clke_t* pllClock)

Disable PLL clock.

Parameters

- base – CCM_ANALOG base pointer.
- pllClock – PLL clock name, refer to ccm_analog_pll_clock_t enumeration

static inline void CLOCK_OverridePllClke(CCM_ANALOG_Type *base, *clock_pll_clke_t* ovClock, bool override)

Override PLL clock output enable.

Parameters

- base – CCM_ANALOG base pointer.
- ovClock – PLL clock name (see clock_pll_clke_t enumeration)
- override – Override the PLL.
  - true: Override the PLL clke, CCM will handle it.
  - false: Do not override the PLL clke.

static inline void CLOCK_OverridePllPd(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* pdClock, bool override)

Override PLL power down.

Parameters

- base – CCM_ANALOG base pointer.
- pdClock – PLL clock name (see clock_pll_ctrl_t enumeration)
- override – Override the PLL.
  - true: Override the PLL clke, CCM will handle it.
  - false: Do not override the PLL clke.

void CLOCK_InitArmPll(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG ARM PLL.

Note: This function can't detect whether the Arm PLL has been enabled and used by some IPs.

Parameters

- config – Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).

void CLOCK_DeinitArmPll(void)

De-initialize the ARM PLL.

void CLOCK_InitSysPll1(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG SYS PLL1.

Note: This function can't detect whether the SYS PLL has been enabled and used by some IPs.

Parameters

- config – Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).

void CLOCK_DeinitSysPll1(void)

De-initialize the System PLL1.

void CLOCK_InitSysPll2(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG SYS PLL2.

Note: This function can't detect whether the SYS PLL has been enabled and used by some IPs.

Parameters

- config – Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).

void CLOCK_DeinitSysPll2(void)

De-initialize the System PLL2.

void CLOCK_InitSysPll3(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG SYS PLL3.

Note: This function can't detect whether the SYS PLL has been enabled and used by some IPs.

Parameters

- config – Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).

void CLOCK_DeinitSysPll3(void)

De-initialize the System PLL3.

void CLOCK_InitAudioPll1(const *ccm_analog_frac_pll_config_t* *config)

Initializes the ANALOG AUDIO PLL1.

Note: This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

Parameters

- config – Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).

void CLOCK_DeinitAudioPll1(void)

De-initialize the Audio PLL1.

void CLOCK_InitAudioPll2(const *ccm_analog_frac_pll_config_t* *config)

Initializes the ANALOG AUDIO PLL2.

Note: This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

Parameters

- config – Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).

void CLOCK_DeinitAudioPll2(void)

De-initialize the Audio PLL2.

void CLOCK_InitVideoPll1(const *ccm_analog_frac_pll_config_t* *config)

Initializes the ANALOG VIDEO PLL1.

Parameters

- config – Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).

void CLOCK_DeinitVideoPll1(void)

De-initialize the Video PLL1.

void CLOCK_InitIntegerPll(CCM_ANALOG_Type *base, const *ccm_analog_integer_pll_config_t* *config, *clock_pll_ctrl_t* type)

Initializes the ANALOG Integer PLL.

Parameters

- base – CCM ANALOG base address
- config – Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).
- type – integer pll type

uint32_t CLOCK_GetIntegerPllFreq(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* type, uint32_t refClkFreq, bool pll1Bypass)

Get the ANALOG Integer PLL clock frequency.

Parameters

- base – CCM ANALOG base address.
- type – integer pll type
- refClkFreq – pll reference clock frequency
- pll1Bypass – pll1 bypass flag

Returns

Clock frequency

void CLOCK_InitFracPll(CCM_ANALOG_Type *base, const *ccm_analog_frac_pll_config_t* *config, *clock_pll_ctrl_t* type)

Initializes the ANALOG Fractional PLL.

Parameters

- base – CCM ANALOG base address.
- config – Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).
- type – fractional pll type.

uint32_t CLOCK_GetFracPllFreq(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* type, uint32_t refClkFreq)

Gets the ANALOG Fractional PLL clock frequency.

Parameters

- base – CCM_ANALOG base pointer.
- type – Fractional pll type.

- refClkFreq – Pll reference clock frequency

Returns
Clock frequency

uint32_t CLOCK_GetPllFreq(*clock_pll_ctrl_t* pll)

Gets PLL clock frequency.

Parameters

- pll – Fractional pll type.

Returns
Clock frequency

uint32_t CLOCK_GetPllRefClkFreq(*clock_pll_ctrl_t* ctrl)

Gets PLL reference clock frequency.

Parameters

- ctrl – Fractional pll type.

Returns
Clock frequency

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Gets the clock frequency for a specific clock name.

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

Parameters

- clockName – Clock names defined in clock_name_t

Returns
Clock frequency value in hertz

uint32_t CLOCK_GetClockRootFreq(*clock_root_t* clockRoot)

Gets the frequency of selected clock root.

Parameters

- clockRoot – The clock root used to get the frequency, please refer to clock_root_t.

Returns
The frequency of selected clock root.

uint32_t CLOCK_GetCoreM7Freq(void)

Get the CCM Cortex M7 core frequency.

Returns
Clock frequency; If the clock is invalid, returns 0.

uint32_t CLOCK_GetAxiFreq(void)

Get the CCM Axi bus frequency.

Returns
Clock frequency; If the clock is invalid, returns 0.

uint32_t CLOCK_GetAhbFreq(void)

Get the CCM Ahb bus frequency.

Returns
Clock frequency; If the clock is invalid, returns 0.

uint32_t CLOCK_GetEnetAxiFreq(void)

brief Get the CCM Enet AXI bus frequency.

return Clock frequency; If the clock is invalid, returns 0.

uint8_t refSel

pll reference clock sel

uint32_t mainDiv

Value of the 10-bit programmable main-divider, range must be 64~1023

uint32_t dsm

Value of 16-bit DSM

uint8_t preDiv

Value of the 6-bit programmable pre-divider, range must be 1~63

uint8_t postDiv

Value of the 3-bit programmable Scaler, range must be 0~6

uint8_t refSel

pll reference clock sel

uint32_t mainDiv

Value of the 10-bit programmable main-divider, range must be 64~1023

uint8_t preDiv

Value of the 6-bit programmable pre-divider, range must be 1~63

uint8_t postDiv

Value of the 3-bit programmable Scaler, range must be 0~6

struct _ccm_analog_frac_pll_config

#include <fsl_clock.h> Fractional-N PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

struct _ccm_analog_integer_pll_config

#include <fsl_clock.h> Integer PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

2.6 ECSPI: Enhanced Configurable Serial Peripheral Interface Driver

2.7 ECSPI Driver

void ECSPI_MasterGetDefaultConfig(*ecspi_master_config_t* *config)

Sets the ECSPI configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in ECSPI_MasterInit(). User may use the initialized structure unchanged in ECSPI_MasterInit, or modify some fields of the structure before calling ECSPI_MasterInit. After calling this API, the master is ready to transfer. Example:

```
ecspi_master_config_t config;
ECSPI_MasterGetDefaultConfig(&config);
```

Parameters

- config – pointer to config structure

void ECSPI_MasterInit(ECSPI_Type *base, const *ecspi_master_config_t* *config, uint32_t srcClock_Hz)

Initializes the ECSPI with configuration.

The configuration structure can be filled by user from scratch, or be set with default values by ECSPI_MasterGetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
ecspi_master_config_t config = {
.baudRate_Bps = 400000,
...
};
ECSPI_MasterInit(ECSPI0, &config);
```

Parameters

- base – ECSPI base pointer
- config – pointer to master configuration structure
- srcClock_Hz – Source clock frequency.

void ECSPI_SlaveGetDefaultConfig(*ecspi_slave_config_t* *config)

Sets the ECSPI configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in ECSPI_SlaveInit(). User may use the initialized structure unchanged in ECSPI_SlaveInit(), or modify some fields of the structure before calling ECSPI_SlaveInit(). After calling this API, the master is ready to transfer. Example:

```
ecspi_Slaveconfig_t config;
ECSPI_SlaveGetDefaultConfig(&config);
```

Parameters

- config – pointer to config structure

void ECSPI_SlaveInit(ECSPI_Type *base, const *ecspi_slave_config_t* *config)

Initializes the ECSPI with configuration.

The configuration structure can be filled by user from scratch, or be set with default values by ECSPI_SlaveGetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
ecspi_Salveconfig_t config = {
.baudRate_Bps = 400000,
...
};
ECSPI_SlaveInit(ECSPI1, &config);
```

Parameters

- base – ECSPI base pointer
- config – pointer to master configuration structure

void ECSPI_Deinit(ECSPI_Type *base)

De-initializes the ECSPI.

Calling this API resets the ECSPI module, gates the ECSPI clock. The ECSPI module can't work unless calling the ECSPI_MasterInit/ECSPI_SlaveInit to initialize module.

Parameters

- base – ECSPI base pointer

static inline void ECSPI_Enable(ECSPI_Type *base, bool enable)

Enables or disables the ECSPI.

Parameters

- base – ECSPI base pointer
- enable – pass true to enable module, false to disable module

static inline uint32_t ECSPI_GetStatusFlags(ECSPI_Type *base)

Gets the status flag.

Parameters

- base – ECSPI base pointer

Returns

ECSPI Status, use status flag to AND _ecspi_flags could get the related status.

static inline void ECSPI_ClearStatusFlags(ECSPI_Type *base, uint32_t mask)

Clear the status flag.

Parameters

- base – ECSPI base pointer
- mask – ECSPI Status, use status flag to AND _ecspi_flags could get the related status.

static inline void ECSPI_EnableInterrupts(ECSPI_Type *base, uint32_t mask)

Enables the interrupt for the ECSPI.

Parameters

- base – ECSPI base pointer
- mask – ECSPI interrupt source. The parameter can be any combination of the following values:
  - kECSPI_TxfifoEmptyInterruptEnable
  - kECSPI_TxFifoDataRequstInterruptEnable
  - kECSPI_TxFifoFullInterruptEnable
  - kECSPI_RxFifoReadyInterruptEnable
  - kECSPI_RxFifoDataRequstInterruptEnable
  - kECSPI_RxFifoFullInterruptEnable
  - kECSPI_RxFifoOverFlowInterruptEnable
  - kECSPI_TransferCompleteInterruptEnable
  - kECSPI_AllInterruptEnable

static inline void ECSPI_DisableInterrupts(ECSPI_Type *base, uint32_t mask)

Disables the interrupt for the ECSPI.

Parameters

- base – ECSPI base pointer
- mask – ECSPI interrupt source. The parameter can be any combination of the following values:
  - kECSPI_TxfifoEmptyInterruptEnable
  - kECSPI_TxFifoDataRequstInterruptEnable

- kECSPI_TxFifoFullInterruptEnable
- kECSPI_RxFifoReadyInterruptEnable
- kECSPI_RxFifoDataRequstInterruptEnable
- kECSPI_RxFifoFullInterruptEnable
- kECSPI_RxFifoOverFlowInterruptEnable
- kECSPI_TransferCompleteInterruptEnable
- kECSPI_AllInterruptEnable

static inline void ECSPI_SoftwareReset(ECSPI_Type *base)

Software reset.

Parameters

- base – ECSPI base pointer

static inline bool ECSPI_IsMaster(ECSPI_Type *base, *ecspi_channel_source_t* channel)

Mode check.

Parameters

- base – ECSPI base pointer
- channel – ECSPI channel source

Returns

mode of channel

static inline void ECSPI_EnableDMA(ECSPI_Type *base, uint32_t mask, bool enable)

Enables the DMA source for ECSPI.

Parameters

- base – ECSPI base pointer
- mask – ECSPI DMA source. The parameter can be any of the following values:
  - kECSPI_TxDmaEnable
  - kECSPI_RxDmaEnable
  - kECSPI_DmaAllEnable
- enable – True means enable DMA, false means disable DMA

static inline uint8_t ECSPI_GetTxFifoCount(ECSPI_Type *base)

Get the Tx FIFO data count.

Parameters

- base – ECSPI base pointer.

Returns

the number of words in Tx FIFO buffer.

static inline uint8_t ECSPI_GetRxFifoCount(ECSPI_Type *base)

Get the Rx FIFO data count.

Parameters

- base – ECSPI base pointer.

Returns

the number of words in Rx FIFO buffer.

static inline void ECSPI_SetChannelSelect(ECSPI_Type *base, *ecspi_channel_source_t* channel)

Set channel select for transfer.

Parameters

- base – ECSPI base pointer
- channel – Channel source.

void ECSPI_SetChannelConfig(ECSPI_Type *base, *ecspi_channel_source_t* channel, const *ecspi_channel_config_t* *config)

Set channel select configuration for transfer.

The purpose of this API is to set the channel will be use to transfer. User may use this API after instance has been initialized or before transfer start. The configuration structure *ecspi_channel_config* can be filled by user from scratch. After calling this API, user can select this channel as transfer channel.

Parameters

- base – ECSPI base pointer
- channel – Channel source.
- config – Configuration struct of channel

void ECSPI_SetBaudRate(ECSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the baud rate for ECSPI transfer. This is only used in master.

Parameters

- base – ECSPI base pointer
- baudRate_Bps – baud rate needed in Hz.
- srcClock_Hz – ECSPI source clock frequency in Hz.

status_t ECSPI_WriteBlocking(ECSPI_Type *base, const uint32_t *buffer, size_t size)

Sends a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – ECSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to send

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_ECSPI_Timeout – The transfer timed out and was aborted.

static inline void ECSPI_WriteData(ECSPI_Type *base, uint32_t data)

Writes a data into the ECSPI data register.

Parameters

- base – ECSPI base pointer
- data – Data needs to be write.

static inline uint32_t ECSPI_ReadData(ECSPI_Type *base)

Gets a data from the ECSPI data register.

Parameters

- base – ECSPI base pointer

Returns

Data in the register.

void ECSPI_MasterTransferCreateHandle(ECSPI_Type *base, *ecspi_master_handle_t* *handle, *ecspi_master_callback_t* callback, void *userData)

Initializes the ECSPI master handle.

This function initializes the ECSPI master handle which can be used for other ECSPI master transactional APIs. Usually, for a specified ECSPI instance, call this API once to get the initialized handle.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI handle pointer.
- callback – Callback function.
- userData – User data.

status_t ECSPI_MasterTransferBlocking(ECSPI_Type *base, *ecspi_transfer_t* *xfer)

Transfers a block of data using a polling method.

Parameters

- base – SPI base pointer
- xfer – pointer to spi_xfer_config_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_ECSPI_Timeout – The transfer timed out and was aborted.

status_t ECSPI_MasterTransferNonBlocking(ECSPI_Type *base, *ecspi_master_handle_t* *handle, *ecspi_transfer_t* *xfer)

Performs a non-blocking ECSPI interrupt transfer.

Note: The API immediately returns after transfer initialization is finished.

Note: If ECSPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

Parameters

- base – ECSPI peripheral base address.
- handle – pointer to ecspi_master_handle_t structure which stores the transfer state
- xfer – pointer to ecspi_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.

- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_ECSPI_Busy – ECSPI is not idle, is running another transfer.

status_t ECSPI_MasterTransferGetCount(ECSPI_Type *base, *ecspi_master_handle_t* *handle, size_t *count)

Gets the bytes of the ECSPI interrupt transferred.

Parameters

- base – ECSPI peripheral base address.
- handle – Pointer to ECSPI transfer handle, this should be a static variable.
- count – Transferred bytes of ECSPI master.

Return values

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

void ECSPI_MasterTransferAbort(ECSPI_Type *base, *ecspi_master_handle_t* *handle)

Aborts an ECSPI transfer using interrupt.

Parameters

- base – ECSPI peripheral base address.
- handle – Pointer to ECSPI transfer handle, this should be a static variable.

void ECSPI_MasterTransferHandleIRQ(ECSPI_Type *base, *ecspi_master_handle_t* *handle)

Interrupts the handler for the ECSPI.

Parameters

- base – ECSPI peripheral base address.
- handle – pointer to ecspi_master_handle_t structure which stores the transfer state.

void ECSPI_SlaveTransferCreateHandle(ECSPI_Type *base, *ecspi_slave_handle_t* *handle, *ecspi_slave_callback_t* callback, void *userData)

Initializes the ECSPI slave handle.

This function initializes the ECSPI slave handle which can be used for other ECSPI slave transactional APIs. Usually, for a specified ECSPI instance, call this API once to get the initialized handle.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI handle pointer.
- callback – Callback function.
- userData – User data.

static inline *status_t* ECSPI_SlaveTransferNonBlocking(ECSPI_Type *base, *ecspi_slave_handle_t* *handle, *ecspi_transfer_t* *xfer)

Performs a non-blocking ECSPI slave interrupt transfer.

Note: The API returns immediately after the transfer initialization is finished.

Parameters

- base – ECSPI peripheral base address.
- handle – pointer to ecspi_master_handle_t structure which stores the transfer state
- xfer – pointer to ecspi_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_ECSPI_Busy – ECSPI is not idle, is running another transfer.

static inline *status_t* ECSPI_SlaveTransferGetCount(ECSPI_Type *base, *ecspi_slave_handle_t* *handle, size_t *count)

Gets the bytes of the ECSPI interrupt transferred.

Parameters

- base – ECSPI peripheral base address.
- handle – Pointer to ECSPI transfer handle, this should be a static variable.
- count – Transferred bytes of ECSPI slave.

Return values

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

static inline void ECSPI_SlaveTransferAbort(ECSPI_Type *base, *ecspi_slave_handle_t* *handle)

Aborts an ECSPI slave transfer using interrupt.

Parameters

- base – ECSPI peripheral base address.
- handle – Pointer to ECSPI transfer handle, this should be a static variable.

void ECSPI_SlaveTransferHandleIRQ(ECSPI_Type *base, *ecspi_slave_handle_t* *handle)

Interrupts a handler for the ECSPI slave.

Parameters

- base – ECSPI peripheral base address.
- handle – pointer to ecspi_slave_handle_t structure which stores the transfer state

FSL_ECSPI_DRIVER_VERSION

ECSPI driver version.

Return status for the ECSPI driver.

Values:

enumerator kStatus_ECSPI_Busy

ECSPI bus is busy

enumerator kStatus_ECSPI_Idle

ECSPI is idle

enumerator kStatus_ECSPI_Error

ECSPI error

enumerator kStatus_ECSPI_HardwareOverFlow
ECSPI hardware overflow

enumerator kStatus_ECSPI_Timeout
ECSPI timeout polling status flags.

enum _ecspi_clock_polarity
ECSPI clock polarity configuration.

Values:

enumerator kECSPI_PolarityActiveHigh
Active-high ECSPI polarity high (idles low).

enumerator kECSPI_PolarityActiveLow
Active-low ECSPI polarity low (idles high).

enum _ecspi_clock_phase
ECSPI clock phase configuration.

Values:

enumerator kECSPI_ClockPhaseFirstEdge
First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kECSPI_ClockPhaseSecondEdge
First edge on SPSCK occurs at the start of the first cycle of a data transfer.

ECSPI interrupt sources.

Values:

enumerator kECSPI_TxfifoEmptyInterruptEnable
Transmit FIFO buffer empty interrupt

enumerator kECSPI_TxFifoDataRequstInterruptEnable
Transmit FIFO data requst interrupt

enumerator kECSPI_TxFifoFullInterruptEnable
Transmit FIFO full interrupt

enumerator kECSPI_RxFifoReadyInterruptEnable
Receiver FIFO ready interrupt

enumerator kECSPI_RxFifoDataRequstInterruptEnable
Receiver FIFO data requst interrupt

enumerator kECSPI_RxFifoFullInterruptEnable
Receiver FIFO full interrupt

enumerator kECSPI_RxFifoOverFlowInterruptEnable
Receiver FIFO buffer overflow interrupt

enumerator kECSPI_TransferCompleteInterruptEnable
Transfer complete interrupt

enumerator kECSPI_AllInterruptEnable
All interrupt

ECSPI status flags.

Values:

enumerator kECSPI_TxfifoEmptyFlag
Transmit FIFO buffer empty flag

enumerator kECSPI_TxFifoDataRequstFlag
Transmit FIFO data requst flag

enumerator kECSPI_TxFifoFullFlag
Transmit FIFO full flag

enumerator kECSPI_RxFifoReadyFlag
Receiver FIFO ready flag

enumerator kECSPI_RxFifoDataRequstFlag
Receiver FIFO data requst flag

enumerator kECSPI_RxFifoFullFlag
Receiver FIFO full flag

enumerator kECSPI_RxFifoOverFlowFlag
Receiver FIFO buffer overflow flag

enumerator kECSPI_TransferCompleteFlag
Transfer complete flag

ECSPI DMA enable.

Values:

enumerator kECSPI_TxDmaEnable
Tx DMA request source

enumerator kECSPI_RxDmaEnable
Rx DMA request source

enumerator kECSPI_DmaAllEnable
All DMA request source

enum _ecspi_data_ready
ECSPI SPI_RDY signal configuration.

Values:

enumerator kECSPI_DataReadyIgnore
SPI_RDY signal is ignored

enumerator kECSPI_DataReadyFallingEdge
SPI_RDY signal will be triggerd by the falling edge

enumerator kECSPI_DataReadyLowLevel
SPI_RDY signal will be triggerd by a low level

enum _ecspi_channel_source
ECSPI channel select source.

Values:

enumerator kECSPI_Channel0
Channel 0 is selectd

enumerator kECSPI_Channel1
Channel 1 is selectd

enumerator kECSPI_Channel2
Channel 2 is selectd

enumerator kECSPI_Channel3
Channel 3 is selectd

enum __ecspi_master_slave_mode
ECSPI master or slave mode configuration.

Values:

enumerator kECSPI_Slave
ECSPI peripheral operates in slave mode.

enumerator kECSPI_Master
ECSPI peripheral operates in master mode.

enum __ecspi_data_line_inactive_state_t
ECSPI data line inactive state configuration.

Values:

enumerator kECSPI_DataLineInactiveStateHigh
The data line inactive state stays high.

enumerator kECSPI_DataLineInactiveStateLow
The data line inactive state stays low.

enum __ecspi_clock_inactive_state_t
ECSPI clock inactive state configuration.

Values:

enumerator kECSPI_ClockInactiveStateLow
The SCLK inactive state stays low.

enumerator kECSPI_ClockInactiveStateHigh
The SCLK inactive state stays high.

enum __ecspi_chip_select_active_state_t
ECSPI active state configuration.

Values:

enumerator kECSPI_ChipSelectActiveStateLow
The SS signal line active stays low.

enumerator kECSPI_ChipSelectActiveStateHigh
The SS signal line active stays high.

enum __ecspi_sample_period_clock_source
ECSPI sample period clock configuration.

Values:

enumerator kECSPI_spiClock
The sample period clock source is SCLK.

enumerator kECSPI_lowFreqClock
The sample seriod clock source is low_frequency reference clock(32.768 kHz).

typedef enum *_ecspi_clock_polarity* ecspi_clock_polarity_t
ECSPI clock polarity configuration.

typedef enum *_ecspi_clock_phase* ecspi_clock_phase_t
ECSPI clock phase configuration.

typedef enum *_ecspi_data_ready* ecspi_Data_ready_t
ECSPI SPI_RDY signal configuration.

typedef enum *_ecspi_channel_source* ecspi_channel_source_t
ECSPI channel select source.

typedef enum *_ecspi_master_slave_mode* ecspi_master_slave_mode_t
ECSPI master or slave mode configuration.

typedef enum *_ecspi_data_line_inactive_state_t* ecspi_data_line_inactive_state_t
ECSPI data line inactive state configuration.

typedef enum *_ecspi_clock_inactive_state_t* ecspi_clock_inactive_state_t
ECSPI clock inactive state configuration.

typedef enum *_ecspi_chip_select_active_state_t* ecspi_chip_select_active_state_t
ECSPI active state configuration.

typedef enum *_ecspi_sample_period_clock_source* ecspi_sample_period_clock_source_t
ECSPI sample period clock configuration.

typedef struct *_ecspi_channel_config* ecspi_channel_config_t
ECSPI user channel configure structure.

typedef struct *_ecspi_master_config* ecspi_master_config_t
ECSPI master configure structure.

typedef struct *_ecspi_slave_config* ecspi_slave_config_t
ECSPI slave configure structure.

typedef struct *_ecspi_transfer* ecspi_transfer_t
ECSPI transfer structure.

typedef struct *_ecspi_master_handle* ecspi_master_handle_t

typedef *ecspi_master_handle_t* ecspi_slave_handle_t
Slave handle is the same with master handle

typedef void (*ecspi_master_callback_t)(ECSPI_Type *base, *ecspi_master_handle_t* *handle, *status_t* status, void *userData)
ECSPI master callback for finished transmit.

typedef void (*ecspi_slave_callback_t)(ECSPI_Type *base, *ecspi_slave_handle_t* *handle, *status_t* status, void *userData)
ECSPI slave callback for finished transmit.

uint32_t ECSPI_GetInstance(ECSPI_Type *base)
Get the instance for ECSPI module.

Parameters

- base – ECSPI base address

ECSPI_DUMMYDATA
ECSPI dummy transfer data, the data is sent while txBuff is NULL.

SPI_RETRY_TIMES
Retry times for waiting flag.

struct _ecspi_channel_config
#include <fsl_ecspi.h> ECSPI user channel configure structure.

Public Members

ecspi_master_slave_mode_t channelMode
Channel mode

ecspi_clock_inactive_state_t clockInactiveState
Clock line (SCLK) inactive state

ecspi_data_line_inactive_state_t dataLineInactiveState
Data line (MOSI&MISO) inactive state

ecspi_chip_select_active_state_t chipSlectActiveState
Chip select(SS) line active state

ecspi_clock_polarity_t polarity
Clock polarity

ecspi_clock_phase_t phase
Clock phase

struct __ecspi__master__config
#include <fsl_ecspi.h> ECSPI master configure structure.

Public Members

ecspi_channel_source_t channel
Channel number

ecspi_channel_config_t channelConfig
Channel configuration

ecspi_sample_period_clock_source_t samplePeriodClock
Sample period clock source

uint16_t burstLength
Burst length. The length shall be less than 4096 bits

uint8_t chipSelectDelay
SS delay time

uint16_t samplePeriod
Sample period

uint8_t txFifoThreshold
TX Threshold

uint8_t rxFifoThreshold
RX Threshold

uint32_t baudRate_Bps
ECSPI baud rate for master mode

bool enableLoopback
Enable the ECSPI loopback test.

struct __ecspi__slave__config
#include <fsl_ecspi.h> ECSPI slave configure structure.

Public Members

uint16_t burstLength
: Burst length. The length shall be less than 4096 bits

uint8_t txFifoThreshold
: TX Threshold

uint8_t rxFifoThreshold
: RX Threshold

ecspi_channel_config_t channelConfig
: Channel configuration

struct __ecspi__transfer
: *#include <fsl_ecspi.h>* ECSPI transfer structure.

Public Members

const uint32_t *txData
: Send buffer

uint32_t *rxData
: Receive buffer

size_t dataSize
: Transfer bytes

ecspi_channel_source_t channel
: ECSPI channel select

struct __ecspi__master__handle
: *#include <fsl_ecspi.h>* ECSPI master handle structure.

Public Members

ecspi_channel_source_t channel
: Channel number

const uint32_t *volatile txData
: Transfer buffer

uint32_t *volatile rxData
: Receive buffer

volatile size_t txRemainingBytes
: Send data remaining in bytes

volatile size_t rxRemainingBytes
: Receive data remaining in bytes

volatile uint32_t state
: ECSPI internal state

size_t transferSize
: Bytes to be transferred

ecspi_master_callback_t callback
: ECSPI callback

void *userData
: Callback parameter

2.8 ECSPI SDMA Driver

void ECSPI_MasterTransferCreateHandleSDMA(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle, *ecspi_sdma_callback_t* callback, void *userData, *sdma_handle_t* *txHandle, *sdma_handle_t* *rxHandle, uint32_t eventSourceTx, uint32_t eventSourceRx, uint32_t TxChannel, uint32_t RxChannel)

Initialize the ECSPI master SDMA handle.

This function initializes the ECSPI master SDMA handle which can be used for other SPI master transactional APIs. Usually, for a specified ECSPI instance, user need only call this API once to get the initialized handle.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users.
- rxHandle – SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users.
- eventSourceTx – event source for ECSPI send, which can be found in SDMA mapping.
- eventSourceRx – event source for ECSPI receive, which can be found in SDMA mapping.
- TxChannel – SDMA channel for ECSPI send.
- RxChannel – SDMA channel for ECSPI receive.

void ECSPI_SlaveTransferCreateHandleSDMA(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle, *ecspi_sdma_callback_t* callback, void *userData, *sdma_handle_t* *txHandle, *sdma_handle_t* *rxHandle, uint32_t eventSourceTx, uint32_t eventSourceRx, uint32_t TxChannel, uint32_t RxChannel)

Initialize the ECSPI Slave SDMA handle.

This function initializes the ECSPI Slave SDMA handle which can be used for other SPI Slave transactional APIs. Usually, for a specified ECSPI instance, user need only call this API once to get the initialized handle.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users.
- rxHandle – SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users.

- eventSourceTx – event source for ECSPI send, which can be found in SDMA mapping.
- eventSourceRx – event source for ECSPI receive, which can be found in SDMA mapping.
- TxChannel – SDMA channel for ECSPI send.
- RxChannel – SDMA channel for ECSPI receive.

status_t ECSPI_MasterTransferSDMA(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle, *ecspi_transfer_t* *xfer)

Perform a non-blocking ECSPI master transfer using SDMA.

Note: This interface returned immediately after transfer initiates.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.
- xfer – Pointer to sdma transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_ECSPI_Busy – EECSPI is not idle, is running another transfer.

status_t ECSPI_SlaveTransferSDMA(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle, *ecspi_transfer_t* *xfer)

Perform a non-blocking ECSPI slave transfer using SDMA.

Note: This interface returned immediately after transfer initiates.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.
- xfer – Pointer to sdma transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_ECSPI_Busy – EECSPI is not idle, is running another transfer.

void ECSPI_MasterTransferAbortSDMA(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle)

Abort a ECSPI master transfer using SDMA.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.

void ECSPI_SlaveTransferAbortSDMA(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle)

Abort a ECSPI slave transfer using SDMA.

Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.

FSL_ECSPI_FREERTOS_DRIVER_VERSION

ECSPI FreeRTOS driver version.

typedef struct *_ecspi_sdma_handle* ecspi_sdma_handle_t

typedef void (*ecspi_sdma_callback_t)(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle, *status_t* status, void *userData)

ECSPI SDMA callback called at the end of transfer.

struct _ecspi_sdma_handle

#include <fsl_ecspi_sdma.h> ECSPI SDMA transfer handle, users should not touch the content of the handle.

Public Members

bool txInProgress

Send transfer finished

bool rxInProgress

Receive transfer finished

sdma_handle_t *txSdmaHandle

DMA handler for ECSPI send

sdma_handle_t *rxSdmaHandle

DMA handler for ECSPI receive

ecspi_sdma_callback_t callback

Callback for ECSPI SDMA transfer

void *userData

User Data for ECSPI SDMA callback

uint32_t state

Internal state of ECSPI SDMA transfer

uint32_t ChannelTx

Channel for send handle

uint32_t ChannelRx

Channel for receive handler

2.9 ENET: Ethernet MAC Driver

void ENET_GetDefaultConfig(*enet_config_t* *config)

Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET MAC controller configure structure for ENET_Init(). User may use the initialized structure unchanged in ENET_Init(), or modify some fields of the structure before calling ENET_Init(). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

- config – The ENET mac controller configuration structure pointer.

status_t ENET_Up(ENET_Type *base, *enet_handle_t* *handle, const *enet_config_t* *config, const *enet_buffer_config_t* *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)

Initializes the ENET module.

This function initializes the module with the ENET configuration.

Note: ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET_Up().

Parameters

- base – ENET peripheral base address.
- handle – ENET handler pointer.
- config – ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- bufferConfig – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- srcClock_Hz – The internal module clock source for MII clock.

Return values

- kStatus_Success – Succeed to initialize the ethernet driver.
- kStatus_ENET_InitMemoryFail – Init fails since buffer memory is not enough.

status_t ENET_Init(ENET_Type *base, *enet_handle_t* *handle, const *enet_config_t* *config, const *enet_buffer_config_t* *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET configuration.

Note: ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET_Init().

Parameters

- base – ENET peripheral base address.
- handle – ENET handler pointer.
- config – ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- bufferConfig – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- srcClock_Hz – The internal module clock source for MII clock.

Return values

- kStatus_Success – Succeed to initialize the ethernet driver.
- kStatus_ENET_InitMemoryFail – Init fails since buffer memory is not enough.

void ENET_Down(ENET_Type *base)

Stops the ENET module.

This function disables the ENET module.

Parameters

- base – ENET peripheral base address.

void ENET_Deinit(ENET_Type *base)

Deinitializes the ENET module.

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

- base – ENET peripheral base address.

static inline void ENET_Reset(ENET_Type *base)

Resets the ENET module.

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Parameters

- base – ENET peripheral base address.

void ENET_SetMII(ENET_Type *base, *enet_mii_speed_t* speed, *enet_mii_duplex_t* duplex)

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and dulpex for MAC.

Parameters

- base – ENET peripheral base address.
- speed – The speed of the RMII mode.

- duplex – The duplex of the RMII mode.

void ENET_SetSMI(ENET_Type *base, uint32_t srcClock_Hz, bool isPreambleDisabled)

Sets the ENET SMI(serial management interface)- MII management interface.

Parameters

- base – ENET peripheral base address.
- srcClock_Hz – This is the ENET module clock frequency. See clock distribution.
- isPreambleDisabled – The preamble disable flag.
  - true Enables the preamble.
  - false Disables the preamble.

static inline bool ENET_GetSMI(ENET_Type *base)

Gets the ENET SMI- MII management interface configuration.

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

- base – ENET peripheral base address.

Returns

The SMI setup status true or false.

static inline uint32_t ENET_ReadSMIData(ENET_Type *base)

Reads data from the PHY register through an SMI interface.

Parameters

- base – ENET peripheral base address.

Returns

The data read from PHY

static inline void ENET_StartSMIWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, *enet_mii_write_t* operation, uint16_t data)

Sends the MDIO IEEE802.3 Clause 22 format write command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIOWrite() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register address. Range from 0 ~ 31.
- operation – The write operation.
- data – The data written to PHY.

static inline void ENET_StartSMIRead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, *enet_mii_read_t* operation)

Sends the MDIO IEEE802.3 Clause 22 format read command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIORead() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register address. Range from 0 ~ 31.
- operation – The read operation.

status_t ENET_MDIOWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- data – The data written to PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

status_t ENET_MDIORead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- pData – The data read from PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

static inline void ENET_StartExtC45SMIWriteReg(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr)

Sends the MDIO IEEE802.3 Clause 45 format write register command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIOC45Write()/ENET_MDIOC45Read() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.

static inline void ENET_StartExtC45SMIWriteData(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t data)

Sends the MDIO IEEE802.3 Clause 45 format write data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIOC45Write() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- data – The data written to PHY.

static inline void ENET_StartExtC45SMIReadData(ENET_Type *base, uint8_t portAddr, uint8_t devAddr)

Sends the MDIO IEEE802.3 Clause 45 format read data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIOC45Read() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.

status_t ENET_MDIOC45Write(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns
kStatus_Success MDIO access succeeds.

Returns
kStatus_Timeout MDIO access timeout.

status_t ENET_MDIOC45Read(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The data read from PHY.

Returns
kStatus_Success MDIO access succeeds.

Returns
kStatus_Timeout MDIO access timeout.

static inline void ENET_SetRGMIIClockDelay(ENET_Type *base, bool txEnabled, bool rxEnabled)

Control the usage of the delayed tx/rx RGMII clock.

Parameters

- base – ENET peripheral base address.
- txEnabled – Enable or disable to generate the delayed version of RGMII_TXC.
- rxEnabled – Enable or disable to use the delayed version of RGMII_RXC.

void ENET_SetMacAddr(ENET_Type *base, uint8_t *macAddr)

Sets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET_GetMacAddr(ENET_Type *base, uint8_t *macAddr)

Gets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET_AddMulticastGroup(ENET_Type *base, uint8_t *address)

Adds the ENET device to a multicast group.

Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

void ENET_LeaveMulticastGroup(ENET_Type *base, uint8_t *address)

Moves the ENET device from a multicast group.

Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

static inline void ENET_ActiveRead(ENET_Type *base)

Activates frame reception for multiple rings.

This function is to active the enet read process.

Note: This must be called after the MAC configuration and state are ready. It must be called after the ENET_Init(). This should be called when the frame reception is required.

Parameters

- base – ENET peripheral base address.

static inline void ENET_EnableSleepMode(ENET_Type *base, bool enable)

Enables/disables the MAC to enter sleep mode. This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

Parameters

- base – ENET peripheral base address.
- enable – True enable sleep mode, false disable sleep mode.

static inline void ENET_GetAccelFunction(ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)

Gets ENET transmit and receive accelerator functions from MAC controller.

Parameters

- base – ENET peripheral base address.
- txAccelOption – The transmit accelerator option. The "enet_tx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.
- rxAccelOption – The receive accelerator option. The "enet_rx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.

static inline void ENET_EnableInterrupts(ENET_Type *base, uint32_t mask)

Enables the ENET interrupt.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See enet_interrupt_enable_t. For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of the enumeration enet_interrupt_enable_t.

static inline void ENET_DisableInterrupts(ENET_Type *base, uint32_t mask)

Disables the ENET interrupt.

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See enet_interrupt_enable_t. For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to disable. This is a logical OR of the enumeration enet_interrupt_enable_t.

static inline uint32_t ENET_GetInterruptStatus(ENET_Type *base)

Gets the ENET interrupt status flag.

Parameters

- base – ENET peripheral base address.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration enet_interrupt_enable_t.

static inline void ENET_ClearInterruptStatus(ENET_Type *base, uint32_t mask)

Clears the ENET interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the enet_interrupt_enable_t. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_ClearInterruptStatus(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration enet_interrupt_enable_t.

void ENET_SetRxISRHandler(ENET_Type *base, *enet_isr_t* ISRHandler)

Set the second level Rx IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

void ENET_SetTxISRHandler(ENET_Type *base, *enet_isr_t* ISRHandler)

Set the second level Tx IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

void ENET_SetErrISRHandler(ENET_Type *base, *enet_isr_t* ISRHandler)

Set the second level Err IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

void ENET_GetRxErrBeforeReadFrame(*enet_handle_t* *handle, *enet_data_error_stats_t* *eErrorStatic, uint8_t ringId)

Gets the error statistics of a received frame for ENET specified ring.

This API must be called after the ENET_GetRxFrameSize and before the ENET_ReadFrame(). If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (status == kStatus_ENET_RxFrameError)
{
    Comments: Get the error information of the received frame.
```

(continues on next page)

(continued from previous page)

```
    ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
    Comments: update the receive buffer.
    ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
}
```

Parameters

- handle – The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init.
- eErrorStatic – The error statistics structure pointer.
- ringId – The ring index, range from 0 ~ (FSL_FEATURE_ENET_INSTANCE_QUEUEn(x) - 1).

void ENET_EnableStatistics(ENET_Type *base, bool enable)

Enables/disables collection of transfer statistics.

Note that this function does not reset any of the already collected data, use the function ENET_ResetStatistics to clear the transfer statistics if needed.

Parameters

- base – ENET peripheral base address.
- enable – True enable statistics collection, false disable statistics collection.

void ENET_GetStatistics(ENET_Type *base, *enet_transfer_stats_t* *statistics)

Gets transfer statistics.

Copies the actual value of hardware counters into the provided structure. Calling this function does not reset the counters in hardware.

Parameters

- base – ENET peripheral base address.
- statistics – The statistics structure pointer.

void ENET_ResetStatistics(ENET_Type *base)

Resets transfer statistics.

Sets the value of hardware transfer counters to zero.

Parameters

- base – ENET peripheral base address.

status_t ENET_GetRxFrameSize(*enet_handle_t* *handle, uint32_t *length, uint8_t ringId)

Gets the size of the read frame for specified ring.

This function gets a received frame size from the ENET buffer descriptors.

Note: The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, ENET_ReadFrame() should be called to receive frame and update the BD if the result is not “kStatus_ENET_RxFrameEmpty”.

Parameters

- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- length – The length of the valid frame received.
- ringId – The ring index or ring number.

Return values

- kStatus_ENET_RxFrameEmpty – No frame received. Should not call ENET_ReadFrame to read frame.
- kStatus_ENET_RxFrameError – Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.
- kStatus_Success – Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input.

status_t ENET_ReadFrame(ENET_Type *base, *enet_handle_t* *handle, uint8_t *data, uint32_t length, uint8_t ringId, uint32_t *ts)

Reads a frame from the ENET device. This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL.

Note: It doesn't store the timestamp in the receive timestamp queue. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption. This is an example:

```
uint32_t length;
enet_handle_t g_handle;
Comments: Get the received frame size firstly.
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    Comments: Allocate memory here with the size of "length"
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
        Comments: Add the console warning log.
    }
    else
    {
        status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
        Comments: Call stack input API to deliver the data to stack
    }
}
else if (status == kStatus_ENET_RxFrameError)
{
    Comments: Update the received buffer when a error frame is received.
    ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
}
```

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- data – The data buffer provided by user to store the frame which memory size should be at least "length".
- length – The size of the data buffer which is still the length of the received frame.

- ringId – The ring index or ring number.
- ts – The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

status_t ENET_SendFrame(ENET_Type *base, *enet_handle_t* *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)

Transmits an ENET frame for specified ring.

Note: The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- data – The data buffer provided by user to send.
- length – The length of the data to send.
- ringId – The ring index or ring number.
- tsFlag – Timestamp enable flag.
- context – Used by user to handle some events after transmit over.

Return values

- kStatus_Success – Send frame succeed.
- kStatus_ENET_TxFrameBusy – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus_ENET_TxFrameBusy.

status_t ENET_SetTxReclaim(*enet_handle_t* *handle, bool isEnable, uint8_t ringId)

Enable or disable tx descriptors reclaim mechanism.

Note: This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

Parameters

- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- isEnable – Enable or disable flag.
- ringId – The ring index or ring number.

Return values

- kStatus_Success – Succeed to enable/disable Tx reclaim.
- kStatus_Fail – Fail to enable/disable Tx reclaim.

void ENET_ReclaimTxDescriptor(ENET_Type *base, *enet_handle_t* *handle, uint8_t ringId)

Reclaim tx descriptors. This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interupt IRQ handler after the complete of a frame transmission.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- ringId – The ring index or ring number.

status_t ENET_GetRxFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_rx_frame_struct_t* *rxFrame, uint8_t ringId)

Receives one frame in specified BD ring with zero copy.

This function uses the user-defined allocation and free callbacks. Every time application gets one frame through this function, driver stores the buffer address(es) in enet_buffer_struct_t and allocate new buffer(s) for the BD(s). If there's no memory buffer in the pool, this function drops current one frame to keep the Rx frame in BD ring is as fresh as possible.

Note: Application must provide a memory pool including at least BD number + n buffers in order for this function to work properly, because each BD must always take one buffer while driver is running, then other extra n buffer(s) can be taken by application. Here n is the ceil(max_frame_length(set by RCR) / bd_rx_size(set by MRBR)). Application must also provide an array structure in rxFrame->rxBuffArray with n index to receive one complete frame in any case.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- rxFrame – The received frame information structure provided by user.
- ringId – The ring index or ring number.

Return values

- kStatus_Success – Succeed to get one frame and allocate new memory for Rx buffer.
- kStatus_ENET_RxFrameEmpty – There's no Rx frame in the BD.
- kStatus_ENET_RxFrameError – There's issue in this receiving.
- kStatus_ENET_RxFrameDrop – There's no new buffer memory for BD, drop this frame.

status_t ENET_StartTxFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_tx_frame_struct_t* *txFrame, uint8_t ringId)

Sends one frame in specified BD ring with zero copy.

This function supports scattered buffer transmit, user needs to provide the buffer array.

Note: Tx reclaim should be enabled to ensure the Tx buffer ownership can be given back to application after Tx is over.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- txFrame – The Tx frame structure.
- ringId – The ring index or ring number.

Return values

- kStatus_Success – Succeed to send one frame.
- kStatus_ENET_TxFrameBusy – The BD is not ready for Tx or the reclaim operation still not finishs.
- kStatus_ENET_TxFrameOverLen – The Tx frame length is over max ethernet frame length.

void ENET_TransmitIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

The transmit IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET_ReceiveIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

The receive IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET_ErrorIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

Some special IRQ handler including the error, mii, wakeup irq handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET_Ptp1588IRQHandler(ENET_Type *base)

the common IRQ handler for the 1588 irq handler.

This is used for the 1588 timer interrupt.

Parameters

- base – ENET peripheral base address.

void ENET_CommonFrame0IRQHandler(ENET_Type *base)

the common IRQ handler for the tx/rx/error etc irq handler.

This is used for the combined tx/rx/error interrupt for single/mutli-ring (frame 0).

Parameters

- base – ENET peripheral base address.

FSL_ENET_DRIVER_VERSION

Defines the driver version.

ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK

Empty bit mask.

ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK
: Software owner one mask.

ENET_BUFFDESCRIPTOR_RX_WRAP_MASK
: Next buffer descriptor is the start address.

ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask
: Software owner two mask.

ENET_BUFFDESCRIPTOR_RX_LAST_MASK
: Last BD of the frame mask.

ENET_BUFFDESCRIPTOR_RX_MISS_MASK
: Received because of the promiscuous mode.

ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK
: Broadcast packet mask.

ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK
: Multicast packet mask.

ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK
: Length violation mask.

ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK
: Non-octet aligned frame mask.

ENET_BUFFDESCRIPTOR_RX_CRC_MASK
: CRC error mask.

ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK
: FIFO overrun mask.

ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK
: Frame is truncated mask.

ENET_BUFFDESCRIPTOR_TX_READY_MASK
: Ready bit mask.

ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK
: Software owner one mask.

ENET_BUFFDESCRIPTOR_TX_WRAP_MASK
: Wrap buffer descriptor mask.

ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK
: Software owner two mask.

ENET_BUFFDESCRIPTOR_TX_LAST_MASK
: Last BD of the frame mask.

ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK
: Transmit CRC mask.

ENET_FRAME_MAX_FRAMELEN
: Default maximum Ethernet frame size without VLAN tag.

ENET_FRAME_VLAN_TAGLEN
: Ethernet single VLAN tag size.

ENET_FRAME_CRC_LEN
: CRC size in a frame.

ENET_FRAME_TX_LEN_LIMITATION(x)

ENET_FIFO_MIN_RX_FULL
ENET minimum receive FIFO full.

ENET_RX_MIN_BUFFERSIZE
ENET minimum buffer size.

ENET_PHY_MAXADDRESS
Maximum PHY address.

ENET_TX_INTERRUPT
Enet Tx interrupt flag.

ENET_RX_INTERRUPT
Enet Rx interrupt flag.

ENET_TS_INTERRUPT
Enet timestamp interrupt flag.

ENET_ERR_INTERRUPT
Enet error interrupt flag.

Defines the status return codes for transaction.

Values:

enumerator kStatus_ENET_InitMemoryFail
Init fails since buffer memory is not enough.

enumerator kStatus_ENET_RxFrameError
A frame received but data error happen.

enumerator kStatus_ENET_RxFrameFail
Failed to receive a frame.

enumerator kStatus_ENET_RxFrameEmpty
No frame arrive.

enumerator kStatus_ENET_RxFrameDrop
Rx frame is dropped since no buffer memory.

enumerator kStatus_ENET_TxFrameOverLen
Tx frame over length.

enumerator kStatus_ENET_TxFrameBusy
Tx buffer descriptors are under process.

enumerator kStatus_ENET_TxFrameFail
Transmit frame fail.

enum _enet_mii_mode
Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

Values:

enumerator kENET_MiiMode
MII mode for data interface.

enumerator kENET_RmiiMode
RMII mode for data interface.

enumerator kENET_RgmiiMode
RGMII mode for data interface.

enum _enet_mii_speed
Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

Values:

enumerator kENET_MiiSpeed10M
Speed 10 Mbps.

enumerator kENET_MiiSpeed100M
Speed 100 Mbps.

enumerator kENET_MiiSpeed1000M
Speed 1000M bps.

enum _enet_mii_duplex
Defines the half or full duplex for the MII data interface.

Values:

enumerator kENET_MiiHalfDuplex
Half duplex mode.

enumerator kENET_MiiFullDuplex
Full duplex mode.

enum _enet_mii_write
Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

Values:

enumerator kENET_MiiWriteNoCompliant
Write frame operation, but not MII-compliant.

enumerator kENET_MiiWriteValidFrame
Write frame operation for a valid MII management frame.

enum _enet_mii_read
Defines the read operation for the MII management frame.

Values:

enumerator kENET_MiiReadValidFrame
Read frame operation for a valid MII management frame.

enumerator kENET_MiiReadNoCompliant
Read frame operation, but not MII-compliant.

enum _enet_mii_extend_opcode
Define the MII opcode for extended MDIO_CLAUSES_45 Frame.

Values:

enumerator kENET_MiiAddrWrite_C45
Address Write operation.

enumerator kENET_MiiWriteFrame_C45
Write frame operation for a valid MII management frame.

enumerator kENET_MiiReadFrame_C45
Read frame operation for a valid MII management frame.

enum _enet_special_control_flag

Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the enet_config_t.

Values:

enumerator kENET_ControlFlowControlEnable

Enable ENET flow control: pause frame.

enumerator kENET_ControlRxPayloadCheckEnable

Enable ENET receive payload length check.

enumerator kENET_ControlRxPadRemoveEnable

Padding is removed from received frames.

enumerator kENET_ControlRxBroadCastRejectEnable

Enable broadcast frame reject.

enumerator kENET_ControlMacAddrInsert

Enable MAC address insert.

enumerator kENET_ControlStoreAndFwdDisable

Enable FIFO store and forward.

enumerator kENET_ControlSMIPreambleDisable

Enable SMI preamble.

enumerator kENET_ControlPromiscuousEnable

Enable promiscuous mode.

enumerator kENET_ControlMIILoopEnable

Enable ENET MII loop back.

enumerator kENET_ControlVLANTagEnable

Enable normal VLAN (single vlan tag).

enumerator kENET_ControlSVLANEnable

Enable S-VLAN.

enumerator kENET_ControlVLANUseSecondTag

Enable extracting the second vlan tag for further processing.

enum _enet_interrupt_enable

List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Values:

enumerator kENET_BabrInterrupt

Babbling receive error interrupt source

enumerator kENET_BabtInterrupt

Babbling transmit error interrupt source

enumerator kENET_GraceStopInterrupt

Graceful stop complete interrupt source

enumerator kENET_TxFrameInterrupt
TX FRAME interrupt source

enumerator kENET_TxBufferInterrupt
TX BUFFER interrupt source

enumerator kENET_RxFrameInterrupt
RX FRAME interrupt source

enumerator kENET_RxBufferInterrupt
RX BUFFER interrupt source

enumerator kENET_MiiInterrupt
MII interrupt source

enumerator kENET_EBusERInterrupt
Ethernet bus error interrupt source

enumerator kENET_LateCollisionInterrupt
Late collision interrupt source

enumerator kENET_RetryLimitInterrupt
Collision Retry Limit interrupt source

enumerator kENET_UnderrunInterrupt
Transmit FIFO underrun interrupt source

enumerator kENET_PayloadRxInterrupt
Payload Receive error interrupt source

enumerator kENET_WakeupInterrupt
WAKEUP interrupt source

enumerator kENET_TsAvailInterrupt
TS AVAIL interrupt source for PTP

enumerator kENET_TsTimerInterrupt
TS WRAP interrupt source for PTP

enum _enet_event
Defines the common interrupt event for callback use.

Values:

enumerator kENET_RxEvent
Receive event.

enumerator kENET_TxEvent
Transmit event.

enumerator kENET_ErrEvent
Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

enumerator kENET_WakeUpEvent
Wake up from sleep mode event.

enumerator kENET_TimeStampEvent
Time stamp event.

enumerator kENET_TimeStampAvailEvent
Time stamp available event.

enum _enet_idle_slope

Defines certain idle slope for bandwidth fraction.

Values:

enumerator kENET_IdleSlope1

The bandwidth fraction is about 0.002.

enumerator kENET_IdleSlope2

The bandwidth fraction is about 0.003.

enumerator kENET_IdleSlope4

The bandwidth fraction is about 0.008.

enumerator kENET_IdleSlope8

The bandwidth fraction is about 0.02.

enumerator kENET_IdleSlope16

The bandwidth fraction is about 0.03.

enumerator kENET_IdleSlope32

The bandwidth fraction is about 0.06.

enumerator kENET_IdleSlope64

The bandwidth fraction is about 0.11.

enumerator kENET_IdleSlope128

The bandwidth fraction is about 0.20.

enumerator kENET_IdleSlope256

The bandwidth fraction is about 0.33.

enumerator kENET_IdleSlope384

The bandwidth fraction is about 0.43.

enumerator kENET_IdleSlope512

The bandwidth fraction is about 0.50.

enumerator kENET_IdleSlope640

The bandwidth fraction is about 0.56.

enumerator kENET_IdleSlope768

The bandwidth fraction is about 0.60.

enumerator kENET_IdleSlope896

The bandwidth fraction is about 0.64.

enumerator kENET_IdleSlope1024

The bandwidth fraction is about 0.67.

enumerator kENET_IdleSlope1152

The bandwidth fraction is about 0.69.

enumerator kENET_IdleSlope1280

The bandwidth fraction is about 0.71.

enumerator kENET_IdleSlope1408

The bandwidth fraction is about 0.73.

enumerator kENET_IdleSlope1536

The bandwidth fraction is about 0.75.

enum _enet_tx_accelerator

Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

Values:

enumerator kENET_TxAccelIsShift16Enabled

Transmit FIFO shift-16.

enumerator kENET_TxAccelIpCheckEnabled

Insert IP header checksum.

enumerator kENET_TxAccelProtoCheckEnabled

Insert protocol checksum (TCP, UDP, ICMPv4).

enum _enet_rx_accelerator

Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

Values:

enumerator kENET_RxAccelPadRemoveEnabled

Padding removal for short IP frames.

enumerator kENET_RxAccelIpCheckEnabled

Discard with wrong IP header checksum.

enumerator kENET_RxAccelProtoCheckEnabled

Discard with wrong protocol checksum (TCP, UDP, ICMPv4).

enumerator kENET_RxAccelMacCheckEnabled

Discard with Mac layer errors.

enumerator kENET_RxAccelisShift16Enabled

Receive FIFO shift-16.

typedef enum *_enet_mii_mode* enet_mii_mode_t

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

typedef enum *_enet_mii_speed* enet_mii_speed_t

Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: “kENET_MiiSpeed1000M” only supported when mii mode is “kENET_RgmiiMode”.

typedef enum *_enet_mii_duplex* enet_mii_duplex_t

Defines the half or full duplex for the MII data interface.

typedef enum *_enet_mii_write* enet_mii_write_t

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

typedef enum *_enet_mii_read* enet_mii_read_t

Defines the read operation for the MII management frame.

typedef enum *_enet_mii_extend_opcode* enet_mii_extend_opcode

Define the MII opcode for extended MDIO_CLAUSES_45 Frame.

typedef enum *_enet_special_control_flag* enet_special_control_flag_t

Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the enet_config_t.

typedef enum *_enet_interrupt_enable* enet_interrupt_enable_t

List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

typedef enum *_enet_event* enet_event_t

Defines the common interrupt event for callback use.

typedef enum *_enet_idle_slope* enet_idle_slope_t

Defines certain idle slope for bandwidth fraction.

typedef enum *_enet_tx_accelerator* enet_tx_accelerator_t

Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef enum *_enet_rx_accelerator* enet_rx_accelerator_t

Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef struct *_enet_rx_bd_struct* enet_rx_bd_struct_t

Defines the receive buffer descriptor structure for the little endian system.

typedef struct *_enet_tx_bd_struct* enet_tx_bd_struct_t

Defines the enhanced transmit buffer descriptor structure for the little endian system.

typedef struct *_enet_data_error_stats* enet_data_error_stats_t

Defines the ENET data error statistics structure.

typedef struct *_enet_rx_frame_error* enet_rx_frame_error_t

Defines the Rx frame error structure.

typedef struct *_enet_transfer_stats* enet_transfer_stats_t

Defines the ENET transfer statistics structure.

typedef struct *enet_frame_info* enet_frame_info_t

Defines the frame info structure.

typedef struct *_enet_tx_dirty_ring* enet_tx_dirty_ring_t

Defines the ENET transmit dirty addresses ring/queue structure.

typedef void *(*enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)

Defines the ENET Rx memory buffer alloc function pointer.

typedef void (*enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)

Defines the ENET Rx memory buffer free function pointer.

typedef struct *_enet_buffer_config* enet_buffer_config_t

Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

a. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

b. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.

c. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber * rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber * txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

typedef struct *_enet_intcoalesce_config* enet_intcoalesce_config_t

Defines the interrupt coalescing configure structure.

typedef struct *_enet_avb_config* enet_avb_config_t

Defines the ENET AVB Configure structure.

This is used for to configure the extended ring 1 and ring 2.

a. The classification match format is (CMP3 « 12) | (CMP2 « 8) | (CMP1 « 4) | CMP0. composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

a. The idleSlope is used to calculate the Band Width fraction, BW fraction = 1 / (1 + 512/idleSlope). For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

typedef struct *_enet_handle* enet_handle_t

typedef void (*enet_callback_t)(ENET_Type *base, *enet_handle_t* *handle, *enet_event_t* event, *enet_frame_info_t* *frameInfo, void *userData)

ENET callback function.

typedef struct *_enet_config* enet_config_t

Defines the basic configuration structure for the ENET device.

Note:

a. macSpecialConfig is used for a special control configuration, A logical OR of "enet_special_control_flag_t". For a special configuration for MAC, set this parameter to 0.

b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.

c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is

0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.

d. When “kENET_ControlFlowControlEnable” is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.

e. When “kENET_ControlStoreAndFwdDisabled” is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.

f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The “enet_tx_accelerator_t” and “enet_rx_accelerator_t” are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.

g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrese the CPU loading.

typedef struct *_enet_tx_bd_ring* enet_tx_bd_ring_t

Defines the ENET transmit buffer descriptor ring/queue structure.

typedef struct *_enet_rx_bd_ring* enet_rx_bd_ring_t

Defines the ENET receive buffer descriptor ring/queue structure.

typedef struct *_enet_buffer_struct* enet_buffer_struct_t

typedef struct *_enet_rx_frame_attribute_struct* enet_rx_frame_attribute_t

typedef struct *_enet_rx_frame_struct* enet_rx_frame_struct_t

typedef struct *_enet_tx_frame_struct* enet_tx_frame_struct_t

typedef void (*enet_isr_t)(ENET_Type *base, *enet_handle_t* *handle)

Define interrupt IRQ handler.

const *clock_ip_name_t* s_enetClock[]

Pointers to enet clocks for each instance.

const *clock_ip_name_t* s_enetExtraClock[]

uint32_t ENET_GetInstance(ENET_Type *base)

Get the ENET instance from peripheral base address.

Parameters

- base – ENET peripheral base address.

Returns

ENET instance.

ENET_BUFFDESCRIPTOR_RX_ERR_MASK

Defines the receive error status flag mask.

struct _enet_rx_bd_struct

#include <fsl_enet.h> Defines the receive buffer descriptor structure for the little endian system.

Public Members

uint16_t length

Buffer descriptor data length.

uint16_t control
: Buffer descriptor control and status.

uint32_t buffer
: Data buffer pointer.

struct __enet_tx_bd_struct
: *#include <fsl_enet.h>* Defines the enhanced transmit buffer descriptor structure for the little endian system.

Public Members

uint16_t length
: Buffer descriptor data length.

uint16_t control
: Buffer descriptor control and status.

uint32_t buffer
: Data buffer pointer.

struct __enet_data_error_stats
: *#include <fsl_enet.h>* Defines the ENET data error statistics structure.

Public Members

uint32_t statsRxLenGreaterErr
: Receive length greater than RCR[MAX_FL].

uint32_t statsRxAlignErr
: Receive non-octet alignment/

uint32_t statsRxFcsErr
: Receive CRC error.

uint32_t statsRxOverRunErr
: Receive over run.

uint32_t statsRxTruncateErr
: Receive truncate.

struct __enet_rx_frame_error
: *#include <fsl_enet.h>* Defines the Rx frame error structure.

Public Members

bool statsRxTruncateErr
: Receive truncate.

bool statsRxOverRunErr
: Receive over run.

bool statsRxFcsErr
: Receive CRC error.

bool statsRxAlignErr
: Receive non-octet alignment.

bool statsRxLenGreaterErr
Receive length greater than RCR[MAX_FL].

struct _enet_transfer_stats
#include <fsl_enet.h> Defines the ENET transfer statistics structure.

Public Members

uint32_t statsRxFrameCount
Rx frame number.

uint32_t statsRxFrameOk
Good Rx frame number.

uint32_t statsRxCrcErr
Rx frame number with CRC error.

uint32_t statsRxAlignErr
Rx frame number with alignment error.

uint32_t statsRxDropInvalidSFD
Dropped frame number due to invalid SFD.

uint32_t statsRxFifoOverflowErr
Rx FIFO overflow count.

uint32_t statsTxFrameCount
Tx frame number.

uint32_t statsTxFrameOk
Good Tx frame number.

uint32_t statsTxCrcAlignErr
The transmit frame is error.

uint32_t statsTxFifoUnderRunErr
Tx FIFO underrun count.

struct enet_frame_info
#include <fsl_enet.h> Defines the frame info structure.

Public Members

void *context
User specified data

struct _enet_tx_dirty_ring
#include <fsl_enet.h> Defines the ENET transmit dirty addresses ring/queue structure.

Public Members

enet_frame_info_t *txDirtyBase
Dirty buffer descriptor base address pointer.

uint16_t txGenIdx
tx generate index.

uint16_t txConsumIdx
tx consume index.

uint16_t txRingLen

tx ring length.

bool isFull

tx ring is full flag.

struct __enet_buffer_config

#include <fsl_enet.h> Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

a. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

b. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.

c. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber * rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber * txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

Public Members

uint16_t rxBdNumber

Receive buffer descriptor number.

uint16_t txBdNumber

Transmit buffer descriptor number.

uint16_t rxBuffSizeAlign

Aligned receive data buffer size.

uint16_t txBuffSizeAlign

Aligned transmit data buffer size.

volatile *enet_rx_bd_struct_t* *rxBdStartAddrAlign

Aligned receive buffer descriptor start address: should be non-cacheable.

volatile *enet_tx_bd_struct_t* *txBdStartAddrAlign

Aligned transmit buffer descriptor start address: should be non-cacheable.

uint8_t *rxBufferAlign

Receive data buffer start address.

uint8_t *txBufferAlign

Transmit data buffer start address.

bool rxMaintainEnable

Receive buffer cache maintain.

bool txMaintainEnable

Transmit buffer cache maintain.

enet_frame_info_t *txFrameInfo
Transmit frame information start address.

struct __enet_intcoalesce_config
#include <fsl_enet.h> Defines the interrupt coalescing configure structure.

Public Members

uint8_t txCoalesceFrameCount[1]
Transmit interrupt coalescing frame count threshold.

uint16_t txCoalesceTimeCount[1]
Transmit interrupt coalescing timer count threshold.

uint8_t rxCoalesceFrameCount[1]
Receive interrupt coalescing frame count threshold.

uint16_t rxCoalesceTimeCount[1]
Receive interrupt coalescing timer count threshold.

struct __enet_avb_config
#include <fsl_enet.h> Defines the ENET AVB Configure structure.

This is used for to configure the extended ring 1 and ring 2.

a. The classification match format is (CMP3 « 12) | (CMP2 « 8) | (CMP1 « 4) | CMP0. composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

a. The idleSlope is used to calculate the Band Width fraction, BW fraction = 1 / (1 + 512/idleSlope). For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

Public Members

uint16_t rxClassifyMatch[1 - 1]
The classification match value for the ring.

enet_idle_slope_t idleSlope[1 - 1]
The idle slope for certian bandwidth fraction.

struct __enet_config
#include <fsl_enet.h> Defines the basic configuration structure for the ENET device.

Note:

a. macSpecialConfig is used for a special control configuration, A logical OR of “enet_special_control_flag_t”. For a special configuration for MAC, set this parameter to 0.

b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.

c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the

txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.

d. When “kENET_ControlFlowControlEnable” is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.

e. When “kENET_ControlStoreAndFwdDisabled” is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.

f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The “enet_tx_accelerator_t” and “enet_rx_accelerator_t” are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.

g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrese the CPU loading.

Public Members

uint32_t macSpecialConfig

Mac special configuration. A logical OR of “enet_special_control_flag_t”.

uint32_t interrupt

Mac interrupt source. A logical OR of “enet_interrupt_enable_t”.

uint16_t rxMaxFrameLen

Receive maximum frame length.

enet_mii_mode_t miiMode

MII mode.

enet_mii_speed_t miiSpeed

MII Speed.

enet_mii_duplex_t miiDuplex

MII duplex.

uint8_t rxAccelerConfig

Receive accelerator, A logical OR of “enet_rx_accelerator_t”.

uint8_t txAccelerConfig

Transmit accelerator, A logical OR of “enet_rx_accelerator_t”.

uint16_t pauseDuration

For flow control enabled case: Pause duration.

uint8_t rxFifoEmptyThreshold

For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.

uint8_t rxFifoStatEmptyThreshold

For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept. If the limit is reached, reception continues and a pause frame is triggered.

uint8_t rxFifoFullThreshold

For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.

uint8_t txFifoWatermark

For store and forward disable case, the data required in TX FIFO before a frame transmit start.

enet_intcoalesce_config_t *intCoalesceCfg

If the interrupt coalsecence is not required in the ring n(0,1,2), please set to NULL.

uint8_t ringNum

Number of used rings. default with 1 — single ring.

enet_rx_alloc_callback_t rxBuffAlloc

Callback function to alloc memory, must be provided for zero-copy Rx.

enet_rx_free_callback_t rxBuffFree

Callback function to free memory, must be provided for zero-copy Rx.

enet_callback_t callback

General callback function.

void *userData

Callback function parameter.

struct __enet_tx_bd_ring

#include <fsl_enet.h> Defines the ENET transmit buffer descriptor ring/queue structure.

Public Members

volatile *enet_tx_bd_struct_t* *txBdBase

Buffer descriptor base address pointer.

uint16_t txGenIdx

The current available transmit buffer descriptor pointer.

uint16_t txConsumIdx

Transmit consume index.

volatile uint16_t txDescUsed

Transmit descriptor used number.

uint16_t txRingLen

Transmit ring length.

struct __enet_rx_bd_ring

#include <fsl_enet.h> Defines the ENET receive buffer descriptor ring/queue structure.

Public Members

volatile *enet_rx_bd_struct_t* *rxBdBase

Buffer descriptor base address pointer.

uint16_t rxGenIdx

The current available receive buffer descriptor pointer.

uint16_t rxRingLen

Receive ring length.

struct __enet_handle

#include <fsl_enet.h> Defines the ENET handler structure.

Public Members

enet_rx_bd_ring_t rxBdRing[1]
Receive buffer descriptor.

enet_tx_bd_ring_t txBdRing[1]
Transmit buffer descriptor.

uint16_t rxBuffSizeAlign[1]
Receive buffer size alignment.

uint16_t txBuffSizeAlign[1]
Transmit buffer size alignment.

bool rxMaintainEnable[1]
Receive buffer cache maintain.

bool txMaintainEnable[1]
Transmit buffer cache maintain.

uint8_t ringNum
Number of used rings.

enet_callback_t callback
Callback function.

void *userData
Callback function parameter.

enet_tx_dirty_ring_t txDirtyRing[1]
Ring to store tx frame information.

bool txReclaimEnable[1]
Tx reclaim enable flag.

enet_rx_alloc_callback_t rxBuffAlloc
Callback function to alloc memory for zero copy Rx.

enet_rx_free_callback_t rxBuffFree
Callback function to free memory for zero copy Rx.

uint8_t multicastCount[64]
Multicast collisions counter

uint32_t enetClock
The clock of enet peripheral, to caculate core cycles for PTP timestamp.

uint32_t tsDelayCount
The count of core cycles for PTP timestamp capture delay.

struct __enet_buffer_struct
#include <fsl_enet.h>

Public Members

void *buffer
The buffer store the whole or partial frame.

uint16_t length
The byte length of this buffer.

struct __enet_rx_frame_attribute_struct
#include <fsl_enet.h>

Public Members

bool promiscuous
This frame is received because of promiscuous mode.

struct __enet_rx_frame_struct
#include <fsl_enet.h>

Public Members

enet_buffer_struct_t *rxBuffArray
Rx frame buffer structure.

uint16_t totLen
Rx frame total length.

enet_rx_frame_attribute_t rxAttribute
Rx frame attribute structure.

enet_rx_frame_error_t rxFrameError
Rx frame error.

struct __enet_tx_frame_struct
#include <fsl_enet.h>

Public Members

enet_buffer_struct_t *txBuffArray
Tx frame buffer structure.

uint32_t txBuffNum
Buffer number of this Tx frame.

void *context
Driver reclaims and gives it in Tx over callback, usually store network packet header.

2.10 GPC: General Power Controller Driver

FSL_GPC_DRIVER_VERSION
GPC driver version 2.2.0.

enum _gpc_lpm_mode
GPC LPM mode definition.

Values:

enumerator kGPC_RunMode
run mode

enumerator kGPC_WaitMode
wait mode

enumerator kGPC_StopMode
stop mode

enum _gpc_pgc_ack_sel

PGC ack signal selection

Values:

enumerator kGPC_DummyPGCPowerUpAck

dummy power up ack signal

enumerator kGPC_VirtualPGCPowerUpAck

virtual pgc power up ack signal

enumerator kGPC_DummyPGCPowerDownAck

dummy power down ack signal

enumerator kGPC_VirtualPGCPowerDownAck

virtual pgc power down ack signal

enumerator kGPC_NocPGCPowerUpAck

NOC power up ack signal

enumerator kGPC_NocPGCPowerDownAck

NOC power

enum _gpc_standby_count

Standby counter which GPC will wait between PMIC_STBY_REQ negation and assertion of PMIC_READY

Values:

enumerator kGPC_StandbyCounter4CkilClk

4 ckil clocks

enumerator kGPC_StandbyCounter8CkilClk

8 ckil clocks

enumerator kGPC_StandbyCounter16CkilClk

16 ckil clocks

enumerator kGPC_StandbyCounter32CkilClk

32 ckil clocks

enumerator kGPC_StandbyCounter64CkilClk

64 ckil clocks

enumerator kGPC_StandbyCounter128CkilClk

128 ckil clocks

enumerator kGPC_StandbyCounter256CkilClk

256 ckil clocks

enumerator kGPC_StandbyCounter512CkilClk

512 ckil clocks

typedef struct *_gpc_lpm_config* gpc_lpm_config_t

configuration for enter DSM mode

typedef struct *_gpc_dsm_config* gpc_dsm_config_t

static inline void GPC_AllowIRQs(GPC_Type *base)

Allow all the IRQ/Events within the charge of GPC.

Parameters

- base – GPC peripheral base address.

static inline void GPC_DisallowIRQs(GPC_Type *base)

Disallow all the IRQ/Events within the charge of GPC.

Parameters

- base – GPC peripheral base address.

static inline uint32_t GPC_GetLpmMode(GPC_Type *base)

Get current LPM mode.

Parameters

- base – GPC peripheral base address.

Return values

lpm – mode, reference _gpc_lpm_mode

void GPC_EnableIRQ(GPC_Type *base, uint32_t irqId)

Enable the IRQ.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be enabled, available range is 0-127,reference SOC headerfile IRQn_Type.

void GPC_DisableIRQ(GPC_Type *base, uint32_t irqId)

Disable the IRQ.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be disabled, available range is 0-127,reference SOC headerfile IRQn_Type.

bool GPC_GetIRQStatusFlag(GPC_Type *base, uint32_t irqId)

Get the IRQ/Event flag.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be enabled, available range is 0-127,reference SOC headerfile IRQn_Type.

Returns

Indicated IRQ/Event is asserted or not.

static inline void GPC_DsmTriggerMask(GPC_Type *base, bool enable)

Mask the DSM trigger.

Parameters

- base – GPC peripheral base address.
- enable – true to enable mask, false to disable mask.

static inline void GPC_WFIMask(GPC_Type *base, bool enable)

Mask the WFI.

Parameters

- base – GPC peripheral base address.
- enable – true to enable mask, false to disable mask.

static inline void GPC_SelectPGCAckSignal(GPC_Type *base, uint32_t mask)
Select the PGC ACK signal.

Parameters

- base – GPC peripheral base address.
- mask – reference _gpc_pgc_ack_sel.

static inline void GPC_PowerDownRequestMask(GPC_Type *base, bool enable)
Power down request to virtual PGC mask or not.

Parameters

- base – GPC peripheral base address.
- enable – true to mask, false to not mask.

static inline void GPC_PGCMapping(GPC_Type *base, uint32_t mask)
PGC CPU Mapping.

Parameters

- base – GPC peripheral base address.
- mask – mask value reference PGC CPU mapping definition.

static inline void GPC_TimeSlotConfigureForPUS(GPC_Type *base, uint8_t slotIndex, uint32_t value)
Time slot configure.

Parameters

- base – GPC peripheral base address.
- slotIndex – time slot index.
- value – value to be configured

void GPC_EnterWaitMode(GPC_Type *base, *gpc_lpm_config_t* *config)
Enter WAIT mode.

Parameters

- base – GPC peripheral base address.
- config – lpm mode configurations.

void GPC_EnterStopMode(GPC_Type *base, *gpc_lpm_config_t* *config)
Enter STOP mode.

Parameters

- base – GPC peripheral base address.
- config – lpm mode configurations.

void GPC_Init(GPC_Type *base, uint32_t powerUpSlot, uint32_t powerDownSlot)
GPC init function.

Parameters

- base – GPC peripheral base address.
- powerUpSlot – power up slot number.
- powerDownSlot – power down slot number.

GPC_PCG_TIME_SLOT_TOTAL_NUMBER
Total number of the timeslot.

struct _gpc_lpm_config

#include <fsl_gpc.h>

Public Members

bool enFastWakeUp

enable fast wake up from lpm mode

bool enCpuClk

enable CPU clock when LPM enter

bool enVirtualPGCPowerup

enable virtual PGC power up with LPM enter

bool enVirtualPGCPowerdown

enable virtual PGC power down with LPM enter

bool enWfiMask

enable WFI Mask

bool enDsmMask

enable DSM Mask

struct _gpc_dsm_config

#include <fsl_gpc.h>

Public Members

bool disableRamLpctl

Memory can be defined to go to retention mode or not

bool enPMICStandBy

PMIC can be defined to be stand-by mode or not

uint8_t pmicStandByCounter

PMIC standby counter, reference _gpc_standby_count

uint8_t regBypassCounter

if PMIC standby is request, regulator bypass should be enable, and the counter can be defined

2.11 GPIO: General-Purpose Input/Output Driver

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *Config)

Initializes the GPIO peripheral according to the specified parameters in the initConfig.

Parameters

- base – GPIO base pointer.
- pin – Specifies the pin number
- Config – pointer to a gpio_pin_config_t structure that contains the configuration information.

void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the individual GPIO pin to logic 1 or 0.

Parameters

- base – GPIO base pointer.
- pin – GPIO port pin number.
- output – GPIOpin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_WritePinOutput(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the individual GPIO pin to logic 1 or 0.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinWrite.

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 (GPIO1, GPIO2, GPIO3, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_SetPinsOutput(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Deprecated:

Do not use this function. It has been superceded by GPIO_PortSet.

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 (GPIO1, GPIO2, GPIO3, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_ClearPinsOutput(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Deprecated:

Do not use this function. It has been superceded by GPIO_PortClear.

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 (GPIO1, GPIO2, GPIO3, 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 base pointer.
- pin – GPIO port pin number.

Return values

GPIO – port input value.

static inline uint32_t GPIO_ReadPinInput(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinRead.

static inline uint8_t GPIO_PinReadPadStatus(GPIO_Type *base, uint32_t pin)

Reads the current GPIO pin pad status.

Parameters

- base – GPIO base pointer.
- pin – GPIO port pin number.

Return values

GPIO – pin pad status value.

static inline uint8_t GPIO_ReadPadStatus(GPIO_Type *base, uint32_t pin)

Reads the current GPIO pin pad status.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinReadPadStatus.

void GPIO_PinSetInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_mode_t* pinInterruptMode)

Sets the current pin interrupt mode.

Parameters

- base – GPIO base pointer.
- pin – GPIO port pin number.
- pinInterruptMode – pointer to a gpio_interrupt_mode_t structure that contains the interrupt mode information.

static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_mode_t* pinInterruptMode)

Sets the current pin interrupt mode.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinSetInterruptConfig.

static inline void GPIO_PortEnableInterrupts(GPIO_Type *base, uint32_t mask)

Enables the specific pin interrupt.

Parameters

- base – GPIO base pointer.

- mask – GPIO pin number macro.

static inline void GPIO_EnableInterrupts(GPIO_Type *base, uint32_t mask)

Enables the specific pin interrupt.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

static inline void GPIO_PortDisableInterrupts(GPIO_Type *base, uint32_t mask)

Disables the specific pin interrupt.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

static inline void GPIO_DisableInterrupts(GPIO_Type *base, uint32_t mask)

Disables the specific pin interrupt.

Deprecated:

Do not use this function. It has been superceded by GPIO_PortDisableInterrupts.

static inline uint32_t GPIO_PortGetInterruptFlags(GPIO_Type *base)

Reads individual pin interrupt status.

Parameters

- base – GPIO base pointer.

Return values

current – pin interrupt status flag.

static inline uint32_t GPIO_GetPinsInterruptFlags(GPIO_Type *base)

Reads individual pin interrupt status.

Parameters

- base – GPIO base pointer.

Return values

current – pin interrupt status flag.

static inline void GPIO_PortClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

static inline void GPIO_ClearPinsInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

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_interrupt_mode
GPIO interrupt mode definition.

Values:

enumerator kGPIO_NoIntmode
Set current pin general IO functionality.

enumerator kGPIO_IntLowLevel
Set current pin interrupt is low-level sensitive.

enumerator kGPIO_IntHighLevel
Set current pin interrupt is high-level sensitive.

enumerator kGPIO_IntRisingEdge
Set current pin interrupt is rising-edge sensitive.

enumerator kGPIO_IntFallingEdge
Set current pin interrupt is falling-edge sensitive.

enumerator kGPIO_IntRisingOrFallingEdge
Enable the edge select bit to override the ICR register's configuration.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_interrupt_mode* gpio_interrupt_mode_t
GPIO interrupt mode definition.

typedef struct *_gpio_pin_config* gpio_pin_config_t
GPIO Init structure definition.

struct _gpio_pin_config
#include <fsl_gpio.h> GPIO Init structure definition.

Public Members

gpio_pin_direction_t direction
Specifies the pin direction.

uint8_t outputLogic
Set a default output logic, which has no use in input

gpio_interrupt_mode_t interruptMode
Specifies the pin interrupt mode, a value of gpio_interrupt_mode_t.

2.12 GPT: General Purpose Timer

void GPT_Init(GPT_Type *base, const *gpt_config_t* *initConfig)

Initialize GPT to reset state and initialize running mode.

Parameters

- base – GPT peripheral base address.
- initConfig – GPT mode setting configuration.

void GPT_Deinit(GPT_Type *base)

Disables the module and gates the GPT clock.

Parameters

- base – GPT peripheral base address.

void GPT_GetDefaultConfig(*gpt_config_t* *config)

Fills in the GPT configuration structure with default settings.

The default values are:

```
config->clockSource = kGPT_ClockSource_Periph;
config->divider = 1U;
config->enableRunInStop = true;
config->enableRunInWait = true;
config->enableRunInDoze = false;
config->enableRunInDbg = false;
config->enableFreeRun = false;
config->enableMode  = true;
```

Parameters

- config – Pointer to the user configuration structure.

static inline void GPT_SoftwareReset(GPT_Type *base)

Software reset of GPT module.

Parameters

- base – GPT peripheral base address.

static inline void GPT_SetClockSource(GPT_Type *base, *gpt_clock_source_t* gptClkSource)

Set clock source of GPT.

Parameters

- base – GPT peripheral base address.
- gptClkSource – Clock source (see gpt_clock_source_t typedef enumeration).

static inline *gpt_clock_source_t* GPT_GetClockSource(GPT_Type *base)

Get clock source of GPT.

Parameters

- base – GPT peripheral base address.

Returns

clock source (see gpt_clock_source_t typedef enumeration).

static inline void GPT_SetClockDivider(GPT_Type *base, uint32_t divider)

Set pre scaler of GPT.

Parameters

- base – GPT peripheral base address.

- divider – Divider of GPT (1-4096).

static inline uint32_t GPT_GetClockDivider(GPT_Type *base)

Get clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

clock divider in GPT module (1-4096).

static inline void GPT_SetOscClockDivider(GPT_Type *base, uint32_t divider)

OSC 24M pre-scaler before selected by clock source.

Parameters

- base – GPT peripheral base address.
- divider – OSC Divider(1-16).

static inline uint32_t GPT_GetOscClockDivider(GPT_Type *base)

Get OSC 24M clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

OSC clock divider in GPT module (1-16).

static inline void GPT_StartTimer(GPT_Type *base)

Start GPT timer.

Parameters

- base – GPT peripheral base address.

static inline void GPT_StopTimer(GPT_Type *base)

Stop GPT timer.

Parameters

- base – GPT peripheral base address.

static inline uint32_t GPT_GetCurrentTimerCount(GPT_Type *base)

Reads the current GPT counting value.

Parameters

- base – GPT peripheral base address.

Returns

Current GPT counter value.

static inline void GPT_SetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel, *gpt_input_operation_mode_t* mode)

Set GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).
- mode – GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

static inline *gpt_input_operation_mode_t* GPT_GetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

static inline uint32_t GPT_GetInputCaptureValue(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT input capture value of certain channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

static inline void GPT_SetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel, *gpt_output_operation_mode_t* mode)

Set GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
- mode – GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

static inline *gpt_output_operation_mode_t* GPT_GetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

static inline void GPT_SetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel, uint32_t value)

Set GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.

- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
- value – GPT output compare value.

static inline uint32_t GPT_GetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output compare value.

static inline void GPT_ForceOutput(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Force GPT output action on output compare channel, ignoring comparator.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

static inline void GPT_EnableInterrupts(GPT_Type *base, uint32_t mask)

Enables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration gpt_interrupt_enable_t

static inline void GPT_DisableInterrupts(GPT_Type *base, uint32_t mask)

Disables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration gpt_interrupt_enable_t

static inline uint32_t GPT_GetEnabledInterrupts(GPT_Type *base)

Gets the enabled GPT interrupts.

Parameters

- base – GPT peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration gpt_interrupt_enable_t

static inline uint32_t GPT_GetStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Get GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see gpt_status_flag_t for bit definition).

Returns
GPT status, each bit represents one status flag.

static inline void GPT_ClearStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)
Clears the GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see gpt_status_flag_t for bit definition).

FSL_GPT_DRIVER_VERSION

enum _gpt_clock_source
List of clock sources.

Note: Actual number of clock sources is SoC dependent

Values:

enumerator kGPT_ClockSource_Off
GPT Clock Source Off.

enumerator kGPT_ClockSource_Periph
GPT Clock Source from Peripheral Clock.

enumerator kGPT_ClockSource_HighFreq
GPT Clock Source from High Frequency Reference Clock.

enumerator kGPT_ClockSource_Ext
GPT Clock Source from external pin.

enumerator kGPT_ClockSource_LowFreq
GPT Clock Source from Low Frequency Reference Clock.

enumerator kGPT_ClockSource_Osc
GPT Clock Source from Crystal oscillator.

enum _gpt_input_capture_channel
List of input capture channel number.

Values:

enumerator kGPT_InputCapture_Channel1
GPT Input Capture Channel1.

enumerator kGPT_InputCapture_Channel2
GPT Input Capture Channel2.

enum _gpt_input_operation_mode
List of input capture operation mode.

Values:

enumerator kGPT_InputOperation_Disabled
Don't capture.

enumerator kGPT_InputOperation_RiseEdge
Capture on rising edge of input pin.

enumerator kGPT_InputOperation_FallEdge
Capture on falling edge of input pin.

enumerator kGPT_InputOperation_BothEdge
Capture on both edges of input pin.

enum _gpt_output_compare_channel
List of output compare channel number.

Values:

enumerator kGPT_OutputCompare_Channel1
Output Compare Channel1.

enumerator kGPT_OutputCompare_Channel2
Output Compare Channel2.

enumerator kGPT_OutputCompare_Channel3
Output Compare Channel3.

enum _gpt_output_operation_mode
List of output compare operation mode.

Values:

enumerator kGPT_OutputOperation_Disconnected
Don't change output pin.

enumerator kGPT_OutputOperation_Toggle
Toggle output pin.

enumerator kGPT_OutputOperation_Clear
Set output pin low.

enumerator kGPT_OutputOperation_Set
Set output pin high.

enumerator kGPT_OutputOperation_Activelow
Generate a active low pulse on output pin.

enum _gpt_interrupt_enable
List of GPT interrupts.

Values:

enumerator kGPT_OutputCompare1InterruptEnable
Output Compare Channel1 interrupt enable

enumerator kGPT_OutputCompare2InterruptEnable
Output Compare Channel2 interrupt enable

enumerator kGPT_OutputCompare3InterruptEnable
Output Compare Channel3 interrupt enable

enumerator kGPT_InputCapture1InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_InputCapture2InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_RollOverFlagInterruptEnable
Counter rolled over interrupt enable

enum _gpt_status_flag
Status flag.

Values:

enumerator kGPT_OutputCompare1Flag
Output compare channel 1 event.

enumerator kGPT_OutputCompare2Flag
Output compare channel 2 event.

enumerator kGPT_OutputCompare3Flag
Output compare channel 3 event.

enumerator kGPT_InputCapture1Flag
Input Capture channel 1 event.

enumerator kGPT_InputCapture2Flag
Input Capture channel 2 event.

enumerator kGPT_RollOverFlag
Counter reaches maximum value and rolled over to 0 event.

typedef enum *_gpt_clock_source* gpt_clock_source_t
List of clock sources.

Note: Actual number of clock sources is SoC dependent

typedef enum *_gpt_input_capture_channel* gpt_input_capture_channel_t
List of input capture channel number.

typedef enum *_gpt_input_operation_mode* gpt_input_operation_mode_t
List of input capture operation mode.

typedef enum *_gpt_output_compare_channel* gpt_output_compare_channel_t
List of output compare channel number.

typedef enum *_gpt_output_operation_mode* gpt_output_operation_mode_t
List of output compare operation mode.

typedef enum *_gpt_interrupt_enable* gpt_interrupt_enable_t
List of GPT interrupts.

typedef enum *_gpt_status_flag* gpt_status_flag_t
Status flag.

typedef struct *_gpt_init_config* gpt_config_t
Structure to configure the running mode.

struct _gpt_init_config
#include <fsl_gpt.h> Structure to configure the running mode.

Public Members

gpt_clock_source_t clockSource
clock source for GPT module.

uint32_t divider
clock divider (prescaler+1) from clock source to counter.

bool enableFreeRun
true: FreeRun mode, false: Restart mode.

bool enableRunInWait
GPT enabled in wait mode.

bool enableRunInStop
GPT enabled in stop mode.

bool enableRunInDoze
GPT enabled in doze mode.

bool enableRunInDbg
GPT enabled in debug mode.

bool enableMode
true: counter reset to 0 when enabled; false: counter retain its value when enabled.

2.13 I2C: Inter-Integrated Circuit Driver

2.14 I2C Driver

void I2C_MasterInit(I2C_Type *base, const *i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the I2C peripheral. Call this API to ungate the I2C clock and configure the I2C with master configuration.

Note: This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the I2C_MasterGetDefaultConfig(). After calling this API, the master is ready to transfer. This is an example.

```
i2c_master_config_t config = {
.enableMaster = true,
.baudRate_Bps = 100000
};
I2C_MasterInit(I2C0, &config, 12000000U);
```

Parameters

- base – I2C base pointer
- masterConfig – A pointer to the master configuration structure
- srcClock_Hz – I2C peripheral clock frequency in Hz

void I2C_MasterDeinit(I2C_Type *base)

De-initializes the I2C master peripheral. Call this API to gate the I2C clock. The I2C master module can't work unless the I2C_MasterInit is called.

Parameters

- base – I2C base pointer

void I2C_MasterGetDefaultConfig(*i2c_master_config_t* *masterConfig)

Sets the I2C master configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C_MasterInit(). Use the initialized structure unchanged in the I2C_MasterInit() or modify the structure before calling the I2C_MasterInit(). This is an example.

```
i2c_master_config_t config;
I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – A pointer to the master configuration structure.

void I2C_SlaveInit(I2C_Type *base, const *i2c_slave_config_t* *slaveConfig)

Initializes the I2C peripheral. Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

Note: This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by I2C_SlaveGetDefaultConfig() or it can be custom filled by the user. This is an example.

```
i2c_slave_config_t config = {
.enableSlave = true,
.slaveAddress = 0x1DU,
};
I2C_SlaveInit(I2C0, &config);
```

Parameters

- base – I2C base pointer
- slaveConfig – A pointer to the slave configuration structure

void I2C_SlaveDeinit(I2C_Type *base)

De-initializes the I2C slave peripheral. Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C_SlaveInit is called to enable the clock.

Parameters

- base – I2C base pointer

void I2C_SlaveGetDefaultConfig(*i2c_slave_config_t* *slaveConfig)

Sets the I2C slave configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C_SlaveInit(). Modify fields of the structure before calling the I2C_SlaveInit(). This is an example.

```
i2c_slave_config_t config;
I2C_SlaveGetDefaultConfig(&config);
```

Parameters

- slaveConfig – A pointer to the slave configuration structure.

static inline void I2C_Enable(I2C_Type *base, bool enable)

Enables or disables the I2C peripheral operation.

Parameters

- base – I2C base pointer
- enable – Pass true to enable and false to disable the module.

static inline uint32_t I2C_MasterGetStatusFlags(I2C_Type *base)

Gets the I2C status flags.

Parameters

- base – I2C base pointer

Returns

status flag, use status flag to AND _i2c_flags to get the related status.

static inline void I2C_MasterClearStatusFlags(I2C_Type *base, uint32_t statusMask)

Clears the I2C status flag state.

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag.

Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:
  - kI2C_ArbitrationLostFlag
  - kI2C_IntPendingFlag

static inline uint32_t I2C_SlaveGetStatusFlags(I2C_Type *base)

Gets the I2C status flags.

Parameters

- base – I2C base pointer

Returns

status flag, use status flag to AND _i2c_flags to get the related status.

static inline void I2C_SlaveClearStatusFlags(I2C_Type *base, uint32_t statusMask)

Clears the I2C status flag state.

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag

Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:
  - kI2C_IntPendingFlagFlag

void I2C_EnableInterrupts(I2C_Type *base, uint32_t mask)

Enables I2C interrupt requests.

Parameters

- base – I2C base pointer
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C_GlobalInterruptEnable
  - kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable
  - kI2C_SdaTimeoutInterruptEnable

void I2C_DisableInterrupts(I2C_Type *base, uint32_t mask)

Disables I2C interrupt requests.

Parameters

- base – I2C base pointer
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C_GlobalInterruptEnable
  - kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable

– kI2C_SdaTimeoutInterruptEnable

void I2C_MasterSetBaudRate(I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the I2C master transfer baud rate.

Parameters

- base – I2C base pointer
- baudRate_Bps – the baud rate value in bps
- srcClock_Hz – Source clock

status_t I2C_MasterStart(I2C_Type *base, uint8_t address, *i2c_direction_t* direction)

Sends a START on the I2C bus.

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.
- direction – Master transfer directions(transmit/receive).

Return values

- kStatus_Success – Successfully send the start signal.
- kStatus_I2C_Busy – Current bus is busy.

status_t I2C_MasterStop(I2C_Type *base)

Sends a STOP signal on the I2C bus.

Return values

- kStatus_Success – Successfully send the stop signal.
- kStatus_I2C_Timeout – Send stop signal failed, timeout.

status_t I2C_MasterRepeatedStart(I2C_Type *base, uint8_t address, *i2c_direction_t* direction)

Sends a REPEATED START on the I2C bus.

Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.
- direction – Master transfer directions(transmit/receive).

Return values

- kStatus_Success – Successfully send the start signal.
- kStatus_I2C_Busy – Current bus is busy but not occupied by current I2C master.

status_t I2C_MasterWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize, uint32_t flags)

Performs a polling send transaction on the I2C bus.

Parameters

- base – The I2C peripheral base pointer.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

- flags – Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

status_t I2C_MasterReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transaction on the I2C bus.

Note: The I2C_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

Parameters

- base – I2C peripheral base pointer.
- rxBuff – The pointer to the data to store the received data.
- rxSize – The length in bytes of the data to be received.
- flags – Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_Timeout – Send stop signal failed, timeout.

status_t I2C_SlaveWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize)

Performs a polling send transaction on the I2C bus.

Parameters

- base – The I2C peripheral base pointer.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

status_t I2C_SlaveReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize)

Performs a polling receive transaction on the I2C bus.

Parameters

- base – I2C peripheral base pointer.
- rxBuff – The pointer to the data to store the received data.
- rxSize – The length in bytes of the data to be received.

status_t I2C_MasterTransferBlocking(I2C_Type *base, *i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

- base – I2C peripheral base address.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

void I2C_MasterTransferCreateHandle(I2C_Type *base, *i2c_master_handle_t* *handle, *i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t I2C_MasterTransferNonBlocking(I2C_Type *base, *i2c_master_handle_t* *handle, *i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call I2C_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_I2C_Busy, the transfer is finished.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state.
- xfer – pointer to i2c_master_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.

status_t I2C_MasterTransferGetCount(I2C_Type *base, *i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

status_t I2C_MasterTransferAbort(I2C_Type *base, *i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

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

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state

Return values

- kStatus_I2C_Timeout – Timeout during polling flag.
- kStatus_Success – Successfully abort the transfer.

void I2C_MasterTransferHandleIRQ(I2C_Type *base, void *i2cHandle)

Master interrupt handler.

Parameters

- base – I2C base pointer.
- i2cHandle – pointer to i2c_master_handle_t structure.

void I2C_SlaveTransferCreateHandle(I2C_Type *base, *i2c_slave_handle_t* *handle, *i2c_slave_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_slave_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t I2C_SlaveTransferNonBlocking(I2C_Type *base, *i2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling the I2C_SlaveInit() and I2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes

events to the callback that was passed into the call to I2C_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 i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

- base – The I2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together i2c_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 kI2C_SlaveAllEvents to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_I2C_Busy – Slave transfers have already been started on this handle.

void I2C_SlaveTransferAbort(I2C_Type *base, *i2c_slave_handle_t* *handle)

Aborts the slave transfer.

Note: This API can be called at any time to stop slave for handling the bus events.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_slave_handle_t structure which stores the transfer state.

status_t I2C_SlaveTransferGetCount(I2C_Type *base, *i2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_slave_handle_t structure.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void I2C_SlaveTransferHandleIRQ(I2C_Type *base, void *i2cHandle)

Slave interrupt handler.

Parameters

- base – I2C base pointer.
- i2cHandle – pointer to i2c_slave_handle_t structure which stores the transfer state

FSL_I2C_DRIVER_VERSION
I2C driver version.

I2C status return codes.

Values:

enumerator kStatus_I2C_Busy
I2C is busy with current transfer.

enumerator kStatus_I2C_Idle
Bus is Idle.

enumerator kStatus_I2C_Nak
NAK received during transfer.

enumerator kStatus_I2C_ArbitrationLost
Arbitration lost during transfer.

enumerator kStatus_I2C_Timeout
Timeout polling status flags.

enumerator kStatus_I2C_Addr_Nak
NAK received during the address probe.

enum _i2c_flags
I2C peripheral flags.

The following status register flags can be cleared:

- kI2C_ArbitrationLostFlag
- kI2C_IntPendingFlag

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2C_ReceiveNakFlag
I2C receive NAK flag.

enumerator kI2C_IntPendingFlag
I2C interrupt pending flag.

enumerator kI2C_TransferDirectionFlag
I2C transfer direction flag.

enumerator kI2C_ArbitrationLostFlag
I2C arbitration lost flag.

enumerator kI2C_BusBusyFlag
I2C bus busy flag.

enumerator kI2C_AddressMatchFlag
I2C address match flag.

enumerator kI2C_TransferCompleteFlag
I2C transfer complete flag.

enum _i2c_interrupt_enable
I2C feature interrupt source.

Values:

enumerator kI2C_GlobalInterruptEnable

I2C global interrupt.

enum _i2c_direction

The direction of master and slave transfers.

Values:

enumerator kI2C_Write

Master transmits to the slave.

enumerator kI2C_Read

Master receives from the slave.

enum _i2c_master_transfer_flags

I2C transfer control flag.

Values:

enumerator kI2C_TransferDefaultFlag

A transfer starts with a start signal, stops with a stop signal.

enumerator kI2C_TransferNoStartFlag

A transfer starts without a start signal, only support write only or write+read with no start flag, do not support read only with no start flag.

enumerator kI2C_TransferRepeatedStartFlag

A transfer starts with a repeated start signal.

enumerator kI2C_TransferNoStopFlag

A transfer ends without a stop signal.

enum _i2c_slave_transfer_event

Set of events sent to the callback for nonblocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I2C_SlaveTransferNonBlocking() 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 kI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kI2C_SlaveTransmitEvent

A callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI2C_SlaveReceiveEvent

A callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI2C_SlaveTransmitAckEvent

A callback needs to either transmit an ACK or NACK.

enumerator kI2C_SlaveCompletionEvent

A stop was detected or finished transfer, completing the transfer.

enumerator kI2C_SlaveAllEvents

A bit mask of all available events.

typedef enum *_i2c_direction* i2c_direction_t

The direction of master and slave transfers.

typedef struct *_i2c_master_config* i2c_master_config_t

I2C master user configuration.

typedef struct *_i2c_master_handle* i2c_master_handle_t

I2C master handle typedef.

typedef void (*i2c_master_transfer_callback_t)(I2C_Type *base, *i2c_master_handle_t* *handle, *status_t* status, void *userData)

I2C master transfer callback typedef.

typedef struct *_i2c_master_transfer* i2c_master_transfer_t

I2C master transfer structure.

typedef enum *_i2c_slave_transfer_event* i2c_slave_transfer_event_t

Set of events sent to the callback for nonblocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I2C_SlaveTransferNonBlocking() 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 *_i2c_slave_handle* i2c_slave_handle_t

I2C slave handle typedef.

typedef struct *_i2c_slave_config* i2c_slave_config_t

I2C slave user configuration.

typedef struct *_i2c_slave_transfer* i2c_slave_transfer_t

I2C slave transfer structure.

typedef void (*i2c_slave_transfer_callback_t)(I2C_Type *base, *i2c_slave_transfer_t* *xfer, void *userData)

I2C slave transfer callback typedef.

I2C_RETRY_TIMES

Retry times for waiting flag.

struct i2c_master_config

#include <fsl_i2c.h> I2C master user configuration.

Public Members

bool enableMaster

Enables the I2C peripheral at initialization time.

uint32_t baudRate_Bps

Baud rate configuration of I2C peripheral.

struct i2c_master_transfer

#include <fsl_i2c.h> I2C master transfer structure.

Public Members

uint32_t flags
: A transfer flag which controls the transfer.

uint8_t slaveAddress
: 7-bit slave address.

i2c_direction_t direction
: A transfer direction, read or write.

uint32_t subaddress
: A sub address. Transferred MSB first.

uint8_t subaddressSize
: A size of the command buffer.

uint8_t *volatile data
: A transfer buffer.

volatile size_t dataSize
: A transfer size.

struct __i2c_master_handle
: *#include <fsl_i2c.h>* I2C master handle structure.

Public Members

i2c_master_transfer_t transfer
: I2C master transfer copy.

size_t transferSize
: Total bytes to be transferred.

uint8_t state
: A transfer state maintained during transfer.

i2c_master_transfer_callback_t completionCallback
: A callback function called when the transfer is finished.

void *userData
: A callback parameter passed to the callback function.

struct __i2c_slave_config
: *#include <fsl_i2c.h>* I2C slave user configuration.

Public Members

bool enableSlave
: Enables the I2C peripheral at initialization time.

uint16_t slaveAddress
: A slave address configuration.

struct __i2c_slave_transfer
: *#include <fsl_i2c.h>* I2C slave transfer structure.

Public Members

i2c_slave_transfer_event_t event
A reason that the callback is invoked.

uint8_t *volatile data
A transfer buffer.

volatile size_t dataSize
A transfer size.

status_t completionStatus
Success or error code describing how the transfer completed. Only applies for kI2C_SlaveCompletionEvent.

size_t transferredCount
A number of bytes actually transferred since the start or since the last repeated start.

struct __i2c_slave_handle
#include <fsl_i2c.h> I2C slave handle structure.

Public Members

volatile uint8_t state
A transfer state maintained during transfer.

i2c_slave_transfer_t transfer
I2C slave transfer copy.

uint32_t eventMask
A mask of enabled events.

i2c_slave_transfer_callback_t callback
A callback function called at the transfer event.

void *userData
A callback parameter passed to the callback.

2.15 Iomuxc_driver

static inline void IOMUXC_SetPinMux(uintptr_t muxRegister, uint32_t muxMode, uintptr_t inputRegister, uint32_t inputDaisy, uintptr_t configRegister, uint32_t inputOnfield)
Sets the IOMUXC pin mux mode.

This is an example to set the I2C4_SDA as the pwm1_OUT:

```
IOMUXC_SetPinMux(IOMUXC_I2C4_SDA_PWM1_OUT, 0);
```

Note: The first five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register_
- muxMode – The pin mux mode_

- inputRegister – The select input register_
- inputDaisy – The input daisy_
- configRegister – The config register_
- inputOnfield – The pad->module input inversion_

static inline void IOMUXC_SetPinConfig(uintptr_t muxRegister, uint32_t muxMode, uintptr_t inputRegister, uint32_t inputDaisy, uintptr_t configRegister, uint32_t configValue)

Sets the IOMUXC pin configuration.

This is an example to set pin configuration for IOMUXC_I2C4_SDA_PWM1_OUT:

```
IOMUXC_SetPinConfig(IOMUXC_I2C4_SDA_PWM1_OUT, IOMUXC_SW_PAD_CTL_PAD_
↪ODE_MASK | IOMUXC0_SW_PAD_CTL_PAD_DSE(2U))
```

Note: The previous five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register_
- muxMode – The pin mux mode_
- inputRegister – The select input register_
- inputDaisy – The input daisy_
- configRegister – The config register_
- configValue – The pin config value_

FSL_IOMUXC_DRIVER_VERSION

IOMUXC driver version 2.0.1.

IOMUXC_BOOT_MODE0_SRC_BOOT_MODE0

IOMUXC_BOOT_MODE1_SRC_BOOT_MODE1

IOMUXC_BOOT_MODE2_SRC_BOOT_MODE2

IOMUXC_BOOT_MODE2_I2C1_SCL

IOMUXC_BOOT_MODE3_SRC_BOOT_MODE3

IOMUXC_BOOT_MODE3_I2C1_SDA

IOMUXC_JTAG_MOD_JTAG_MODE

IOMUXC_JTAG_TDI_JTAG_TDI

IOMUXC_JTAG_TMS_JTAG_TMS

IOMUXC_JTAG_TCK_JTAG_TCK

IOMUXC_JTAG_TDO_JTAG_TDO

IOMUXC_RTC_XTALI_SNVS_RTC

IOMUXC_PMIC_STBY_REQ_CCM_PMIC_STBY_REQ

IOMUXC_PMIC_ON_REQ_SNVS_PMIC_ON_REQ

IOMUXC_ONOFF_SNVS_ONOFF

IOMUXC_POR_B_SNVS_POR_B

IOMUXC_RTC_RESET_B_SNVS_RTC_RESET_B

IOMUXC_GPIO1_IO00_GPIO1_IO00

IOMUXC_GPIO1_IO00_CCM_ENET_PHY_REF_CLK_ROOT

IOMUXC_GPIO1_IO00_CCM_REF_CLK_32K

IOMUXC_GPIO1_IO00_CCM_EXT_CLK1

IOMUXC_GPIO1_IO01_GPIO1_IO01

IOMUXC_GPIO1_IO01_PWM1_OUT

IOMUXC_GPIO1_IO01_CCM_REF_CLK_24M

IOMUXC_GPIO1_IO01_CCM_EXT_CLK2

IOMUXC_GPIO1_IO02_GPIO1_IO02

IOMUXC_GPIO1_IO02_WDOG1_WDOG_B

IOMUXC_GPIO1_IO02_WDOG1_WDOG_ANY

IOMUXC_GPIO1_IO02_SJC_DE_B

IOMUXC_GPIO1_IO03_GPIO1_IO03

IOMUXC_GPIO1_IO03_USDHC1_VSELECT

IOMUXC_GPIO1_IO03_SDMA1_EXT_EVENT0

IOMUXC_GPIO1_IO04_GPIO1_IO04

IOMUXC_GPIO1_IO04_USDHC2_VSELECT

IOMUXC_GPIO1_IO04_SDMA1_EXT_EVENT1

IOMUXC_GPIO1_IO05_GPIO1_IO05

IOMUXC_GPIO1_IO05_M7_NMI

IOMUXC_GPIO1_IO05_CCM_PMIC_READY

IOMUXC_GPIO1_IO06_GPIO1_IO06

IOMUXC_GPIO1_IO06_ENET1_MDC

IOMUXC_GPIO1_IO06_USDHC1_CD_B

IOMUXC_GPIO1_IO06_CCM_EXT_CLK3

IOMUXC_GPIO1_IO07_GPIO1_IO07

IOMUXC_GPIO1_IO07_ENET1_MDIO

IOMUXC_GPIO1_IO07_USDHC1_WP

IOMUXC_GPIO1_IO07_CCM_EXT_CLK4

IOMUXC_GPIO1_IO08_GPIO1_IO08

IOMUXC_GPIO1_IO08_ENET1_1588_EVENT0_IN

IOMUXC_GPIO1_IO08_PWM1_OUT

IOMUXC_GPIO1_IO08_USDHC2_RESET_B

IOMUXC_GPIO1_IO09_GPIO1_IO09

IOMUXC_GPIO1_IO09_ENET1_1588_EVENT0_OUT

IOMUXC_GPIO1_IO09_PWM2_OUT

IOMUXC_GPIO1_IO09_USDHC3_RESET_B

IOMUXC_GPIO1_IO09_SDMA2_EXT_EVENT0

IOMUXC_GPIO1_IO10_GPIO1_IO10

IOMUXC_GPIO1_IO10_USB1_OTG_ID

IOMUXC_GPIO1_IO10_PWM3_OUT

IOMUXC_GPIO1_IO11_GPIO1_IO11

IOMUXC_GPIO1_IO11_PWM2_OUT

IOMUXC_GPIO1_IO11_USDHC3_VSELECT

IOMUXC_GPIO1_IO11_CCM_PMIC_READY

IOMUXC_GPIO1_IO12_GPIO1_IO12

IOMUXC_GPIO1_IO12_USB1_OTG_PWR

IOMUXC_GPIO1_IO12_SDMA2_EXT_EVENT1

IOMUXC_GPIO1_IO13_GPIO1_IO13

IOMUXC_GPIO1_IO13_USB1_OTG_OC

IOMUXC_GPIO1_IO13_PWM2_OUT

IOMUXC_GPIO1_IO14_GPIO1_IO14

IOMUXC_GPIO1_IO14_USDHC3_CD_B

IOMUXC_GPIO1_IO14_PWM3_OUT

IOMUXC_GPIO1_IO14_CCM_CLKO1

IOMUXC_GPIO1_IO15_GPIO1_IO15

IOMUXC_GPIO1_IO15_USDHC3_WP

IOMUXC_GPIO1_IO15_PWM4_OUT

IOMUXC_GPIO1_IO15_CCM_CLKO2

IOMUXC_ENET_MDC_ENET1_MDC

IOMUXC_ENET_MDC_SAI6_TX_DATA0

IOMUXC_ENET_MDC_PDM_BIT_STREAM3

IOMUXC_ENET_MDC_SPDIF1_OUT

IOMUXC_ENET_MDC_GPIO1_IO16

IOMUXC_ENET_MDC_USDHC3_STROBE

IOMUXC_ENET_MDIO_ENET1_MDIO

IOMUXC_ENET_MDIO_SAI6_TX_SYNC

IOMUXC_ENET_MDIO_PDM_BIT_STREAM2

IOMUXC_ENET_MDIO_SPDIF1_IN

IOMUXC_ENET_MDIO_GPIO1_IO17

IOMUXC_ENET_MDIO_USDHC3_DATA5

IOMUXC_ENET_TD3_ENET1_RGMII_TD3

IOMUXC_ENET_TD3_SAI6_TX_BCLK

IOMUXC_ENET_TD3_PDM_BIT_STREAM1

IOMUXC_ENET_TD3_SPDIF1_EXT_CLK

IOMUXC_ENET_TD3_GPIO1_IO18

IOMUXC_ENET_TD3_USDHC3_DATA6

IOMUXC_ENET_TD2_ENET1_RGMII_TD2

IOMUXC_ENET_TD2_ENET1_TX_CLK

IOMUXC_ENET_TD2_SAI6_RX_DATA0

IOMUXC_ENET_TD2_PDM_BIT_STREAM3

IOMUXC_ENET_TD2_GPIO1_IO19

IOMUXC_ENET_TD2_USDHC3_DATA7

IOMUXC_ENET_TD1_ENET1_RGMII_TD1

IOMUXC_ENET_TD1_SAI6_RX_SYNC

IOMUXC_ENET_TD1_PDM_BIT_STREAM2

IOMUXC_ENET_TD1_GPIO1_IO20

IOMUXC_ENET_TD1_USDHC3_CD_B

IOMUXC_ENET_TD0_ENET1_RGMII_TD0

IOMUXC_ENET_TD0_SAI6_RX_BCLK

IOMUXC_ENET_TD0_PDM_BIT_STREAM1

IOMUXC_ENET_TD0_GPIO1_IO21

IOMUXC_ENET_TD0_USDHC3_WP

IOMUXC_ENET_TX_CTL_ENET1_RGMII_TX_CTL

IOMUXC_ENET_TX_CTL_SAI6_MCLK

IOMUXC_ENET_TX_CTL_GPIO1_IO22

IOMUXC_ENET_TX_CTL_USDHC3_DATA0

IOMUXC_ENET_TXC_ENET1_RGMII_TXC

IOMUXC_ENET_TXC_ENET1_TX_ER

IOMUXC_ENET_TXC_SAI7_TX_DATA0

IOMUXC_ENET_TXC_GPIO1_IO23

IOMUXC_ENET_TXC_USDHC3_DATA1

IOMUXC_ENET_RX_CTL_ENET1_RGMII_RX_CTL

IOMUXC_ENET_RX_CTL_SAI7_TX_SYNC

IOMUXC_ENET_RX_CTL_PDM_BIT_STREAM3

IOMUXC_ENET_RX_CTL_GPIO1_IO24

IOMUXC_ENET_RX_CTL_USDHC3_DATA2

IOMUXC_ENET_RXC_ENET1_RGMII_RXC

IOMUXC_ENET_RXC_ENET1_RX_ER

IOMUXC_ENET_RXC_SAI7_TX_BCLK

IOMUXC_ENET_RXC_PDM_BIT_STREAM2

IOMUXC_ENET_RXC_GPIO1_IO25

IOMUXC_ENET_RXC_USDHC3_DATA3

IOMUXC_ENET_RD0_ENET1_RGMII_RD0

IOMUXC_ENET_RD0_SAI7_RX_DATA0

IOMUXC_ENET_RD0_PDM_BIT_STREAM1

IOMUXC_ENET_RD0_GPIO1_IO26

IOMUXC_ENET_RD0_USDHC3_DATA4

IOMUXC_ENET_RD1_ENET1_RGMII_RD1

IOMUXC_ENET_RD1_SAI7_RX_SYNC

IOMUXC_ENET_RD1_PDM_BIT_STREAM0

IOMUXC_ENET_RD1_GPIO1_IO27

IOMUXC_ENET_RD1_USDHC3_RESET_B

IOMUXC_ENET_RD2_ENET1_RGMII_RD2

IOMUXC_ENET_RD2_SAI7_RX_BCLK

IOMUXC_ENET_RD2_PDM_CLK

IOMUXC_ENET_RD2_GPIO1_IO28

IOMUXC_ENET_RD2_USDHC3_CLK

IOMUXC_ENET_RD3_ENET1_RGMII_RD3

IOMUXC_ENET_RD3_SAI7_MCLK

IOMUXC_ENET_RD3_SPDIF1_IN

IOMUXC_ENET_RD3_GPIO1_IO29

IOMUXC_ENET_RD3_USDHC3_CMD

IOMUXC_SD1_CLK_USDHC1_CLK

IOMUXC_SD1_CLK_ENET1_MDC

IOMUXC_SD1_CLK_UART1_TX

IOMUXC_SD1_CLK_UART1_RX

IOMUXC_SD1_CLK_GPIO2_IO00

IOMUXC_SD1_CMD_USDHC1_CMD

IOMUXC_SD1_CMD_ENET1_MDIO

IOMUXC_SD1_CMD_UART1_RX

IOMUXC_SD1_CMD_UART1_TX

IOMUXC_SD1_CMD_GPIO2_IO01

IOMUXC_SD1_DATA0_USDHC1_DATA0

IOMUXC_SD1_DATA0_ENET1_RGMII_TD1

IOMUXC_SD1_DATA0_UART1_RTS_B

IOMUXC_SD1_DATA0_UART1_CTS_B

IOMUXC_SD1_DATA0_GPIO2_IO02

IOMUXC_SD1_DATA1_USDHC1_DATA1

IOMUXC_SD1_DATA1_ENET1_RGMII_TD0

IOMUXC_SD1_DATA1_UART1_CTS_B

IOMUXC_SD1_DATA1_UART1_RTS_B

IOMUXC_SD1_DATA1_GPIO2_IO03

IOMUXC_SD1_DATA2_USDHC1_DATA2

IOMUXC_SD1_DATA2_ENET1_RGMII_RD0

IOMUXC_SD1_DATA2_UART2_TX

IOMUXC_SD1_DATA2_UART2_RX

IOMUXC_SD1_DATA2_GPIO2_IO04

IOMUXC_SD1_DATA3_USDHC1_DATA3

IOMUXC_SD1_DATA3_ENET1_RGMII_RD1

IOMUXC_SD1_DATA3_UART2_RX

IOMUXC_SD1_DATA3_UART2_TX

IOMUXC_SD1_DATA3_GPIO2_IO05

IOMUXC_SD1_DATA4_USDHC1_DATA4

IOMUXC_SD1_DATA4_ENET1_RGMII_TX_CTL

IOMUXC_SD1_DATA4_I2C1_SCL

IOMUXC_SD1_DATA4_UART2_RTS_B

IOMUXC_SD1_DATA4_UART2_CTS_B

IOMUXC_SD1_DATA4_GPIO2_IO06

IOMUXC_SD1_DATA5_USDHC1_DATA5

IOMUXC_SD1_DATA5_ENET1_TX_ER

IOMUXC_SD1_DATA5_I2C1_SDA

IOMUXC_SD1_DATA5_UART2_CTS_B

IOMUXC_SD1_DATA5_UART2_RTS_B

IOMUXC_SD1_DATA5_GPIO2_IO07

IOMUXC_SD1_DATA6_USDHC1_DATA6

IOMUXC_SD1_DATA6_ENET1_RGMII_RX_CTL

IOMUXC_SD1_DATA6_I2C2_SCL

IOMUXC_SD1_DATA6_UART3_TX

IOMUXC_SD1_DATA6_UART3_RX

IOMUXC_SD1_DATA6_GPIO2_IO08

IOMUXC_SD1_DATA7_USDHC1_DATA7

IOMUXC_SD1_DATA7_ENET1_RX_ER

IOMUXC_SD1_DATA7_I2C2_SDA

IOMUXC_SD1_DATA7_UART3_RX

IOMUXC_SD1_DATA7_UART3_TX

IOMUXC_SD1_DATA7_GPIO2_IO09

IOMUXC_SD1_RESET_B_USDHC1_RESET_B

IOMUXC_SD1_RESET_B_ENET1_TX_CLK

IOMUXC_SD1_RESET_B_I2C3_SCL

IOMUXC_SD1_RESET_B_UART3_RTS_B

IOMUXC_SD1_RESET_B_UART3_CTS_B

IOMUXC_SD1_RESET_B_GPIO2_IO10

IOMUXC_SD1_STROBE_USDHC1_STROBE

IOMUXC_SD1_STROBE_I2C3_SDA

IOMUXC_SD1_STROBE_UART3_CTS_B

IOMUXC_SD1_STROBE_UART3_RTS_B

IOMUXC_SD1_STROBE_GPIO2_IO11

IOMUXC_SD2_CD_B_USDHC2_CD_B

IOMUXC_SD2_CD_B_GPIO2_IO12

IOMUXC_SD2_CLK_USDHC2_CLK

IOMUXC_SD2_CLK_SAI5_RX_SYNC

IOMUXC_SD2_CLK_ECSPI2_SCLK

IOMUXC_SD2_CLK_UART4_RX

IOMUXC_SD2_CLK_UART4_TX

IOMUXC_SD2_CLK_SAI5_MCLK

IOMUXC_SD2_CLK_GPIO2_IO13

IOMUXC_SD2_CMD_USDHC2_CMD

IOMUXC_SD2_CMD_SAI5_RX_BCLK

IOMUXC_SD2_CMD_ECSPI2_MOSI

IOMUXC_SD2_CMD_UART4_TX

IOMUXC_SD2_CMD_UART4_RX

IOMUXC_SD2_CMD_PDM_CLK

IOMUXC_SD2_CMD_GPIO2_IO14

IOMUXC_SD2_DATA0_USDHC2_DATA0

IOMUXC_SD2_DATA0_SAI5_RX_DATA0

IOMUXC_SD2_DATA0_I2C4_SDA

IOMUXC_SD2_DATA0_UART2_RX

IOMUXC_SD2_DATA0_UART2_TX

IOMUXC_SD2_DATA0_PDM_BIT_STREAM0

IOMUXC_SD2_DATA0_GPIO2_IO15

IOMUXC_SD2_DATA1_USDHC2_DATA1

IOMUXC_SD2_DATA1_SAI5_TX_SYNC

IOMUXC_SD2_DATA1_I2C4_SCL

IOMUXC_SD2_DATA1_UART2_TX

IOMUXC_SD2_DATA1_UART2_RX

IOMUXC_SD2_DATA1_PDM_BIT_STREAM1

IOMUXC_SD2_DATA1_GPIO2_IO16

IOMUXC_SD2_DATA2_USDHC2_DATA2

IOMUXC_SD2_DATA2_SAI5_TX_BCLK

IOMUXC_SD2_DATA2_ECSPI2_SS0

IOMUXC_SD2_DATA2_SPDIF1_OUT

IOMUXC_SD2_DATA2_PDM_BIT_STREAM2

IOMUXC_SD2_DATA2_GPIO2_IO17

IOMUXC_SD2_DATA3_USDHC2_DATA3

IOMUXC_SD2_DATA3_SAI5_TX_DATA0

IOMUXC_SD2_DATA3_ECSPI2_MISO

IOMUXC_SD2_DATA3_SPDIF1_IN

IOMUXC_SD2_DATA3_PDM_BIT_STREAM3

IOMUXC_SD2_DATA3_GPIO2_IO18

IOMUXC_SD2_DATA3_SRC_EARLY_RESET

IOMUXC_SD2_RESET_B_USDHC2_RESET_B

IOMUXC_SD2_RESET_B_GPIO2_IO19

IOMUXC_SD2_RESET_B_SRC_SYSTEM_RESET

IOMUXC_SD2_WP_USDHC2_WP

IOMUXC_SD2_WP_GPIO2_IO20

IOMUXC_SD2_WP_CORESIGHT_EVENTI

IOMUXC_NAND_ALE_NAND_ALE

IOMUXC_NAND_ALE_QSPI_A_SCLK

IOMUXC_NAND_ALE_PDM_BIT_STREAM0

IOMUXC_NAND_ALE_UART3_RX

IOMUXC_NAND_ALE_UART3_TX

IOMUXC_NAND_ALE_GPIO3_IO00

IOMUXC_NAND_ALE_CORESIGHT_TRACE_CLK

IOMUXC_NAND_CE0_B_NAND_CE0_B

IOMUXC_NAND_CE0_B_QSPI_A_SS0_B

IOMUXC_NAND_CE0_B_PDM_BIT_STREAM1

IOMUXC_NAND_CE0_B_UART3_TX

IOMUXC_NAND_CE0_B_UART3_RX

IOMUXC_NAND_CE0_B_GPIO3_IO01

IOMUXC_NAND_CE0_B_CORESIGHT_TRACE_CTL

IOMUXC_NAND_CE1_B_NAND_CE1_B

IOMUXC_NAND_CE1_B_QSPI_A_SS1_B

IOMUXC_NAND_CE1_B_USDHC3_STROBE

IOMUXC_NAND_CE1_B_PDM_BIT_STREAM0

IOMUXC_NAND_CE1_B_I2C4_SCL

IOMUXC_NAND_CE1_B_GPIO3_IO02

IOMUXC_NAND_CE1_B_CORESIGHT_TRACE00

IOMUXC_NAND_CE2_B_NAND_CE2_B

IOMUXC_NAND_CE2_B_QSPI_B_SS0_B

IOMUXC_NAND_CE2_B_USDHC3_DATA5

IOMUXC_NAND_CE2_B_PDM_BIT_STREAM1

IOMUXC_NAND_CE2_B_I2C4_SDA

IOMUXC_NAND_CE2_B_GPIO3_IO03

IOMUXC_NAND_CE2_B_CORESIGHT_TRACE01

IOMUXC_NAND_CE3_B_NAND_CE3_B

IOMUXC_NAND_CE3_B_QSPI_B_SS1_B

IOMUXC_NAND_CE3_B_USDHC3_DATA6

IOMUXC_NAND_CE3_B_PDM_BIT_STREAM2

IOMUXC_NAND_CE3_B_I2C3_SDA

IOMUXC_NAND_CE3_B_GPIO3_IO04

IOMUXC_NAND_CE3_B_CORESIGHT_TRACE02

IOMUXC_NAND_CLE_NAND_CLE

IOMUXC_NAND_CLE_QSPI_B_SCLK

IOMUXC_NAND_CLE_USDHC3_DATA7

IOMUXC_NAND_CLE_GPIO3_IO05

IOMUXC_NAND_CLE_CORESIGHT_TRACE03

IOMUXC_NAND_DATA00_NAND_DATA00

IOMUXC_NAND_DATA00_QSPI_A_DATA0

IOMUXC_NAND_DATA00_PDM_BIT_STREAM2

IOMUXC_NAND_DATA00_UART4_RX

IOMUXC_NAND_DATA00_UART4_TX

IOMUXC_NAND_DATA00_GPIO3_IO06

IOMUXC_NAND_DATA00_CORESIGHT_TRACE04

IOMUXC_NAND_DATA01_NAND_DATA01

IOMUXC_NAND_DATA01_QSPI_A_DATA1

IOMUXC_NAND_DATA01_PDM_BIT_STREAM3

IOMUXC_NAND_DATA01_UART4_TX

IOMUXC_NAND_DATA01_UART4_RX

IOMUXC_NAND_DATA01_GPIO3_IO07

IOMUXC_NAND_DATA01_CORESIGHT_TRACE05

IOMUXC_NAND_DATA02_NAND_DATA02

IOMUXC_NAND_DATA02_QSPI_A_DATA2

IOMUXC_NAND_DATA02_USDHC3_CD_B

IOMUXC_NAND_DATA02_I2C4_SDA

IOMUXC_NAND_DATA02_GPIO3_IO08

IOMUXC_NAND_DATA02_CORESIGHT_TRACE06

IOMUXC_NAND_DATA03_NAND_DATA03

IOMUXC_NAND_DATA03_QSPI_A_DATA3

IOMUXC_NAND_DATA03_USDHC3_WP

IOMUXC_NAND_DATA03_GPIO3_IO09

IOMUXC_NAND_DATA03_CORESIGHT_TRACE07

IOMUXC_NAND_DATA04_NAND_DATA04

IOMUXC_NAND_DATA04_QSPI_B_DATA0

IOMUXC_NAND_DATA04_USDHC3_DATA0

IOMUXC_NAND_DATA04_GPIO3_IO10

IOMUXC_NAND_DATA04_CORESIGHT_TRACE08

IOMUXC_NAND_DATA05_NAND_DATA05

IOMUXC_NAND_DATA05_QSPI_B_DATA1

IOMUXC_NAND_DATA05_USDHC3_DATA1

IOMUXC_NAND_DATA05_GPIO3_IO11

IOMUXC_NAND_DATA05_CORESIGHT_TRACE09

IOMUXC_NAND_DATA06_NAND_DATA06

IOMUXC_NAND_DATA06_QSPI_B_DATA2

IOMUXC_NAND_DATA06_USDHC3_DATA2

IOMUXC_NAND_DATA06_GPIO3_IO12

IOMUXC_NAND_DATA06_CORESIGHT_TRACE10

IOMUXC_NAND_DATA07_NAND_DATA07

IOMUXC_NAND_DATA07_QSPI_B_DATA3

IOMUXC_NAND_DATA07_USDHC3_DATA3

IOMUXC_NAND_DATA07_GPIO3_IO13

IOMUXC_NAND_DATA07_CORESIGHT_TRACE11

IOMUXC_NAND_DQS_NAND_DQS

IOMUXC_NAND_DQS_QSPI_A_DQS

IOMUXC_NAND_DQS_PDM_CLK

IOMUXC_NAND_DQS_I2C3_SCL

IOMUXC_NAND_DQS_GPIO3_IO14

IOMUXC_NAND_DQS_CORESIGHT_TRACE12

IOMUXC_NAND_RE_B_NAND_RE_B

IOMUXC_NAND_RE_B_QSPI_B_DQS

IOMUXC_NAND_RE_B_USDHC3_DATA4

IOMUXC_NAND_RE_B_PDM_BIT_STREAM1

IOMUXC_NAND_RE_B_GPIO3_IO15

IOMUXC_NAND_RE_B_CORESIGHT_TRACE13

IOMUXC_NAND_READY_B_NAND_READY_B

IOMUXC_NAND_READY_B_USDHC3_RESET_B

IOMUXC_NAND_READY_B_PDM_BIT_STREAM3

IOMUXC_NAND_READY_B_I2C3_SCL

IOMUXC_NAND_READY_B_GPIO3_IO16

IOMUXC_NAND_READY_B_CORESIGHT_TRACE14

IOMUXC_NAND_WE_B_NAND_WE_B

IOMUXC_NAND_WE_B_USDHC3_CLK

IOMUXC_NAND_WE_B_I2C3_SDA

IOMUXC_NAND_WE_B_GPIO3_IO17

IOMUXC_NAND_WE_B_CORESIGHT_TRACE15

IOMUXC_NAND_WP_B_NAND_WP_B

IOMUXC_NAND_WP_B_USDHC3_CMD

IOMUXC_NAND_WP_B_I2C4_SDA

IOMUXC_NAND_WP_B_GPIO3_IO18

IOMUXC_NAND_WP_B_CORESIGHT_EVENTO

IOMUXC_SAI5_RXFS_SAI5_RX_SYNC

IOMUXC_SAI5_RXFS_GPIO3_IO19

IOMUXC_SAI5_RXC_SAI5_RX_BCLK

IOMUXC_SAI5_RXC_PDM_CLK

IOMUXC_SAI5_RXC_GPIO3_IO20

IOMUXC_SAI5_RXD0_SAI5_RX_DATA0

IOMUXC_SAI5_RXD0_PDM_BIT_STREAM0

IOMUXC_SAI5_RXD0_GPIO3_IO21

IOMUXC_SAI5_RXD1_SAI5_RX_DATA1

IOMUXC_SAI5_RXD1_SAI5_TX_SYNC

IOMUXC_SAI5_RXD1_PDM_BIT_STREAM1

IOMUXC_SAI5_RXD1_GPIO3_IO22

IOMUXC_SAI5_RXD2_SAI5_RX_DATA2

IOMUXC_SAI5_RXD2_SAI5_TX_BCLK

IOMUXC_SAI5_RXD2_PDM_BIT_STREAM2

IOMUXC_SAI5_RXD2_GPIO3_IO23

IOMUXC_SAI5_RXD3_SAI5_RX_DATA3

IOMUXC_SAI5_RXD3_SAI5_TX_DATA0

IOMUXC_SAI5_RXD3_PDM_BIT_STREAM3

IOMUXC_SAI5_RXD3_GPIO3_IO24

IOMUXC_SAI5_MCLK_SAI5_MCLK

IOMUXC_SAI5_MCLK_GPIO3_IO25

IOMUXC_SAI2_RXFS_SAI2_RX_SYNC

IOMUXC_SAI2_RXFS_SAI5_TX_SYNC

IOMUXC_SAI2_RXFS_SAI5_TX_DATA1

IOMUXC_SAI2_RXFS_SAI2_RX_DATA1

IOMUXC_SAI2_RXFS_UART1_TX

IOMUXC_SAI2_RXFS_UART1_RX

IOMUXC_SAI2_RXFS_GPIO4_IO21

IOMUXC_SAI2_RXFS_PDM_BIT_STREAM2

IOMUXC_SAI2_RXC_SAI2_RX_BCLK

IOMUXC_SAI2_RXC_SAI5_TX_BCLK

IOMUXC_SAI2_RXC_UART1_RX

IOMUXC_SAI2_RXC_UART1_TX

IOMUXC_SAI2_RXC_GPIO4_IO22

IOMUXC_SAI2_RXC_PDM_BIT_STREAM1

IOMUXC_SAI2_RXD0_SAI2_RX_DATA0

IOMUXC_SAI2_RXD0_SAI5_TX_DATA0

IOMUXC_SAI2_RXD0_SAI2_TX_DATA1

IOMUXC_SAI2_RXD0_UART1_RTS_B

IOMUXC_SAI2_RXD0_UART1_CTS_B

IOMUXC_SAI2_RXD0_GPIO4_IO23

IOMUXC_SAI2_RXD0_PDM_BIT_STREAM3

IOMUXC_SAI2_TXFS_SAI2_TX_SYNC

IOMUXC_SAI2_TXFS_SAI5_TX_DATA1

IOMUXC_SAI2_TXFS_SAI2_TX_DATA1

IOMUXC_SAI2_TXFS_UART1_CTS_B

IOMUXC_SAI2_TXFS_UART1_RTS_B

IOMUXC_SAI2_TXFS_GPIO4_IO24

IOMUXC_SAI2_TXFS_PDM_BIT_STREAM2

IOMUXC_SAI2_TXC_SAI2_TX_BCLK

IOMUXC_SAI2_TXC_SAI5_TX_DATA2

IOMUXC_SAI2_TXC_GPIO4_IO25

IOMUXC_SAI2_TXC_PDM_BIT_STREAM1

IOMUXC_SAI2_TXD0_SAI2_TX_DATA0

IOMUXC_SAI2_TXD0_SAI5_TX_DATA3

IOMUXC_SAI2_TXD0_GPIO4_IO26

IOMUXC_SAI2_TXD0_SRC_BOOT_MODE4

IOMUXC_SAI2_MCLK_SAI2_MCLK

IOMUXC_SAI2_MCLK_SAI5_MCLK

IOMUXC_SAI2_MCLK_GPIO4_IO27

IOMUXC_SAI2_MCLK_SAI3_MCLK

IOMUXC_SAI3_RXFS_SAI3_RX_SYNC

IOMUXC_SAI3_RXFS_GPT1_CAPTURE1

IOMUXC_SAI3_RXFS_SAI5_RX_SYNC

IOMUXC_SAI3_RXFS_SAI3_RX_DATA1

IOMUXC_SAI3_RXFS_SPDIF1_IN

IOMUXC_SAI3_RXFS_GPIO4_IO28

IOMUXC_SAI3_RXFS_PDM_BIT_STREAM0

IOMUXC_SAI3_RXC_SAI3_RX_BCLK

IOMUXC_SAI3_RXC_GPT1_CLK

IOMUXC_SAI3_RXC_SAI5_RX_BCLK

IOMUXC_SAI3_RXC_SAI2_RX_DATA1

IOMUXC_SAI3_RXC_UART2_CTS_B

IOMUXC_SAI3_RXC_UART2_RTS_B

IOMUXC_SAI3_RXC_GPIO4_IO29

IOMUXC_SAI3_RXC_PDM_CLK

IOMUXC_SAI3_RXD_SAI3_RX_DATA0

IOMUXC_SAI3_RXD_GPT1_COMPARE1

IOMUXC_SAI3_RXD_SAI5_RX_DATA0

IOMUXC_SAI3_RXD_SAI3_TX_DATA1

IOMUXC_SAI3_RXD_UART2_RTS_B

IOMUXC_SAI3_RXD_UART2_CTS_B

IOMUXC_SAI3_RXD_GPIO4_IO30

IOMUXC_SAI3_RXD_PDM_BIT_STREAM1

IOMUXC_SAI3_TXFS_SAI3_TX_SYNC

IOMUXC_SAI3_TXFS_GPT1_CAPTURE2

IOMUXC_SAI3_TXFS_SAI5_RX_DATA1

IOMUXC_SAI3_TXFS_SAI3_TX_DATA1

IOMUXC_SAI3_TXFS_UART2_RX

IOMUXC_SAI3_TXFS_UART2_TX

IOMUXC_SAI3_TXFS_GPIO4_IO31

IOMUXC_SAI3_TXFS_PDM_BIT_STREAM3

IOMUXC_SAI3_TXC_SAI3_TX_BCLK

IOMUXC_SAI3_TXC_GPT1_COMPARE2

IOMUXC_SAI3_TXC_SAI5_RX_DATA2

IOMUXC_SAI3_TXC_SAI2_TX_DATA1

IOMUXC_SAI3_TXC_UART2_TX

IOMUXC_SAI3_TXC_UART2_RX

IOMUXC_SAI3_TXC_GPIO5_IO00

IOMUXC_SAI3_TXC_PDM_BIT_STREAM2

IOMUXC_SAI3_TXD_SAI3_TX_DATA0

IOMUXC_SAI3_TXD_GPT1_COMPARE3

IOMUXC_SAI3_TXD_SAI5_RX_DATA3

IOMUXC_SAI3_TXD_SPDIF1_EXT_CLK

IOMUXC_SAI3_TXD_GPIO5_IO01

IOMUXC_SAI3_TXD_SRC_BOOT_MODE5

IOMUXC_SAI3_MCLK_SAI3_MCLK

IOMUXC_SAI3_MCLK_PWM4_OUT

IOMUXC_SAI3_MCLK_SAI5_MCLK

IOMUXC_SAI3_MCLK_SPDIF1_OUT

IOMUXC_SAI3_MCLK_GPIO5_IO02

IOMUXC_SAI3_MCLK_SPDIF1_IN

IOMUXC_SPDIF_TX_SPDIF1_OUT

IOMUXC_SPDIF_TX_PWM3_OUT

IOMUXC_SPDIF_TX_GPIO5_IO03

IOMUXC_SPDIF_RX_SPDIF1_IN

IOMUXC_SPDIF_RX_PWM2_OUT

IOMUXC_SPDIF_RX_GPIO5_IO04

IOMUXC_SPDIF_EXT_CLK_SPDIF1_EXT_CLK

IOMUXC_SPDIF_EXT_CLK_PWM1_OUT

IOMUXC_SPDIF_EXT_CLK_GPIO5_IO05

IOMUXC_ECSPI1_SCLK_ECSPI1_SCLK

IOMUXC_ECSPI1_SCLK_UART3_RX

IOMUXC_ECSPI1_SCLK_UART3_TX

IOMUXC_ECSPI1_SCLK_I2C1_SCL

IOMUXC_ECSPI1_SCLK_SAI5_RX_SYNC

IOMUXC_ECSPI1_SCLK_GPIO5_IO06

IOMUXC_ECSPI1_MOSI_ECSPI1_MOSI

IOMUXC_ECSPI1_MOSI_UART3_TX

IOMUXC_ECSPI1_MOSI_UART3_RX

IOMUXC_ECSPI1_MOSI_I2C1_SDA

IOMUXC_ECSPI1_MOSI_SAI5_RX_BCLK

IOMUXC_ECSPI1_MOSI_GPIO5_IO07

IOMUXC_ECSPI1_MISO_ECSPI1_MISO

IOMUXC_ECSPI1_MISO_UART3_CTS_B

IOMUXC_ECSPI1_MISO_UART3_RTS_B

IOMUXC_ECSPI1_MISO_I2C2_SCL

IOMUXC_ECSPI1_MISO_SAI5_RX_DATA0

IOMUXC_ECSPI1_MISO_GPIO5_IO08

IOMUXC_ECSPI1_SS0_ECSPI1_SS0

IOMUXC_ECSPI1_SS0_UART3_RTS_B

IOMUXC_ECSPI1_SS0_UART3_CTS_B

IOMUXC_ECSPI1_SS0_I2C2_SDA

IOMUXC_ECSPI1_SS0_SAI5_RX_DATA1

IOMUXC_ECSPI1_SS0_SAI5_TX_SYNC

IOMUXC_ECSPI1_SS0_GPIO5_IO09

IOMUXC_ECSPI2_SCLK_ECSPI2_SCLK

IOMUXC_ECSPI2_SCLK_UART4_RX

IOMUXC_ECSPI2_SCLK_UART4_TX

IOMUXC_ECSPI2_SCLK_I2C3_SCL

IOMUXC_ECSPI2_SCLK_SAI5_RX_DATA2

IOMUXC_ECSPI2_SCLK_SAI5_TX_BCLK

IOMUXC_ECSPI2_SCLK_GPIO5_IO10

IOMUXC_ECSPI2_MOSI_ECSPI2_MOSI

IOMUXC_ECSPI2_MOSI_UART4_TX

IOMUXC_ECSPI2_MOSI_UART4_RX

IOMUXC_ECSPI2_MOSI_I2C3_SDA

IOMUXC_ECSPI2_MOSI_SAI5_RX_DATA3

IOMUXC_ECSPI2_MOSI_SAI5_TX_DATA0

IOMUXC_ECSPI2_MOSI_GPIO5_IO11

IOMUXC_ECSPI2_MISO_ECSPI2_MISO

IOMUXC_ECSPI2_MISO_UART4_CTS_B

IOMUXC_ECSPI2_MISO_UART4_RTS_B

IOMUXC_ECSPI2_MISO_I2C4_SCL

IOMUXC_ECSPI2_MISO_SAI5_MCLK

IOMUXC_ECSPI2_MISO_GPIO5_IO12

IOMUXC_ECSPI2_SS0_ECSPI2_SS0

IOMUXC_ECSPI2_SS0_UART4_RTS_B

IOMUXC_ECSPI2_SS0_UART4_CTS_B

IOMUXC_ECSPI2_SS0_I2C4_SDA

IOMUXC_ECSPI2_SS0_GPIO5_IO13

IOMUXC_I2C1_SCL_I2C1_SCL

IOMUXC_I2C1_SCL_ENET1_MDC

IOMUXC_I2C1_SCL_ECSPI1_SCLK

IOMUXC_I2C1_SCL_GPIO5_IO14

IOMUXC_I2C1_SDA_I2C1_SDA

IOMUXC_I2C1_SDA_ENET1_MDIO

IOMUXC_I2C1_SDA_ECSPI1_MOSI

IOMUXC_I2C1_SDA_GPIO5_IO15

IOMUXC_I2C2_SCL_I2C2_SCL

IOMUXC_I2C2_SCL_ENET1_1588_EVENT1_IN

IOMUXC_I2C2_SCL_USDHC3_CD_B

IOMUXC_I2C2_SCL_ECSPI1_MISO

IOMUXC_I2C2_SCL_GPIO5_IO16

IOMUXC_I2C2_SDA_I2C2_SDA

IOMUXC_I2C2_SDA_ENET1_1588_EVENT1_OUT

IOMUXC_I2C2_SDA_USDHC3_WP

IOMUXC_I2C2_SDA_ECSPI1_SS0

IOMUXC_I2C2_SDA_GPIO5_IO17

IOMUXC_I2C3_SCL_I2C3_SCL

IOMUXC_I2C3_SCL_PWM4_OUT

IOMUXC_I2C3_SCL_GPT2_CLK

IOMUXC_I2C3_SCL_ECSPI2_SCLK

IOMUXC_I2C3_SCL_GPIO5_IO18

IOMUXC_I2C3_SDA_I2C3_SDA

IOMUXC_I2C3_SDA_PWM3_OUT

IOMUXC_I2C3_SDA_GPT3_CLK

IOMUXC_I2C3_SDA_ECSPI2_MOSI

IOMUXC_I2C3_SDA_GPIO5_IO19

IOMUXC_I2C4_SCL_I2C4_SCL

IOMUXC_I2C4_SCL_PWM2_OUT

IOMUXC_I2C4_SCL_ECSPI2_MISO

IOMUXC_I2C4_SCL_GPIO5_IO20

IOMUXC_I2C4_SDA_I2C4_SDA

IOMUXC_I2C4_SDA_PWM1_OUT

IOMUXC_I2C4_SDA_ECSPI2_SS0

IOMUXC_I2C4_SDA_GPIO5_IO21

IOMUXC_UART1_RXD_UART1_RX

IOMUXC_UART1_RXD_UART1_TX

IOMUXC_UART1_RXD_ECSPI3_SCLK

IOMUXC_UART1_RXD_GPIO5_IO22

IOMUXC_UART1_TXD_UART1_TX

IOMUXC_UART1_TXD_UART1_RX

IOMUXC_UART1_TXD_ECSPI3_MOSI

IOMUXC_UART1_TXD_GPIO5_IO23

IOMUXC_UART2_RXD_UART2_RX

IOMUXC_UART2_RXD_UART2_TX

IOMUXC_UART2_RXD_ECSPI3_MISO

IOMUXC_UART2_RXD_GPT1_COMPARE3

IOMUXC_UART2_RXD_GPIO5_IO24

IOMUXC_UART2_TXD_UART2_TX

IOMUXC_UART2_TXD_UART2_RX

IOMUXC_UART2_TXD_ECSPI3_SS0

IOMUXC_UART2_TXD_GPT1_COMPARE2

IOMUXC_UART2_TXD_GPIO5_IO25

IOMUXC_UART3_RXD_UART3_RX

IOMUXC_UART3_RXD_UART3_TX

IOMUXC_UART3_RXD_UART1_CTS_B

IOMUXC_UART3_RXD_UART1_RTS_B

IOMUXC_UART3_RXD_USDHC3_RESET_B

IOMUXC_UART3_RXD_GPT1_CAPTURE2

IOMUXC_UART3_RXD_GPIO5_IO26

IOMUXC_UART3_TXD_UART3_TX

IOMUXC_UART3_TXD_UART3_RX

IOMUXC_UART3_TXD_UART1_RTS_B

IOMUXC_UART3_TXD_UART1_CTS_B

IOMUXC_UART3_TXD_USDHC3_VSELECT

IOMUXC_UART3_TXD_GPT1_CLK

IOMUXC_UART3_TXD_GPIO5_IO27

IOMUXC_UART4_RXD_UART4_RX

IOMUXC_UART4_RXD_UART4_TX

IOMUXC_UART4_RXD_UART2_CTS_B

IOMUXC_UART4_RXD_UART2_RTS_B

IOMUXC_UART4_RXD_GPT1_COMPARE1

IOMUXC_UART4_RXD_GPIO5_IO28

IOMUXC_UART4_TXD_UART4_TX

IOMUXC_UART4_TXD_UART4_RX

IOMUXC_UART4_TXD_UART2_RTS_B

IOMUXC_UART4_TXD_UART2_CTS_B

IOMUXC_UART4_TXD_GPT1_CAPTURE1

IOMUXC_UART4_TXD_GPIO5_IO29

FSL_COMPONENT_ID

2.16 ISI: Image Sensing Interface

void ISI_Init(ISI_Type *base)

Initializes the ISI peripheral.

This function ungates the ISI clock, it should be called before any other ISI functions.

Parameters

- base – ISI peripheral base address.

void ISI_Deinit(ISI_Type *base)

Deinitializes the ISI peripheral.

This function gates the ISI clock.

Parameters

- base – ISI peripheral base address.

void ISI_Reset(ISI_Type *base)

Reset the ISI peripheral.

This function resets the ISI channel processing pipeline similar to a hardware reset. The channel will need to be reconfigured after reset before it can be used.

Parameters

- base – ISI peripheral base address.

static inline uint32_t ISI_EnableInterrupts(ISI_Type *base, uint32_t mask)

Enables ISI interrupts.

Parameters

- base – ISI peripheral base address
- mask – Interrupt source, OR'ed value of _isi_interrupt.

Returns

OR'ed value of the enabled interrupts before calling this function.

static inline uint32_t ISI_DisableInterrupts(ISI_Type *base, uint32_t mask)

Disables ISI interrupts.

Parameters

- base – ISI peripheral base address
- mask – Interrupt source, OR'ed value of _isi_interrupt.

Returns

OR'ed value of the enabled interrupts before calling this function.

static inline uint32_t ISI_GetInterruptStatus(ISI_Type *base)

Get the ISI interrupt pending flags.

All interrupt pending flags are returned, upper layer could compare with the OR'ed value of _isi_interrupt. For example, to check whether memory read completed, use like this:

```
uint32_t mask = ISI_GetInterruptStatus(ISI);
if (mask & kISI_MemReadCompletedInterrupt)
{
    memory read completed
}
```

Parameters

- base – ISI peripheral base address

Returns

The OR'ed value of the pending interrupt flags. of _isi_interrupt.

static inline void ISI_ClearInterruptStatus(ISI_Type *base, uint32_t mask)

Clear ISI interrupt pending flags.

This function could clear one or more flags at one time, the flags to clear are passed in as an OR'ed value of _isi_interrupt. For example, to clear both line received interrupt flag and frame received flag, use like this:

```
ISI_ClearInterruptStatus(ISI, kISI_LineReceivedInterrupt | kISI_FrameReceivedInterrupt);
```

Parameters

- base – ISI peripheral base address
- mask – The flags to clear, it is OR'ed value of _isi_interrupt.

void ISI_SetScalerConfig(ISI_Type *base, uint16_t inputWidth, uint16_t inputHeight, uint16_t outputWidth, uint16_t outputHeight)

Set the ISI channel scaler configurations.

This function sets the scaling configurations. If the ISI channel is bypassed, then the scaling feature could not be used.

ISI only supports down scaling but not up scaling.

Note: Total bytes in one line after down scaling must be more than 256 bytes.

Parameters

- base – ISI peripheral base address
- inputWidth – Input image width.
- inputHeight – Input image height.
- outputWidth – Output image width.
- outputHeight – Output image height.

void ISI_SetColorSpaceConversionConfig(ISI_Type *base, const *isi_csc_config_t* *config)

Set the ISI color space conversion configurations.

This function sets the color space conversion configurations. After setting the configuration, use the function ISI_EnableColorSpaceConversion to enable this feature. If the ISI channel is bypassed, then the color space conversion feature could not be used.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_ColorSpaceConversionGetDefaultConfig(*isi_csc_config_t* *config)

Get the ISI color space conversion default configurations.

The default value is:

```
config->mode = kISI_CscYUV2RGB;
config->A1 = 0.0;
config->A2 = 0.0;
config->A3 = 0.0;
config->B1 = 0.0;
config->B2 = 0.0;
config->B3 = 0.0;
config->C1 = 0.0;
config->C2 = 0.0;
config->C3 = 0.0;
config->D1 = 0;
config->D2 = 0;
config->D3 = 0;
```

Parameters

- config – Pointer to the configuration structure.

static inline void ISI_EnableColorSpaceConversion(ISI_Type *base, bool enable)

Enable or disable the ISI color space conversion.

If the ISI channel is bypassed, then the color space conversion feature could not be used even enable using this function.

Note: The CSC is enabled by default. Disable it if it is not required.

Parameters

- base – ISI peripheral base address
- enable – True to enable, false to disable.

void ISI_SetCropConfig(ISI_Type *base, const *isi_crop_config_t* *config)

Set the ISI cropping configurations.

This function sets the cropping configurations. After setting the configuration, use the function ISI_EnableCrop to enable the feature. Cropping still works when the ISI channel is bypassed.

Note: The upper left corner and lower right corner should be configured base on the image resolution output from the scaler.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_CropGetDefaultConfig(*isi_crop_config_t* *config)

Get the ISI cropping default configurations.

The default value is:

```
config->upperLeftX = 0U;
config->upperLeftY = 0U;
config->lowerRightX = 0U;
config->lowerRightY = 0U;
```

Parameters

- config – Pointer to the configuration structure.

static inline void ISI_EnableCrop(ISI_Type *base, bool enable)

Enable or disable the ISI cropping.

If the ISI channel is bypassed, the cropping still works.

Parameters

- base – ISI peripheral base address
- enable – True to enable, false to disable.

static inline void ISI_SetGlobalAlpha(ISI_Type *base, uint8_t alpha)

Set the global alpha value.

Parameters

- base – ISI peripheral base address
- alpha – The global alpha value.

static inline void ISI_EnableGlobalAlpha(ISI_Type *base, bool enable)

Enable the global alpha insertion.

Alpha still works when channel bypassed.

Parameters

- base – ISI peripheral base address
- enable – True to enable, false to disable.

void ISI_SetRegionAlphaConfig(ISI_Type *base, uint8_t index, const *isi_region_alpha_config_t* *config)

Set the alpha value for region of interest.

Set the alpha insertion configuration for specific region of interest. The function ISI_EnableRegionAlpha could be used to enable the alpha insertion. Alpha insertion still works when channel bypassed.

Note: The upper left corner and lower right corner should be configured base on the image resolution output from the scaler.

Parameters

- base – ISI peripheral base address
- index – Index of the region of interest, Could be 0, 1, 2, and 3.
- config – Pointer to the configuration structure.

void ISI_RegionAlphaGetDefaultConfig(*isi_region_alpha_config_t* *config)

Get the regional alpha insertion default configurations.

The default configuration is:

```
config->upperLeftX = 0U;
config->upperLeftY = 0U;
config->lowerRightX = 0U;
config->lowerRightY = 0U;
config->alpha = 0U;
```

Parameters

- config – Pointer to the configuration structure.

void ISI_EnableRegionAlpha(ISI_Type *base, uint8_t index, bool enable)

Enable or disable the alpha value insertion for region of interest.

Alpha insertion still works when channel bypassed.

Parameters

- base – ISI peripheral base address
- index – Index of the region of interest, Could be 0, 1, 2, and 3.
- enable – True to enable, false to disable.

void ISI_SetInputMemConfig(ISI_Type *base, const *isi_input_mem_config_t* *config)

Set the input memory configuration.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_InputMemGetDefaultConfig(*isi_input_mem_config_t* *config)

Get the input memory default configurations.

The default configuration is:

```
config->adddr = 0U;
config->linePitchBytes = 0U;
config->framePitchBytes = 0U;
config->format = kISI_InputMemBGR8P;
```

Parameters

- config – Pointer to the configuration structure.

static inline void ISI_SetInputMemAddr(ISI_Type *base, uint32_t addr)

Set the input memory address.

This function only sets the input memory address, it is used for fast run-time setting.

Parameters

- base – ISI peripheral base address
- addr – Input memory address.

void ISI_TriggerInputMemRead(ISI_Type *base)

Trigger the ISI pipeline to read the input memory.

Parameters

- base – ISI peripheral base address

static inline void ISI_SetFlipMode(ISI_Type *base, *isi_flip_mode_t* mode)

Set the ISI channel flipping mode.

Parameters

- base – ISI peripheral base address
- mode – Flipping mode.

void ISI_SetOutputBufferAddr(ISI_Type *base, uint8_t index, uint32_t addrY, uint32_t addrU, uint32_t addrV)

Set the ISI output buffer address.

This function sets the output buffer address and trigger the ISI to shadow the address, it is used for fast run-time setting.

Parameters

- base – ISI peripheral base address
- index – Index of output buffer, could be 0 and 1.
- addrY – RGB or Luma (Y) output buffer address.
- addrU – Chroma (U/Cb/UV/CbCr) output buffer address.
- addrV – Chroma (V/Cr) output buffer address.

static inline void ISI_Start(ISI_Type *base)

Start the ISI channel.

Start the ISI channel to work, this function should be called after all channel configuration finished.

Parameters

- base – ISI peripheral base address

static inline void ISI_Stop(ISI_Type *base)

Stop the ISI channel.

Parameters

- base – ISI peripheral base address

FSL_ISI_DRIVER_VERSION
ISI driver version.

enum _isi_interrupt
ISI interrupts.

Values:

enumerator kISI_MemReadCompletedInterrupt
Input memory read completed.

enumerator kISI_LineReceivedInterrupt
Line received.

enumerator kISI_FrameReceivedInterrupt
Frame received.

enumerator kISI_AxiWriteErrorVInterrupt
AXI Bus write error when storing V data to memory.

enumerator kISI_AxiWriteErrorUInterrupt
AXI Bus write error when storing U data to memory.

enumerator kISI_AxiWriteErrorYInterrupt
AXI Bus write error when storing Y data to memory.

enumerator kISI_AxiReadErrorInterrupt
AXI Bus error when reading the input memory.

enum _isi_output_format
ISI output image format.

Values:

enumerator kISI_OutputRGBA8888
RGBA8888.

enumerator kISI_OutputABGR8888
ABGR8888.

enumerator kISI_OutputARGB8888
ARGB8888.

enumerator kISI_OutputRGBX8888
RGBX8888 unpacked and MSB aligned in 32-bit.

enumerator kISI_OutputXBGR8888
XBGR8888 unpacked and LSB aligned in 32-bit.

enumerator kISI_OutputXRGB8888
XRGB8888 unpacked and LSB aligned in 32-bit.

enumerator kISI_OutputRGB888
RGB888 packed into 32-bit.

enumerator kISI_OutputBGR888
BGR888 packed into 32-bit.

enumerator kISI_OutputA2BGR10
BGR format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_OutputA2RGB10

RGB format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_OutputRGB565

RGB565 packed into 32-bit.

enumerator kISI_OutputRaw8

8-bit raw data packed into 32-bit.

enumerator kISI_OutputRaw10

10-bit raw data packed into 16-bit with 6 LSBs wasted.

enumerator kISI_OutputRaw10P

10-bit raw data packed into 32-bit.

enumerator kISI_OutputRaw12P

16-bit raw data packed into 16-bit with 4 LSBs wasted.

enumerator kISI_OutputRaw16P

16-bit raw data packed into 32-bit.

enumerator kISI_OutputYUV444_1P8P

8-bits per color component; 1-plane, YUV interleaved packed bytes.

enumerator kISI_OutputYUV444_2P8P

8-bits per color component; 2-plane, UV interleaved packed bytes.

enumerator kISI_OutputYUV444_3P8P

8-bits per color component; 3-plane, non-interleaved packed bytes.

enumerator kISI_OutputYUV444_1P8

8-bits per color component; 1-plane YUV interleaved unpacked bytes (8 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_1P10

10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_2P10

10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_3P10

10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_1P10P

10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_2P10P

10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_3P10P

10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_1P12

12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_2P12

12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_3P12

12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_1P8P

8-bits per color component; 1-plane, YUV interleaved packed bytes.

enumerator kISI_OutputYUV422_2P8P

8-bits per color component; 2-plane, UV interleaved packed bytes.

enumerator kISI_OutputYUV422_3P8P

8-bits per color component; 3-plane, non-interleaved packed bytes.

enumerator kISI_OutputYUV422_1P10

10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_2P10

10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_3P10

10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_1P10P

10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV422_2P10P

10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV422_3P10P

10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV422_1P12

12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_2P12

12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_3P12

12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_2P8P

8-bits per color component; 2-plane, UV interleaved packed bytes.

enumerator kISI_OutputYUV420_3P8P

8-bits per color component; 3-plane, non-interleaved packed bytes.

enumerator kISI_OutputYUV420_2P10

10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_3P10
10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_2P10P
10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV420_3P10P
10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV420_2P12
12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_3P12
12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enum _isi_chain_mode
ISI line buffer chain mode.

Values:

enumerator kISI_ChainDisable
No line buffers chained, for 2048 or less horizontal resolution.

enumerator kISI_ChainTwo
Line buffers of channel n and n+1 chained, for 4096 horizontal resolution.

enum _isi_deint_mode
ISI de-interlacing mode.

Values:

enumerator kISI_DeintDisable
No de-interlacing.

enumerator kISI_DeintWeaveOddOnTop
Weave de-interlacing (Odd, Even) method used.

enumerator kISI_DeintWeaveEvenOnTop
Weave de-interlacing (Even, Odd) method used.

enumerator kISI_DeintBlendingOddFirst
Blending or linear interpolation (Odd + Even).

enumerator kISI_DeintBlendingEvenFirst
Blending or linear interpolation (Even + Odd).

enumerator kISI_DeintDoublingOdd
Doubling odd frame and discard even frame.

enumerator kISI_DeintDoublingEven
Doubling even frame and discard odd frame.

enum _isi_threshold
ISI overflow panic alert threshold.

Values:

enumerator kISI_ThresholdDisable
No panic alert will be asserted.

enumerator kISI_Threshold25Percent
Panic will assert when the buffers are 25% full.

enumerator kISI_Threshold50Percent
Panic will assert when the buffers are 50% full.

enumerator kISI_Threshold75Percent
Panic will assert when the buffers are 75% full.

enum _isi_csc_mode
ISI color space conversion mode.

Values:

enumerator kISI_CscYUV2RGB
Convert YUV to RGB.

enumerator kISI_CscYCbCr2RGB
Convert YCbCr to RGB.

enumerator kISI_CscRGB2YUV
Convert RGB to YUV.

enumerator kISI_CscRGB2YCbCr
Convert RGB to YCbCr.

enum _isi_flip_mode
ISI flipping mode.

Values:

enumerator kISI_FlipDisable
Flip disabled.

enumerator kISI_FlipHorizontal
Horizontal flip.

enumerator kISI_FlipVertical
Vertical flip.

enumerator kISI_FlipBoth
Flip both direction.

enum _isi_input_mem_format
ISI image format of the input memory.

Values:

enumerator kISI_InputMemBGR888
BGR format with 8-bits per color component, packed into 32-bit, 24 bits per pixel.

enumerator kISI_InputMemRGB888
RGB format with 8-bits per color component, packed into 32-bit, 24 bits per pixel.

enumerator kISI_InputMemXRGB8888
RGB format with 8-bits per color component, unpacked and LSB aligned in 32-bit, 32 bits per pixel.

enumerator kISI_InputMemRGBX8888
RGB format with 8-bits per color component, unpacked and MSB alinged in 32-bit, 32 bits per pixel.

enumerator kISI_InputMemXBGR8888

BGR format with 8-bits per color component, unpacked and LSB aligned in 32-bit, 32 bits per pixel.

enumerator kISI_InputMemRGB565

RGB format with 5-bits of R, B; 6-bits of G (packed into 32-bit)

enumerator kISI_InputMemA2BGR10

BGR format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_InputMemA2RGB10

RGB format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_InputMemYUV444_1P8P

8-bits per color component; 1-plane, YUV interleaved packed bytes.

enumerator kISI_InputMemYUV444_1P10

10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_InputMemYUV444_1P10P

10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit WORD).

enumerator kISI_InputMemYUV444_1P12

12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_InputMemYUV444_1P8

8-bits per color component; 1-plane YUV interleaved unpacked bytes (8 MSBs waste bits in 32-bit DWORD).

enumerator kISI_InputMemYUV422_1P8P

8-bits per color component; 1-plane YUV interleaved packed bytes.

enumerator kISI_InputMemYUV422_1P10

10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_InputMemYUV422_1P12

12-bits per color component; 1-plane, YUV interleaved packed bytes (4 MSBs waste bits in 16-bit WORD).

enum _isi_roi_index

ISI roi index number.

Values:

enumerator ISI_ROI_INDEX_0

ISI ROI index 0

enumerator ISI_ROI_INDEX_1

ISI ROI index 1

enumerator ISI_ROI_INDEX_2

ISI ROI index 2

enumerator ISI_ROI_INDEX_3

ISI ROI index 3

typedef enum *_isi_output_format* isi_output_format_t

ISI output image format.

typedef enum *_isi_chain_mode* isi_chain_mode_t
ISI line buffer chain mode.

typedef enum *_isi_deint_mode* isi_deint_mode_t
ISI de-interlacing mode.

typedef enum *_isi_threshold* isi_threshold_t
ISI overflow panic alert threshold.

typedef struct *_isi_config* isi_config_t
ISI basic configuration.

typedef enum *_isi_csc_mode* isi_csc_mode_t
ISI color space conversion mode.

typedef struct *_isi_csc_config* isi_csc_config_t
ISI color space conversion configurations.

(a) RGB to YUV (or YCbCr) conversion

- Y = (A1 x R) + (A2 x G) + (A3 x B) + D1
- U = (B1 x R) + (B2 x G) + (B3 x B) + D2
- V = (C1 x R) + (C2 x G) + (C3 x B) + D3

(b) YUV (or YCbCr) to RGB conversion

- R = (A1 x (Y + D1)) + (A2 x (U + D2)) + (A3 x (V + D3))
- G = (B1 x (Y + D1)) + (B2 x (U + D2)) + (B3 x (V + D3))
- B = (C1 x (Y + D1)) + (C2 x (U + D2)) + (C3 x (V + D3))

Overflow for the three channels are saturated at 0x255 and underflow is saturated at 0x00.

typedef enum *_isi_flip_mode* isi_flip_mode_t
ISI flipping mode.

typedef struct *_isi_crop_config* isi_crop_config_t
ISI cropping configurations.

typedef struct *_isi_regoin_alpha_config* isi_region_alpha_config_t
ISI regional region alpha configurations.

typedef enum *_isi_input_mem_format* isi_input_mem_format_t
ISI image format of the input memory.

typedef struct *_isi_input_mem_config* isi_input_mem_config_t
ISI input memory configurations.

typedef enum *_isi_roi_index* isi_roi_index_t
ISI roi index number.

void ISI_SetConfig(ISI_Type *base, const *isi_config_t* *config)
Set the ISI channel basic configurations.

This function sets the basic configurations, generally the channel could be started to work after this function. To enable other features such as croping, flipping, please call the functions accordingly.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_GetDefaultConfig(*isi_config_t* *config)

Get the ISI channel default basic configurations.

The default value is:

```
config->isChannelBypassed = false;
config->isSourceMemory = false;
config->isYCbCr = false;
config->chainMode = kISI_ChainDisable;
config->deintMode = kISI_DeintDisable;
config->blankPixel = 0xFFU;
config->sourcePort = 0U;
config->mipiChannel = 0U;
config->inputHeight = 1080U;
config->inputWidth = 1920U;
config->outputFormat = kISI_OutputRGBA8888;
config->outputLinePitchBytes = 0U;
config->thresholdY = kISI_ThresholdDisable;
config->thresholdU = kISI_ThresholdDisable;
config->thresholdV = kISI_ThresholdDisable;
```

Parameters

- config – Pointer to the configuration structure.

struct _isi_config

#include <fsl_isi.h> ISI basic configuration.

Public Members

bool isChannelBypassed

Bypass the channel, if bypassed, the scaling and color space conversion could not work.

bool isSourceMemory

Whether the input source is memory or not.

bool isYCbCr

Whether the input source is YCbCr mode or not.

isi_chain_mode_t chainMode

The line buffer chain mode.

isi_deint_mode_t deintMode

The de-interlacing mode.

uint8_t blankPixel

The pixel to insert into image when overflow occors.

uint8_t sourcePort

Input source port selection.

uint8_t mipiChannel

MIPI virtual channel, ignored if input source is not MIPI CSI.

uint16_t inputHeight

Input image height(lines).

uint16_t inputWidth

Input image width(pixels).

isi_output_format_t outputFormat

Output image format.

isi_threshold_t thresholdY
: Panic alert threshold for RGB or Luma (Y) buffer.

isi_threshold_t thresholdU
: Panic alert threshold for Chroma (U/Cb/UV/CbCr) buffer.

isi_threshold_t thresholdV
: Panic alert threshold for Chroma (V/Cr) buffer.

struct __isi_csc_config

#include <fsl_isi.h> ISI color space conversion configurations.

(a) RGB to YUV (or YCbCr) conversion

- Y = (A1 x R) + (A2 x G) + (A3 x B) + D1
- U = (B1 x R) + (B2 x G) + (B3 x B) + D2
- V = (C1 x R) + (C2 x G) + (C3 x B) + D3

(b) YUV (or YCbCr) to RGB conversion

- R = (A1 x (Y + D1)) + (A2 x (U + D2)) + (A3 x (V + D3))
- G = (B1 x (Y + D1)) + (B2 x (U + D2)) + (B3 x (V + D3))
- B = (C1 x (Y + D1)) + (C2 x (U + D2)) + (C3 x (V + D3))

Overflow for the three channels are saturated at 0x255 and underflow is saturated at 0x00.

Public Members

isi_csc_mode_t mode
: Convertion mode.

float A1
: Must be in the range of [-3.99609375, 3.99609375].

float A2
: Must be in the range of [-3.99609375, 3.99609375].

float A3
: Must be in the range of [-3.99609375, 3.99609375].

float B1
: Must be in the range of [-3.99609375, 3.99609375].

float B2
: Must be in the range of [-3.99609375, 3.99609375].

float B3
: Must be in the range of [-3.99609375, 3.99609375].

float C1
: Must be in the range of [-3.99609375, 3.99609375].

float C2
: Must be in the range of [-3.99609375, 3.99609375].

float C3
: Must be in the range of [-3.99609375, 3.99609375].

int32_t D1
: Must be in the range of [-256, 255].

int32_t D2
: Must be in the range of [-256, 255].

int32_t D3
: Must be in the range of [-256, 255].

struct __isi_crop_config
: *#include <fsl_isi.h>* ISI cropping configurations.

Public Members

uint16_t upperLeftX
: X of upper left corner.

uint16_t upperLeftY
: Y of upper left corner.

uint16_t lowerRightX
: X of lower right corner.

uint16_t lowerRightY
: Y of lower right corner.

struct __isi_regoin_alpha_config
: *#include <fsl_isi.h>* ISI regional region alpha configurations.

Public Members

uint16_t upperLeftX
: X of upper left corner.

uint16_t upperLeftY
: Y of upper left corner.

uint16_t lowerRightX
: X of lower right corner.

uint16_t lowerRightY
: Y of lower right corner.

uint8_t alpha
: Alpha value.

struct __isi_input_mem_config
: *#include <fsl_isi.h>* ISI input memory configurations.

Public Members

uint32_t adddr
: Address of the input memory.

uint16_t linePitchBytes
: Line phtch in bytes.

uint16_t framePitchBytes
: Frame phtch in bytes.

isi_input_mem_format_t format
: Image format of the input memory.

2.17 Common Driver

FSL_COMMON_DRIVER_VERSION
: common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE
: No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART
: Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART
: Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
: Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
: Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
: Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART
: Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART
: Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART
: Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO
: Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI
: Debug console based on QSCI.

MIN(a, b)
: Computes the minimum of *a* and *b*.

MAX(a, b)
: Computes the maximum of *a* and *b*.

UINT16_MAX
: Max value of uint16_t type.

UINT32_MAX
: Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(addr, val)
: Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)
: Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)
: Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)
: Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value (rounded up)

SDK_SIZEALIGN_UP(var, alignbytes)

Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK_SIZEALIGN

SDK_SIZEALIGN_DOWN(var, alignbytes)

Macro to change a value to a given size aligned value (rounded down)

SDK_IS_ALIGNED(var, alignbytes)

Macro to check if a value is aligned to a given size

AT_NONCACHEABLE_SECTION(var)

Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)

Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_CACHE_LINE_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

AT_QUICKACCESS_SECTION_CODE(func)

Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)

Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)

Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

MCUX_RAMFUNC

Function attribute to place function in RAM. For example, to place function my_func in ram, use like:

```
MCUX_RAMFUNC my_func
```

RAMFUNCTION_SECTION_CODE(func)

Place function in ram.

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI

Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART

Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C

Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C

Group number for LPI2C status codes.

enumerator kStatusGroup_UART

Group number for UART status codes.

enumerator kStatusGroup_I2C

Group number for UART status codes.

enumerator kStatusGroup_LPSCI
Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART
Group number for LPUART status codes.

enumerator kStatusGroup_SPI
Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPI
Group number for ECSPI status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator kStatusGroup_SDIOSLV
Group number for SDIOSLV status codes.

enumerator kStatusGroup_MECC
Group number for MECC status codes.

enumerator kStatusGroup_ENET_QOS
Group number for ENET_QOS status codes.

enumerator kStatusGroup_LOG
Group number for LOG status codes.

enumerator kStatusGroup_I3CBUS
Group number for I3CBUS status codes.

enumerator kStatusGroup_QSCI
Group number for QSCI status codes.

enumerator kStatusGroup_ELEMU
Group number for ELEMU status codes.

enumerator kStatusGroup_QUEUEDSPI
Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.

enumerator kStatusGroup_CLIF
Group number for CLIF status codes.

enumerator kStatusGroup_BMA
Group number for BMA status codes.

enumerator kStatusGroup_NETC
Group number for NETC status codes.

enumerator kStatusGroup_ELE
Group number for ELE status codes.

enumerator kStatusGroup_GLIKEY
Group number for GLIKEY status codes.

enumerator kStatusGroup_AON_POWER
Group number for AON_POWER status codes.

enumerator kStatusGroup_AON_COMMON
Group number for AON_COMMON status codes.

enumerator kStatusGroup_ENDAT3
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

enumerator kStatusGroup_LPC_QSPI
Group number for LPC QSPI status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
Generic status for Success.

enumerator kStatus_Fail
Generic status for Fail.

enumerator kStatus_ReadOnly
Generic status for read only failure.

enumerator kStatus_OutOfRange
Generic status for out of range access.

enumerator kStatus_InvalidArgument
Generic status for invalid argument check.

enumerator kStatus_Timeout
Generic status for timeout.

enumerator kStatus_NoTransferInProgress
Generic status for no transfer in progress.

enumerator kStatus_Busy
Generic status for module is busy.

enumerator kStatus_NoData
Generic status for no data is found for the operation.

typedef int32_t status_t
Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values
The – allocated memory.

void SDK_Free(void *ptr)
Free memory.

Parameters

- ptr – The memory to be release.

void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)
Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- delayTime_us – Delay time in unit of microsecond.
- coreClock_Hz – Core clock frequency with Hz.

static inline *status_t* EnableIRQ(IRQn_Type interrupt)

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt enabled successfully
- kStatus_Fail – Failed to enable the interrupt

static inline *status_t* DisableIRQ(IRQn_Type interrupt)

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt disabled successfully
- kStatus_Fail – Failed to disable the interrupt

static inline *status_t* EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the

LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

Returns
Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

static inline bool _SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t newValue)

static inline uint32_t _SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

```
| Unused    || Major Version || Minor Version ||  Bug Fix    |
31       25  24          17  16          9  8           0
```

ARRAY_SIZE(x)

Computes the number of elements in an array.

UINT64_H(X)

Macro to get upper 32 bits of a 64-bit value

UINT64_L(X)

Macro to get lower 32 bits of a 64-bit value

SUPPRESS_FALL_THROUGH_WARNING()

For switch case code block, if case section ends without "break;" statement, there wil be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, "SUPPRESS_FALL_THROUGH_WARNING();" need to be added at the end of each case section which misses "break;"statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK_HAS_DWT_CYCCNT

The chip supports DWT CYCCNT or not.

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.18 LCDIF: LCD interface

status_t LCDIF_Init(LCDIF_Type *base)

Initialize the LCDIF.

This function initializes the LCDIF to work.

Parameters

- base – LCDIF peripheral base address.

Return values

kStatus_Success – Initialize successfully.

void LCDIF_Deinit(LCDIF_Type *base)

De-initialize the LCDIF.

This function disables the LCDIF peripheral clock.

Parameters

- base – LCDIF peripheral base address.

void LCDIF_DpiModeGetDefaultConfig(*lcdif_dpi_config_t* *config)

Get the default configuration for to initialize the LCDIF.

The default configuration value is:

```
config->panelWidth = 0;
config->panelHeight = 0;
config->hsw = 0;
config->hfp = 0;
config->hbp = 0;
config->vsw = 0;
config->vfp = 0;
config->vbp = 0;
config->polarityFlags = kLCDIF_VsyncActiveLow | kLCDIF_HsyncActiveLow | kLCDIF_
↪DataEnableActiveHigh |
kLCDIF_DriveDataOnFallingClkEdge; config->format = kLCDIF_Output24Bit;
```

Parameters

- config – Pointer to the LCDIF configuration.

status_t LCDIF_DpiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dpi_config_t* *config)

Initialize the LCDIF to work in DPI mode.

This function configures the LCDIF DPI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

status_t LCDIF_DbiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dbi_config_t* *config)

Initialize the LCDIF to work in DBI mode.

This function configures the LCDIF DBI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

void LCDIF_DbiModeGetDefaultConfig(*lcdif_dbi_config_t* *config)

Get the default configuration to initialize the LCDIF DBI mode.

The default configuration value is:

```
config->swizzle         = kLCDIF_DbiOutSwizzleRGB;
config->format          = kLCDIF_DbiOutD8RGB332;
config->acTimeUnit      = 0;
config->type            = kLCDIF_DbiTypeA_ClockedE;
config->reversePolarity = false;
config->writeWRPeriod   = 3U;
config->writeWRAssert   = 0U;
config->writeCSAssert   = 0U;
config->writeWRDeassert = 0U;
config->writeCSDeassert = 0U;
config->typeCTas        = 1U;
config->typeCSCLTwrl    = 1U;
config->typeCSCLTwrh    = 1U;
```

Parameters

- config – Pointer to the LCDIF DBI configuration.

static inline void LCDIF_DbiReset(LCDIF_Type *base, uint8_t displayIndex)

Reset the DBI module.

Parameters

- displayIndex – Display index.
- base – LCDIF peripheral base address.

void LCDIF_DbiSelectArea(LCDIF_Type *base, uint8_t displayIndex, uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY, bool isTiled)

Select the update area in DBI mode.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startX – X coordinate for start pixel.
- startY – Y coordinate for start pixel.
- endX – X coordinate for end pixel.
- endY – Y coordinate for end pixel.
- isTiled – true if the pixel data is tiled.

static inline void LCDIF_DbiSendCommand(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd)

Send command to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.

void LCDIF_DbiSendData(LCDIF_Type *base, uint8_t displayIndex, const uint8_t *data, uint32_t dataLen_Byte)

brief Send data to DBI port.

Can be used to send light weight data to panel. To send pixel data in frame buffer, use LCDIF_DbiWriteMem.

param base LCDIF peripheral base address. param displayIndex Display index. param data pointer to data buffer. param dataLen_Byte data buffer length in byte.

void LCDIF_DbiSendCommandAndData(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd, const uint8_t *data, uint32_t dataLen_Byte)

Send command followed by data to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.
- data – pointer to data buffer.
- dataLen_Byte – data buffer length in byte.

static inline void LCDIF_DbiWriteMem(LCDIF_Type *base, uint8_t displayIndex)

Send pixel data in frame buffer to panel controller memory.

This function starts sending the pixel data in frame buffer to panel controller, user can monitor interrupt kLCDIF_Display0FrameDoneInterrupt to know when then data sending finished.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

void LCDIF_SetFrameBufferConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_fb_config_t* *config)

Configure the LCDIF frame buffer.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the legacy video layer, and use LCDIF_SetOverlayFrameBufferConfig to configure the overlay layers.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

void LCDIF_FrameBufferGetDefaultConfig(*lcdif_fb_config_t* *config)

Get default frame buffer configuration.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to get the default configuration for all the 3 layers.

The default configuration is

```
config->enable = true;
config->enableGamma = false;
config->format = kLCDIF_PixelFormatRGB565;
```

Parameters

- config – Pointer to the configuration structure.

static inline void LCDIF_SetFrameBufferAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Set the frame buffer to LCDIF.

Note: The address must be 128 bytes aligned.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

void LCDIF_SetFrameBufferStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

Set the frame buffer stride.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- strideBytes – The stride in byte.

void LCDIF_SetDitherConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dither_config_t* *config)

Set the dither configuration.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Index to configure.
- config – Pointer to the configuration structure.

void LCDIF_SetGammaData(LCDIF_Type *base, uint8_t displayIndex, uint16_t startIndex, const uint32_t *gamma, uint16_t gammaLen)

Set the gamma translation values to the LCDIF gamma table.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startIndex – Start index in the gamma table that the value will be set to.
- gamma – The gamma values to set to the gamma table in LCDIF, could be defined using LCDIF_MAKE_GAMMA_VALUE.
- gammaLen – The length of the gamma.

static inline void LCDIF_EnableInterrupts(LCDIF_Type *base, uint32_t mask)

Enables LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to enable, pass in as OR'ed value of _lcdif_interrupt.

static inline void LCDIF_DisableInterrupts(LCDIF_Type *base, uint32_t mask)

Disable LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.

- mask – The interrupts to disable, pass in as OR'ed value of _lcdif_interrupt.

static inline uint32_t LCDIF_GetAndClearInterruptPendingFlags(LCDIF_Type *base)

Get and clear LCDIF interrupt pending status.

Note: The interrupt must be enabled, otherwise the interrupt flags will not assert.

Parameters

- base – LCDIF peripheral base address.

Returns

The interrupt pending status.

void LCDIF_CursorGetDefaultConfig(*lcdif_cursor_config_t* *config)

Get the hardware cursor default configuration.

The default configuration values are:

```
config->enable = true;
config->format = kLCDIF_CursorMasked;
config->hotspotOffsetX = 0;
config->hotspotOffsetY = 0;
```

Parameters

- config – Pointer to the hardware cursor configuration structure.

void LCDIF_SetCursorConfig(LCDIF_Type *base, const *lcdif_cursor_config_t* *config)

Configure the cursor.

Parameters

- base – LCDIF peripheral base address.
- config – Cursor configuration.

static inline void LCDIF_SetCursorHotspotPosition(LCDIF_Type *base, uint16_t x, uint16_t y)

Set the cursor hotspot postion.

Parameters

- base – LCDIF peripheral base address.
- x – X coordinate of the hotspot, range 0 ~ 8191.
- y – Y coordinate of the hotspot, range 0 ~ 8191.

static inline void LCDIF_SetCursorBufferAddress(LCDIF_Type *base, uint32_t address)

Set the cursor memory address.

Parameters

- base – LCDIF peripheral base address.
- address – Memory address.

void LCDIF_SetCursorColor(LCDIF_Type *base, uint32_t background, uint32_t foreground)

Set the cursor color.

Parameters

- base – LCDIF peripheral base address.
- background – Background color, could be defined use LCDIF_MAKE_CURSOR_COLOR

- foreground – Foreground color, could be defined use LCDIF_MAKE_CURSOR_COLOR

FSL_LCDIF_DRIVER_VERSION

enum _lcdif_polarity_flags

LCDIF signal polarity flags.

Values:

enumerator kLCDIF_VsyncActiveLow

VSYNC active low.

enumerator kLCDIF_VsyncActiveHigh

VSYNC active high.

enumerator kLCDIF_HsyncActiveLow

HSYNC active low.

enumerator kLCDIF_HsyncActiveHigh

HSYNC active high.

enumerator kLCDIF_DataEnableActiveLow

Data enable line active low.

enumerator kLCDIF_DataEnableActiveHigh

Data enable line active high.

enumerator kLCDIF_DriveDataOnFallingClkEdge

Drive data on falling clock edge, capture data on rising clock edge.

enumerator kLCDIF_DriveDataOnRisingClkEdge

Drive data on falling clock edge, capture data on rising clock edge.

enum _lcdif_output_format

LCDIF DPI output format.

Values:

enumerator kLCDIF_Output16BitConfig1

16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator kLCDIF_Output16BitConfig2

16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator kLCDIF_Output16BitConfig3

16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator kLCDIF_Output18BitConfig1

18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator kLCDIF_Output18BitConfig2

18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator kLCDIF_Output24Bit

24-bit.

enum _lcdif_fb_format

LCDIF frame buffer pixel format.

Values:

enumerator kLCDIF_PixelFormatXRGB444

XRGB4444, deprecated, use kLCDIF_PixelFormatXRGB4444 instead.

enumerator kLCDIF_PixelFormatXRGB4444
XRGB4444, 16-bit each pixel, 4-bit each element. R4G4B4 in reference manual.

enumerator kLCDIF_PixelFormatXRGB1555
XRGB1555, 16-bit each pixel, 5-bit each element. R5G5B5 in reference manual.

enumerator kLCDIF_PixelFormatRGB565
RGB565, 16-bit each pixel. R5G6B5 in reference manual.

enumerator kLCDIF_PixelFormatXRGB8888
XRGB8888, 32-bit each pixel, 8-bit each element. R8G8B8 in reference manual.

enum _lcdif_interrupt
LCDIF interrupt and status.

Values:

enumerator kLCDIF_Display0FrameDoneInterrupt
The last pixel of visible area in frame is shown.

enum _lcdif_cursor_format
LCDIF cursor format.

Values:

enumerator kLCDIF_CursorMasked
Masked format.

enumerator kLCDIF_CursorARGB8888
ARGB8888.

enum _lcdif_dbi_cmd_flag
LCDIF DBI command flag.

Values:

enumerator kLCDIF_DbiCmdAddress
Send address (or command).

enumerator kLCDIF_DbiCmdWriteMem
Start write memory.

enumerator kLCDIF_DbiCmdData
Send data.

enumerator kLCDIF_DbiCmdReadMem
Start read memory.

enum _lcdif_dbi_out_format
LCDIF DBI output format.

Values:

enumerator kLCDIF_DbiOutD8RGB332
8-bit data bus width, pixel RGB332. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD8RGB444
8-bit data bus width, pixel RGB444. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD8RGB565
8-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD8RGB666
8-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 3 cycles, data bus 2 LSB not used.

enumerator kLCDIF_DbiOutD8RGB888
8-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 3 cycles.

enumerator kLCDIF_DbiOutD9RGB666
9-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB332
16-bit data bus width, pixel RGB332. For type A or B. 2 pixels sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB444
16-bit data bus width, pixel RGB444. For type A or B. 1 pixel sent in 1 cycle, data bus 4 MSB not used.

enumerator kLCDIF_DbiOutD16RGB565
16-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB666Option1
16-bit data bus width, pixel RGB666. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB666Option2
16-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option1
16-bit data bus width, pixel RGB888. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option2
16-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 2 cycles.

enum __lcdif_dbi_type
LCDIF DBI type.

Values:

enumerator kLCDIF_DbiTypeA_FixedE
Selects DBI type A fixed E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeA_ClockedE
Selects DBI Type A Clocked E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeB
Selects DBI type B, 8080, Intel mode.

enum __lcdif_dbi_out_swizzle
LCDIF DBI output swizzle.

Values:

enumerator kLCDIF_DbiOutSwizzleRGB
RGB

enumerator kLCDIF_DbiOutSwizzleBGR
BGR

typedef enum *_lcdif_output_format* lcdif_output_format_t
LCDIF DPI output format.

typedef struct *_lcdif_dpi_config* lcdif_dpi_config_t
Configuration for LCDIF module to work in DBI mode.

typedef enum *_lcdif_fb_format* lcdif_fb_format_t
LCDIF frame buffer pixel format.

typedef struct *_lcdif_fb_config* lcdif_fb_config_t
LCDIF frame buffer configuration.

typedef enum *_lcdif_cursor_format* lcdif_cursor_format_t

LCDIF cursor format.

typedef struct *_lcdif_cursor_config* lcdif_cursor_config_t

LCDIF cursor configuration.

typedef struct *_lcdif_dither_config* lcdif_dither_config_t

LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

typedef enum *_lcdif_dbi_out_format* lcdif_dbi_out_format_t

LCDIF DBI output format.

typedef enum *_lcdif_dbi_type* lcdif_dbi_type_t

LCDIF DBI type.

typedef enum *_lcdif_dbi_out_swizzle* lcdif_dbi_out_swizzle_t

LCDIF DBI output swizzle.

typedef struct *_lcdif_dbi_config* lcdif_dbi_config_t

LCDIF DBI configuration.

LCDIF_MAKE_CURSOR_COLOR(r, g, b)

Construct the cursor color, every element should be in the range of 0 ~ 255.

LCDIF_MAKE_GAMMA_VALUE(r, g, b)

Construct the gamma value set to LCDIF gamma table, every element should be in the range of 0~255.

LCDIF_ALIGN_ADDR(addr, align)

Calculate the aligned address for LCDIF buffer.

LCDIF_FB_ALIGN

The frame buffer should be 128 byte aligned.

LCDIF_GAMMA_INDEX_MAX

Gamma index max value.

LCDIF_CURSOR_SIZE

The cursor size is 32 x 32.

LCDIF_FRAMEBUFFERCONFIG0_OUTPUT_MASK

LCDIF_ADDR_CPU_2_IP(addr)

struct _lcdif_dpi_config

#include <fsl_lcdif.h> Configuration for LCDIF module to work in DBI mode.

Public Members

uint16_t panelWidth
: Display panel width, pixels per line.

uint16_t panelHeight
: Display panel height, how many lines per panel.

uint8_t hsw
: HSYNC pulse width.

uint8_t hfp
: Horizontal front porch.

uint8_t hbp
: Horizontal back porch.

uint8_t vsw
: VSYNC pulse width.

uint8_t vfp
: Vrtical front porch.

uint8_t vbp
: Vertical back porch.

uint32_t polarityFlags
: OR'ed value of _lcdif_polarity_flags, used to contol the signal polarity.

lcdif_output_format_t format
: DPI output format.

struct __lcdif_fb_config
: *#include <fsl_lcdif.h>* LCDIF frame buffer configuration.

Public Members

bool enable
: Enable the frame buffer output.

bool enableGamma
: Enable the gamma correction.

lcdif_fb_format_t format
: Frame buffer pixel format.

struct __lcdif_cursor_config
: *#include <fsl_lcdif.h>* LCDIF cursor configuration.

Public Members

bool enable
: Enable the cursor or not.

lcdif_cursor_format_t format
: Cursor format.

uint8_t hotspotOffsetX
: Offset of the hotspot to top left point, range 0 ~ 31

uint8_t hotspotOffsetY

Offset of the hotspot to top left point, range 0 ~ 31

struct __lcdif_dither_config

#include <fsl_lcdif.h> LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

Public Members

bool enable

Enable or not.

uint8_t redSize

Red color size, valid region 4-8.

uint8_t greenSize

Green color size, valid region 4-8.

uint8_t blueSize

Blue color size, valid region 4-8.

uint32_t low

Low part of the look up table.

uint32_t high

High part of the look up table.

struct __lcdif_dbi_config

#include <fsl_lcdif.h> LCDIF DBI configuration.

Public Members

lcdif_dbi_out_swizzle_t swizzle

Swizzle.

lcdif_dbi_out_format_t format

Output format.

uint8_t acTimeUnit

Time unit for AC characteristics.

lcdif_dbi_type_t type

DBI type.

uint16_t writeWRPeriod

WR signal period, Cycle number = writeWRPeriod * (acTimeUnit + 1), must be no less than 3. Only for type A and type b.

uint8_t writeWRAssert

Cycle number = writeWRAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to assert E. With kLCDIF_DbiTypeB: Time to assert WRX.

uint8_t writeCSAssert

Cycle number = writeCSAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to assert CSX.

uint16_t writeWRDeassert

Cycle number = writeWRDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to de-assert E. With kLCDIF_DbiTypeB: Time to de-assert WRX.

uint16_t writeCSDeassert

Cycle number = writeCSDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to de-assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to de-assert CSX.

2.19 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.

status_t MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)

Blocks to send a message.

This function waits until the TX register is empty and sends the message. If MU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and returns kStatus_Timeout.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see mu_msg_reg_index_t.
- msg – Message to send.

Return values

- kStatus_Success – Message sent successfully.
- kStatus_Timeout – Timeout occurred while waiting for TX register to be empty.

Returns

status_t

static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
}  Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0);  Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.

Returns

The received message.

status_t MU_ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If MU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

This function provides the same blocking behavior as MU_ReceiveMsg() but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

Note: Both MU_ReceiveMsg() and MU_ReceiveMsgTimeout() are blocking functions. The difference is that this function includes timeout protection while MU_ReceiveMsg() waits indefinitely.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.
- readValue – Pointer to store the received message.

Return values

- kStatus_Success – Message received successfully.
- kStatus_InvalidArgument – Invalid readValue pointer.
- kStatus_Timeout – Timeout occurred while waiting for RX register to be full.

Returns

status_t

uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

Note: Both MU_ReceiveMsg() and MU_ReceiveMsgTimeout() are blocking functions. The difference is that MU_ReceiveMsgTimeout() includes timeout protection while this function waits indefinitely.

Warning: This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use MU_ReceiveMsgTimeout() instead.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.

Returns

The received message.

static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU_FlagsUpdatingFlag is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
}  Wait for previous MU flags updating.

MU_SetFlagsNonBlocking(base, 0U);  Set the mU flags.
```

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

status_t MU_SetFlags(MU_Type *base, uint32_t flags)

Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

If MU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

Return values

- kStatus_Success – Flags were set successfully.
- kStatus_Timeout – Timeout occurred while waiting for flags to update.

Returns

status_t

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.

static inline void MU_MaskHardwareReset(MU_Type *base, bool mask)

Mask hardware reset by the other core.

The other core could call MU_HardwareResetOtherCore() to reset current core. To mask the reset, call this function and pass in true.

Parameters

- base – MU peripheral base address.
- mask – Pass true to mask the hardware reset, pass false to unmask it.

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_OtherSideInResetFlag
The other side is in reset.

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_BUSY_POLL_COUNT

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if a core becomes unresponsive.

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)

2.20 OCOTP: On Chip One-Time Programmable controller.

FSL_OCOTP_DRIVER_VERSION

OCOTP driver version.

_ocotp_status Error codes for the OCOTP driver.

Values:

enumerator kStatus_OCOTP_AccessError
eFuse and shadow register access error.

enumerator kStatus_OCOTP_CrcFail
CRC check failed.

enumerator kStatus_OCOTP_ReloadError
Error happens during reload shadow register.

enumerator kStatus_OCOTP_ProgramFail
Fuse programming failed.

enumerator kStatus_OCOTP_Locked
Fuse is locked and cannot be programmed.

void OCOTP_Init(OCOTP_Type *base, uint32_t srcClock_Hz)
Initializes OCOTP controller.

Parameters

- base – OCOTP peripheral base address.
- srcClock_Hz – source clock frequency in unit of Hz. When the macro FSL_FEATURE_OCOTP_HAS_TIMING_CTRL is defined as 0, this parameter is not used, application could pass in 0 in this case.

void OCOTP_Deinit(OCOTP_Type *base)
De-initializes OCOTP controller.

Return values
kStatus_Success – upon successful execution, error status otherwise.

static inline bool OCOTP_CheckBusyStatus(OCOTP_Type *base)
Checking the BUSY bit in CTRL register. Checking this BUSY bit will help confirm if the OCOTP controller is ready for access.

Parameters

- base – OCOTP peripheral base address.

Return values
true – for bit set and false for cleared.

static inline bool OCOTP_CheckErrorStatus(OCOTP_Type *base)
Checking the ERROR bit in CTRL register.

Parameters

- base – OCOTP peripheral base address.

Return values
true – for bit set and false for cleared.

static inline void OCOTP_ClearErrorStatus(OCOTP_Type *base)
Clear the error bit if this bit is set.

Parameters

- base – OCOTP peripheral base address.

status_t OCOTP_ReloadShadowRegister(OCOTP_Type *base)

Reload the shadow register. This function will help reload the shadow register without reseting the OCOTP module. Please make sure the OCOTP has been initialized before calling this API.

Parameters

- base – OCOTP peripheral base addess.

Return values

- kStatus_Success – Reload success.
- kStatus_OCOTP_ReloadError – Reload failed.

uint32_t OCOTP_ReadFuseShadowRegister(OCOTP_Type *base, uint32_t address)

Read the fuse shadow register with the fuse addess.

Deprecated:

Use OCOTP_ReadFuseShadowRegisterExt instead of this function.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.

Returns

The read out data.

status_t OCOTP_ReadFuseShadowRegisterExt(OCOTP_Type *base, uint32_t address, uint32_t *data, uint8_t fuseWords)

Read the fuse shadow register from the fuse addess.

This function reads fuse from address, how many words to read is specified by the parameter fuseWords. This function could read at most OCOTP_READ_FUSE_DATA_COUNT fuse word one time.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.
- data – Data array to save the readout fuse value.
- fuseWords – How many words to read.

Return values

- kStatus_Success – Read success.
- kStatus_Fail – Error occurs during read.

status_t OCOTP_WriteFuseShadowRegister(OCOTP_Type *base, uint32_t address, uint32_t data)

Write the fuse shadow register with the fuse addess and data. Please make sure the wrtie address is not locked while calling this API.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be written.
- data – the value will be writen to fuse address.

Return values

write – status, kStatus_Success for success and kStatus_Fail for failed.

status_t OCOTP_WriteFuseShadowRegisterWithLock(OCOTP_Type *base, uint32_t address, uint32_t data, bool lock)

Write the fuse shadow register and lock it.

Please make sure the wrtie address is not locked while calling this API.

Some OCOTP controller supports ECC mode and redundancy mode (see reference mananual for more details). OCOTP controller will auto select ECC or redundancy mode to program the fuse word according to fuse map definition. In ECC mode, the 32 fuse bits in one word can only be written once. In redundancy mode, the word can be written more than once as long as they are different fuse bits. Set parameter lock as true to force use ECC mode.

Parameters

- base – OCOTP peripheral base address.
- address – The fuse address to be written.
- data – The value will be writen to fuse address.
- lock – Lock or unlock write fuse shadow register operation.

Return values

- kStatus_Success – Program and reload success.
- kStatus_OCOTP_Locked – The eFuse word is locked and cannot be programmed.
- kStatus_OCOTP_ProgramFail – eFuse word programming failed.
- kStatus_OCOTP_ReloadError – eFuse word programming success, but error happens during reload the values.
- kStatus_OCOTP_AccessError – Cannot access eFuse word.

static inline uint32_t OCOTP_GetVersion(OCOTP_Type *base)

Get the OCOTP controller version from the register.

Parameters

- base – OCOTP peripheral base address.

Return values

return – the version value.

OCOTP_READ_FUSE_DATA_COUNT

2.21 PDM: Microphone Interface

2.22 PDM Driver

void PDM_Init(PDM_Type *base, const *pdm_config_t* *config)

Initializes the PDM peripheral.

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by PDM_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

Parameters

- base – PDM base pointer
- config – PDM configuration structure.

void PDM_Deinit(PDM_Type *base)

De-initializes the PDM peripheral.

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

Parameters

- base – PDM base pointer

static inline void PDM_Reset(PDM_Type *base)

Resets the PDM module.

Parameters

- base – PDM base pointer

static inline void PDM_Enable(PDM_Type *base, bool enable)

Enables/disables PDM interface.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled, false means PDM interface is disabled.

static inline void PDM_EnableDoze(PDM_Type *base, bool enable)

Enables/disables DOZE.

Parameters

- base – PDM base pointer
- enable – True means the module will enter Disable/Low Leakage mode when ipg_doze is asserted, false means the module will not enter Disable/Low Leakage mode when ipg_doze is asserted.

static inline void PDM_EnableDebugMode(PDM_Type *base, bool enable)

Enables/disables debug mode for PDM. The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface enter debug mode, false means PDM interface in normal mode.

static inline void PDM_EnableInDebugMode(PDM_Type *base, bool enable)

Enables/disables PDM interface in debug mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode.

static inline void PDM_EnterLowLeakageMode(PDM_Type *base, bool enable)

Enables/disables PDM interface disable/Low Leakage mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode.

static inline void PDM_EnableChannel(PDM_Type *base, uint8_t channel, bool enable)

Enables/disables the PDM channel.

Parameters

- base – PDM base pointer
- channel – PDM channel number need to enable or disable.
- enable – True means enable PDM channel, false means disable.

void PDM_SetChannelConfig(PDM_Type *base, uint32_t channel, const *pdm_channel_config_t* *config)

PDM one channel configurations.

Parameters

- base – PDM base pointer
- config – PDM channel configurations.
- channel – channel number. after completing the current frame in debug mode.

status_t PDM_SetSampleRateConfig(PDM_Type *base, uint32_t sourceClock_HZ, uint32_t sampleRate_HZ)

PDM set sample rate.

Note: This function is depend on the configuration of the PDM and PDM channel, so the correct call sequence is

```
PDM_Init(base, pdmConfig)
PDM_SetChannelConfig(base, channel, &channelConfig)
PDM_SetSampleRateConfig(base, source, sampleRate)
```

Parameters

- base – PDM base pointer
- sourceClock_HZ – PDM source clock frequency.
- sampleRate_HZ – PDM sample rate.

status_t PDM_SetSampleRate(PDM_Type *base, uint32_t enableChannelMask, *pdm_df_quality_mode_t* qualityMode, uint8_t osr, uint32_t clkDiv)

PDM set sample rate.

Deprecated:

Do not use this function. It has been superceded by PDM_SetSampleRateConfig

Parameters

- base – PDM base pointer
- enableChannelMask – PDM channel enable mask.
- qualityMode – quality mode.
- osr – cic oversample rate

- clkDiv – clock divider

uint32_t PDM_GetInstance(PDM_Type *base)

Get the instance number for PDM.

Parameters

- base – PDM base pointer.

static inline uint32_t PDM_GetStatus(PDM_Type *base)

Gets the PDM internal status flag. Use the Status Mask in _pdm_internal_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

PDM status flag value.

static inline uint32_t PDM_GetFifoStatus(PDM_Type *base)

Gets the PDM FIFO status flag. Use the Status Mask in _pdm_fifo_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

FIFO status.

static inline uint32_t PDM_GetRangeStatus(PDM_Type *base)

Gets the PDM Range status flag. Use the Status Mask in _pdm_range_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

output status.

static inline void PDM_ClearStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status between kPDM_StatusFrequencyLow and kPDM_StatusCh7FifoDataAvaliable.

static inline void PDM_ClearFIFOStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask.It can be a combination of the status in _pdm_fifo_status.

static inline void PDM_ClearRangeStatus(PDM_Type *base, uint32_t mask)

Clears the PDM range status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in _pdm_range_status.

void PDM_EnableInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_ErrorInterruptEnable
  - kPDM_FIFOInterruptEnable

static inline void PDM_DisableInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_ErrorInterruptEnable
  - kPDM_FIFOInterruptEnable

static inline void PDM_EnableDMA(PDM_Type *base, bool enable)

Enables/disables the PDM DMA requests.

Parameters

- base – PDM base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t PDM_GetDataRegisterAddress(PDM_Type *base, uint32_t channel)

Gets the PDM data register address.

This API is used to provide a transfer address for the PDM DMA transfer configuration.

Parameters

- base – PDM base pointer.
- channel – Which data channel used.

Returns

data register address.

void PDM_ReadFifo(PDM_Type *base, uint32_t startChannel, uint32_t channelNums, void *buffer, size_t size, uint32_t dataWidth)

PDM read fifo.

Note: : This function support 16 bit only for IP version that only supports 16bit.

Parameters

- base – PDM base pointer.
- startChannel – start channel number.
- channelNums – total enabled channelnums.
- buffer – received buffer address.
- size – number of samples to read.

- dataWidth – sample width.

void PDM_SetChannelGain(PDM_Type *base, uint32_t channel, *pdm_df_output_gain_t* gain)

Set the PDM channel gain.

Please note for different quality mode, the valid gain value is different, reference RM for detail.

Parameters

- base – PDM base pointer.
- channel – PDM channel index.
- gain – channel gain, the register gain value range is 0 - 15.

void PDM_SetHwvadConfig(PDM_Type *base, const *pdm_hwvad_config_t* *config)

Configure voice activity detector.

Parameters

- base – PDM base pointer
- config – Voice activity detector configure structure pointer .

static inline void PDM_ForceHwvadOutputDisable(PDM_Type *base, bool enable)

PDM hwvad force output disable.

Parameters

- base – PDM base pointer
- enable – true is output force disable, false is output not force.

static inline void PDM_ResetHwvad(PDM_Type *base)

PDM hwvad reset. It will reset VADNDATA register and will clean all internal buffers, should be called when the PDM isn't running.

Parameters

- base – PDM base pointer

static inline void PDM_EnableHwvad(PDM_Type *base, bool enable)

Enable/Disable Voice activity detector. Should be called when the PDM isn't running.

Parameters

- base – PDM base pointer.
- enable – True means enable voice activity detector, false means disable.

static inline void PDM_EnableHwvadInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM Voice Detector interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_HWVADErrorInterruptEnable
  - kPDM_HWVADInterruptEnable

static inline void PDM_DisableHwvadInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM Voice Detector interrupt requests.

Parameters

- base – PDM base pointer

- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_HWVADErrorInterruptEnable
  - kPDM_HWVADInterruptEnable

static inline void PDM_ClearHwvadInterruptStatusFlags(PDM_Type *base, uint32_t mask)

Clears the PDM voice activity detector status flags.

Parameters

- base – PDM base pointer
- mask – State mask,reference _pdm_hwvad_int_status.

static inline uint32_t PDM_GetHwvadInterruptStatusFlags(PDM_Type *base)

Clears the PDM voice activity detector status flags.

Parameters

- base – PDM base pointer

Returns

status, reference _pdm_hwvad_int_status

static inline uint32_t PDM_GetHwvadInitialFlag(PDM_Type *base)

Get the PDM voice activity detector initial flags.

Parameters

- base – PDM base pointer

Returns

initial flag.

static inline void PDM_EnableHwvadSignalFilter(PDM_Type *base, bool enable)

Enables/disables voice activity detector signal filter.

Parameters

- base – PDM base pointer
- enable – True means enable signal filter, false means disable.

void PDM_SetHwvadSignalFilterConfig(PDM_Type *base, bool enableMaxBlock, uint32_t signalGain)

Configure voice activity detector signal filter.

Parameters

- base – PDM base pointer
- enableMaxBlock – If signal maximum block enabled.
- signalGain – Gain value for the signal energy.

void PDM_SetHwvadNoiseFilterConfig(PDM_Type *base, const *pdm_hwvad_noise_filter_t* *config)

Configure voice activity detector noise filter.

Parameters

- base – PDM base pointer
- config – Voice activity detector noise filter configure structure pointer .

static inline void PDM_EnableHwvadZeroCrossDetector(PDM_Type *base, bool enable)

Enables/disables voice activity detector zero cross detector.

Parameters

- base – PDM base pointer
- enable – True means enable zero cross detector, false means disable.

void PDM_SetHwvadZeroCrossDetectorConfig(PDM_Type *base, const *pdm_hwvad_zero_cross_detector_t* *config)

Configure voice activity detector zero cross detector.

Parameters

- base – PDM base pointer
- config – Voice activity detector zero cross detector configure structure pointer .

static inline uint16_t PDM_GetNoiseData(PDM_Type *base)

Reads noise data.

Parameters

- base – PDM base pointer.

Returns

Data in PDM noise data register.

static inline void PDM_SetHwvadInternalFilterStatus(PDM_Type *base, *pdm_hwvad_filter_status_t* status)

set hwvad internal filter status . Note: filter initial status should be asserted for two more cycles, then set it to normal operation.

Parameters

- base – PDM base pointer.
- status – internal filter status.

void PDM_SetHwvadInEnvelopeBasedMode(PDM_Type *base, const *pdm_hwvad_config_t* *hwvadConfig, const *pdm_hwvad_noise_filter_t* *noiseConfig, const *pdm_hwvad_zero_cross_detector_t* *zcdConfig, uint32_t signalGain)

set HWVAD in envelope based mode . Recommand configurations,

```
static const pdm_hwvad_config_t hwvadConfig = {
  .channel          = 0,
  .initializeTime   = 10U,
  .cicOverSampleRate = 0U,
  .inputGain        = 0U,
  .frameTime        = 10U,
  .cutOffFreq       = kPDM_HwvadHpfBypassed,
  .enableFrameEnergy = false,
  .enablePreFilter  = true,
};

static const pdm_hwvad_noise_filter_t noiseFilterConfig = {
  .enableAutoNoiseFilter = false,
  .enableNoiseMin        = true,
  .enableNoiseDecimation = true,
  .noiseFilterAdjustment = 0U,
  .noiseGain             = 7U,
  .enableNoiseDetectOR   = true,
};
```

Parameters

- base – PDM base pointer.
- hwvadConfig – internal filter status.
- noiseConfig – Voice activity detector noise filter configure structure pointer.
- zcdConfig – Voice activity detector zero cross detector configure structure pointer .
- signalGain – signal gain value.

void PDM_SetHwvadInEnergyBasedMode(PDM_Type *base, const *pdm_hwvad_config_t* *hwvadConfig, const *pdm_hwvad_noise_filter_t* *noiseConfig, const *pdm_hwvad_zero_cross_detector_t* *zcdConfig, uint32_t signalGain)

brief set HWVAD in energy based mode . Recommand configurations, code static const pdm_hwvad_config_t hwvadConfig = { .channel = 0, .initializeTime = 10U, .cicOverSampleRate = 0U, .inputGain = 0U, .frameTime = 10U, .cutOffFreq = kPDM_HwvadHpfBypassed, .enableFrameEnergy = true, .enablePreFilter = true, };

static const pdm_hwvad_noise_filter_t noiseFilterConfig = { .enableAutoNoiseFilter = true, .enableNoiseMin = false, .enableNoiseDecimation = false, .noiseFilterAdjustment = 0U, .noiseGain = 7U, .enableNoiseDetectOR = false, }; code param base PDM base pointer. param hwvadConfig internal filter status. param noiseConfig Voice activity detector noise filter configure structure pointer. param zcdConfig Voice activity detector zero cross detector configure structure pointer . param signalGain signal gain value, signal gain value should be properly according to application.

void PDM_EnableHwvadInterruptCallback(PDM_Type *base, *pdm_hwvad_callback_t* vadCallback, void *userData, bool enable)

Enable/Disable hwvad callback.

This function enable/disable the hwvad interrupt for the selected PDM peripheral.

Parameters

- base – Base address of the PDM peripheral.
- vadCallback – callback Pointer to store callback function, should be NULL when disable.
- userData – user data.
- enable – true is enable, false is disable.

Return values

None. –

void PDM_TransferCreateHandle(PDM_Type *base, *pdm_handle_t* *handle, *pdm_transfer_callback_t* callback, void *userData)

Initializes the PDM handle.

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

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

status_t PDM_TransferSetChannelConfig(PDM_Type *base, *pdm_handle_t* *handle, uint32_t channel, const *pdm_channel_config_t* *config, uint32_t format)

PDM set channel transfer config.

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- channel – PDM channel.
- config – channel config.
- format – data format, support data width configurations,_pdm_data_width.

Return values

kStatus_PDM_ChannelConfig_Failed – or kStatus_Success.

status_t PDM_TransferReceiveNonBlocking(PDM_Type *base, *pdm_handle_t* *handle, *pdm_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on PDM.

Note: This API returns immediately after the transfer initiates. Call the PDM_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_PDM_Busy, the transfer is finished.

Parameters

- base – PDM base pointer
- handle – Pointer to the pdm_handle_t structure which stores the transfer state.
- xfer – Pointer to the pdm_transfer_t structure.

Return values

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

void PDM_TransferAbortReceive(PDM_Type *base, *pdm_handle_t* *handle)

Aborts the current IRQ receive.

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

Parameters

- base – PDM base pointer
- handle – Pointer to the pdm_handle_t structure which stores the transfer state.

void PDM_TransferHandleIRQ(PDM_Type *base, *pdm_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – PDM base pointer.
- handle – Pointer to the pdm_handle_t structure.

FSL_PDM_DRIVER_VERSION

Version 2.9.3

PDM return status.

Values:

enumerator kStatus_PDM_Busy
PDM is busy.

enumerator kStatus_PDM_CLK_LOW
PDM clock frequency low

enumerator kStatus_PDM_FIFO_ERROR
PDM FIFO underrun or overflow

enumerator kStatus_PDM_QueueFull
PDM FIFO underrun or overflow

enumerator kStatus_PDM_Idle
PDM is idle

enumerator kStatus_PDM_Output_ERROR
PDM is output error

enumerator kStatus_PDM_ChannelConfig_Failed
PDM channel config failed

enumerator kStatus_PDM_HWVAD_VoiceDetected
PDM hwvad voice detected

enumerator kStatus_PDM_HWVAD_Error
PDM hwvad error

enum _pdm_interrupt_enable
The PDM interrupt enable flag.

Values:

enumerator kPDM_ErrorInterruptEnable
PDM channel error interrupt enable.

enumerator kPDM_FIFOInterruptEnable
PDM channel FIFO interrupt

enum _pdm_internal_status
The PDM status.

Values:

enumerator kPDM_StatusDfBusyFlag
Decimation filter is busy processing data

enumerator kPDM_StatusFIRFilterReady
FIR filter data is ready

enumerator kPDM_StatusFrequencyLow
Mic app clock frequency not high enough

enumerator kPDM_StatusCh0FifoDataAvaliable
channel 0 fifo data reached watermark level

enumerator kPDM_StatusCh1FifoDataAvaliable
channel 1 fifo data reached watermark level

enumerator kPDM_StatusCh2FifoDataAvaliable
channel 2 fifo data reached watermark level

enumerator kPDM_StatusCh3FifoDataAvaliable
channel 3 fifo data reached watermark level

enum _pdm_channel_enable_mask
PDM channel enable mask.

Values:

enumerator kPDM_EnableChannel0
channgel 0 enable mask

enumerator kPDM_EnableChannel1
channgel 1 enable mask

enumerator kPDM_EnableChannel2
channgel 2 enable mask

enumerator kPDM_EnableChannel3
channgel 3 enable mask

enumerator kPDM_EnableChannelAll

enum _pdm_fifo_status
The PDM fifo status.

Values:

enumerator kPDM_FifoStatusUnderflowCh0
channel0 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh1
channel1 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh2
channel2 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh3
channel3 fifo status underflow

enumerator kPDM_FifoStatusOverflowCh0
channel0 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh1
channel1 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh2
channel2 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh3
channel3 fifo status overflow

enum _pdm_range_status
The PDM output status.

Values:

enumerator kPDM_RangeStatusUnderFlowCh0
channel0 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh1
channel1 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh2
channel2 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh3
channel3 range status underflow

enumerator kPDM_RangeStatusOverFlowCh0
channel0 range status overflow

enumerator kPDM_RangeStatusOverFlowCh1
channel1 range status overflow

enumerator kPDM_RangeStatusOverFlowCh2
channel2 range status overflow

enumerator kPDM_RangeStatusOverFlowCh3
channel3 range status overflow

enum _pdm_dc_remover
PDM DC remover configurations.

Values:

enumerator kPDM_DcRemoverCutOff21Hz
DC remover cut off 21HZ

enumerator kPDM_DcRemoverCutOff83Hz
DC remover cut off 83HZ

enumerator kPDM_DcRemoverCutOff152Hz
DC remover cut off 152HZ

enumerator kPDM_DcRemoverBypass
DC remover bypass

enum _pdm_df_quality_mode
PDM decimation filter quality mode.

Values:

enumerator kPDM_QualityModeMedium
quality mode memdium

enumerator kPDM_QualityModeHigh
quality mode high

enumerator kPDM_QualityModeLow
quality mode low

enumerator kPDM_QualityModeVeryLow0
quality mode very low0

enumerator kPDM_QualityModeVeryLow1
quality mode very low1

enumerator kPDM_QualityModeVeryLow2
quality mode very low2

enum _pdm_qulaity_mode_k_factor
PDM quality mode K factor.

Values:

enumerator kPDM_QualityModeHighKFactor
high quality mode K factor = 1 / 2

enumerator kPDM_QualityModeMediumKFactor
medium/very low0 quality mode K factor = 2 / 2

enumerator kPDM_QualityModeLowKFactor
low/very low1 quality mode K factor = 4 / 2

enumerator kPDM_QualityModeVeryLow2KFactor
very low2 quality mode K factor = 8 / 2

enum _pdm_df_output_gain
PDM decimation filter output gain.

Values:

enumerator kPDM_DfOutputGain0
Decimation filter output gain 0

enumerator kPDM_DfOutputGain1
Decimation filter output gain 1

enumerator kPDM_DfOutputGain2
Decimation filter output gain 2

enumerator kPDM_DfOutputGain3
Decimation filter output gain 3

enumerator kPDM_DfOutputGain4
Decimation filter output gain 4

enumerator kPDM_DfOutputGain5
Decimation filter output gain 5

enumerator kPDM_DfOutputGain6
Decimation filter output gain 6

enumerator kPDM_DfOutputGain7
Decimation filter output gain 7

enumerator kPDM_DfOutputGain8
Decimation filter output gain 8

enumerator kPDM_DfOutputGain9
Decimation filter output gain 9

enumerator kPDM_DfOutputGain10
Decimation filter output gain 10

enumerator kPDM_DfOutputGain11
Decimation filter output gain 11

enumerator kPDM_DfOutputGain12
Decimation filter output gain 12

enumerator kPDM_DfOutputGain13
Decimation filter output gain 13

enumerator kPDM_DfOutputGain14
Decimation filter output gain 14

enumerator kPDM_DfOutputGain15
Decimation filter output gain 15

enum _pdm_data_width
PDM data width.

Values:

enumerator kPDM_DataWwidth24
PDM data width 24bit

enumerator kPDM_DataWwidth32
PDM data width 32bit

enum _pdm_hwvad_interrupt_enable
PDM voice activity detector interrupt type.

Values:

enumerator kPDM_HwvadErrorInterruptEnable
PDM channel HWVAD error interrupt enable.

enumerator kPDM_HwvadInterruptEnable
PDM channel HWVAD interrupt

enum _pdm_hwvad_int_status
The PDM hwvad interrupt status flag.

Values:

enumerator kPDM_HwvadStatusInputSaturation
HWVAD saturation condition

enumerator kPDM_HwvadStatusVoiceDetectFlag
HWVAD voice detect interrupt triggered

enum _pdm_hwvad_hpf_config
High pass filter configure cut-off frequency.

Values:

enumerator kPDM_HwvadHpfBypassed
High-pass filter bypass

enumerator kPDM_HwvadHpfCutOffFreq1750Hz
High-pass filter cut off frequency 1750HZ

enumerator kPDM_HwvadHpfCutOffFreq215Hz
High-pass filter cut off frequency 215HZ

enumerator kPDM_HwvadHpfCutOffFreq102Hz
High-pass filter cut off frequency 102HZ

enum _pdm_hwvad_filter_status
HWVAD internal filter status.

Values:

enumerator kPDM_HwvadInternalFilterNormalOperation
internal filter ready for normal operation

enumerator kPDM_HwvadInternalFilterInitial
interla filter are initial

enum __pdm__hwvad__zcd__result
PDM voice activity detector zero cross detector result.

Values:

enumerator kPDM__HwvadResultOREnergyBasedDetection
zero cross detector result will be OR with energy based detection

enumerator kPDM__HwvadResultANDEnergyBasedDetection
zero cross detector result will be AND with energy based detection

typedef enum *_pdm_dc_remover* pdm__dc__remover__t
PDM DC remover configurations.

typedef enum *_pdm_df_quality_mode* pdm__df__quality__mode__t
PDM decimation filter quality mode.

typedef enum *_pdm_df_output_gain* pdm__df__output__gain__t
PDM decimation filter output gain.

typedef struct *_pdm_channel_config* pdm__channel__config__t
PDM channel configurations.

typedef struct *_pdm_config* pdm__config__t
PDM user configuration structure.

typedef enum *_pdm_hwvad_hpf_config* pdm__hwvad__hpf__config__t
High pass filter configure cut-off frequency.

typedef enum *_pdm_hwvad_filter_status* pdm__hwvad__filter__status__t
HWVAD internal filter status.

typedef struct *_pdm_hwvad_config* pdm__hwvad__config__t
PDM voice activity detector user configuration structure.

typedef struct *_pdm_hwvad_noise_filter* pdm__hwvad__noise__filter__t
PDM voice activity detector noise filter user configuration structure.

typedef enum *_pdm_hwvad_zcd_result* pdm__hwvad__zcd__result__t
PDM voice activity detector zero cross detector result.

typedef struct *_pdm_hwvad_zero_cross_detector* pdm__hwvad__zero__cross__detector__t
PDM voice activity detector zero cross detector configuration structure.

typedef struct *_pdm_transfer* pdm__transfer__t
PDM SDMA transfer structure.

typedef struct *_pdm_handle* pdm__handle__t
PDM handle.

typedef void (*pdm__transfer__callback__t)(PDM_Type *base, *pdm_handle_t* *handle, *status_t* status, void *userData)
PDM transfer callback prototype.

typedef void (*pdm__hwvad__callback__t)(*status_t* status, void *userData)
PDM HWVAD callback prototype.

typedef struct *_pdm_hwvad_notification* pdm__hwvad__notification__t
PDM HWVAD notification structure.

PDM__XFER__QUEUE__SIZE
PDM XFER QUEUE SIZE.

struct __pdm__channel__config
#include <fsl_pdm.h> PDM channel configurations.

Public Members

pdm_dc_remover_t cutOffFreq
DC remover cut off frequency

pdm_df_output_gain_t gain
Decimation Filter Output Gain

struct __pdm__config
#include <fsl_pdm.h> PDM user configuration structure.

Public Members

bool enableDoze
This module will enter disable/low leakage mode if DOZEN is active with ipg_doze is asserted

uint8_t fifoWatermark
Watermark value for FIFO

pdm_df_quality_mode_t qualityMode
Quality mode

uint8_t cicOverSampleRate
CIC filter over sampling rate

struct __pdm__hwvad__config
#include <fsl_pdm.h> PDM voice activity detector user configuration structure.

Public Members

uint8_t channel
Which channel uses voice activity detector

uint8_t initializeTime
Number of frames or samples to initialize voice activity detector.

uint8_t cicOverSampleRate
CIC filter over sampling rate

uint8_t inputGain
Voice activity detector input gain

uint32_t frameTime
Voice activity frame time

pdm_hwvad_hpf_config_t cutOffFreq
High pass filter cut off frequency

bool enableFrameEnergy
If frame energy enabled, true means enable

bool enablePreFilter
If pre-filter enabled

struct __pdm__hwvad__noise__filter
#include <fsl_pdm.h> PDM voice activity detector noise filter user configuration structure.

Public Members

bool enableAutoNoiseFilter
: If noise fileter automatically activated, true means enable

bool enableNoiseMin
: If Noise minimum block enabled, true means enabled

bool enableNoiseDecimation
: If enable noise input decimation

bool enableNoiseDetectOR
: Enables a OR logic in the output of minimum noise estimator block

uint32_t noiseFilterAdjustment
: The adjustment value of the noise filter

uint32_t noiseGain
: Gain value for the noise energy or envelope estimated

struct __pdm_hwvad_zero_cross_detector
: *#include <fsl_pdm.h>* PDM voice activity detector zero cross detector configuration structure.

Public Members

bool enableAutoThreshold
: If ZCD auto-threshold enabled, true means enabled.

pdm_hwvad_zcd_result_t zcdAnd
: Is ZCD result is AND'ed with energy-based detection, false means OR'ed

uint32_t threshold
: The adjustment value of the noise filter

uint32_t adjustmentThreshold
: Gain value for the noise energy or envelope estimated

struct __pdm_transfer
: *#include <fsl_pdm.h>* PDM SDMA transfer structure.

Public Members

volatile uint8_t *data
: Data start address to transfer.

volatile size_t dataSize
: Total Transfer bytes size.

struct __pdm_hwvad_notification
: *#include <fsl_pdm.h>* PDM HWVAD notification structure.

struct __pdm_handle
: *#include <fsl_pdm.h>* PDM handle structure.

Public Members

uint32_t state
: Transfer status

pdm_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

pdm_transfer_t pdmQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

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

uint32_t format
: data format

uint8_t watermark
: Watermark value

uint8_t startChannel
: end channel

uint8_t channelNums
: Enabled channel number

2.23 PDM SDMA Driver

void PDM_TransferCreateHandleSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle,
pdm_sdma_callback_t callback, void *userData,
sdma_handle_t *dmaHandle, uint32_t eventSource)

Initializes the PDM eDMA handle.

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

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaHandle – eDMA handle pointer, this handle shall be static allocated by users.
- eventSource – PDM event source number.

status_t PDM_TransferReceiveSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle, *pdm_transfer_t* *xfer)

Performs a non-blocking PDM receive using eDMA.

Note: This interface returns immediately after the transfer initiates. Call the PDM_GetReceiveRemainingBytes to poll the transfer status and check whether the PDM transfer is finished.

Parameters

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

Return values

- kStatus_Success – Start a PDM eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – PDM is busy receiving data.

void PDM_TransferAbortReceiveSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle)

Aborts a PDM receive using eDMA.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

void PDM_SetChannelConfigSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle, uint32_t channel, const *pdm_channel_config_t* *config)

PDM channel configurations.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- channel – channel number.
- config – channel configurations.

void PDM_TransferTerminateReceiveSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle)

Terminate all the PDM sdma receive transfer.

Parameters

- base – PDM base pointer.
- handle – PDM SDMA handle pointer.

FSL_PDM_SDMA_DRIVER_VERSION

Version 2.7.1

typedef struct *_pdm_sdma_handle* pdm_sdma_handle_t

typedef void (*pdm_sdma_callback_t)(PDM_Type *base, *pdm_sdma_handle_t* *handle, *status_t* status, void *userData)

PDM eDMA transfer callback function for finish and error.

struct __pdm_sdma_handle

#include <fsl_pdm_sdma.h> PDM DMA transfer handle, users should not touch the content of the handle.

Public Members

sdma_handle_t *dmaHandle

DMA handler for PDM send

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t fifoWidth

fifo width

uint8_t endChannel

The last enabled channel

uint8_t channelNums

total channel numbers

uint32_t count

The transfer data count in a DMA request

uint32_t state

Internal state for PDM eDMA transfer

uint32_t eventSource

PDM event source number

pdm_sdma_callback_t callback

Callback for users while transfer finish or error occurs

void *userData

User callback parameter

sdma_buffer_descriptor_t bdPool[(4U)]

BD pool for SDMA transfer.

pdm_transfer_t pdmQueue[(4U)]

Transfer queue storing queued transfer.

size_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.24 PWM: Pulse Width Modulation Driver

status_t PWM_Init(PWM_Type *base, const *pwm_config_t* *config)

Ungates the PWM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the PWM driver.

Parameters

- base – PWM peripheral base address
- config – Pointer to user's PWM config structure.

Returns

kStatus_Success means success; else failed.

void PWM_Deinit(PWM_Type *base)

Gate the PWM submodule clock.

Parameters

- base – PWM peripheral base address

void PWM_GetDefaultConfig(*pwm_config_t* *config)

Fill in the PWM config struct with the default settings.

The default values are:

```
config->enableStopMode = false;
config->enableDozeMode = false;
config->enableWaitMode = false;
config->enableDozeMode = false;
config->clockSource = kPWM_LowFrequencyClock;
config->prescale = 0U;
config->outputConfig = kPWM_SetAtRolloverAndClearAtcomparison;
config->fifoWater = kPWM_FIFOWaterMark_2;
config->sampleRepeat = kPWM_EachSampleOnce;
config->byteSwap = kPWM_ByteNoSwap;
config->halfWordSwap = kPWM_HalfWordNoSwap;
```

Parameters

- config – Pointer to user's PWM config structure.

static inline void PWM_StartTimer(PWM_Type *base)

Starts the PWM counter when the PWM is enabled.

When the PWM is enabled, it begins a new period, the output pin is set to start a new period while the prescaler and counter are released and counting begins.

Parameters

- base – PWM peripheral base address

static inline void PWM_StopTimer(PWM_Type *base)

Stops the PWM counter when the pwm is disabled.

Parameters

- base – PWM peripheral base address

static inline void PWM_EnableInterrupts(PWM_Type *base, uint32_t mask)

Enables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t

static inline void PWM_DisableInterrupts(PWM_Type *base, uint32_t mask)

Disables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t

static inline uint32_t PWM_GetEnabledInterrupts(PWM_Type *base)

Gets the enabled PWM interrupts.

Parameters

- base – PWM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration pwm_interrupt_enable_t

static inline uint32_t PWM_GetStatusFlags(PWM_Type *base)

Gets the PWM status flags.

Parameters

- base – PWM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration pwm_status_flags_t

static inline void PWM_clearStatusFlags(PWM_Type *base, uint32_t mask)

Clears the PWM status flags.

Parameters

- base – PWM peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration pwm_status_flags_t

static inline uint32_t PWM_GetFIFOAvailable(PWM_Type *base)

Gets the PWM FIFO available.

Parameters

- base – PWM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration pwm_fifo_available_t

static inline void PWM_SetSampleValue(PWM_Type *base, uint32_t value)

Sets the PWM sample value.

Parameters

- base – PWM peripheral base address
- value – The sample value. This is the input to the 4x16 FIFO. The value in this register denotes the value of the sample being currently used.

static inline uint32_t PWM_GetSampleValue(PWM_Type *base)

Gets the PWM sample value.

Parameters

- base – PWM peripheral base address

Returns

The sample value. It can be read only when the PWM is enable.

FSL_PWM_DRIVER_VERSION

enum _pwm_clock_source

PWM clock source select.

Values:

enumerator kPWM_PeripheralClock

The Peripheral clock is used as the clock

enumerator kPWM_HighFrequencyClock

High-frequency reference clock is used as the clock

enumerator kPWM_LowFrequencyClock

Low-frequency reference clock(32KHz) is used as the clock

enum _pwm_fifo_water_mark

PWM FIFO water mark select. Sets the data level at which the FIFO empty flag will be set.

Values:

enumerator kPWM_FIFOWaterMark_1

FIFO empty flag is set when there are more than or equal to 1 empty slots

enumerator kPWM_FIFOWaterMark_2

FIFO empty flag is set when there are more than or equal to 2 empty slots

enumerator kPWM_FIFOWaterMark_3

FIFO empty flag is set when there are more than or equal to 3 empty slots

enumerator kPWM_FIFOWaterMark_4

FIFO empty flag is set when there are more than or equal to 4 empty slots

enum _pwm_byte_data_swap

PWM byte data swap select. It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

Values:

enumerator kPWM_ByteNoSwap

byte ordering remains the same

enumerator kPWM_ByteSwap

byte ordering is reversed

enum _pwm_half_word_data_swap

PWM half-word data swap select.

Values:

enumerator kPWM_HalfWordNoSwap

Half word swapping does not take place

enumerator kPWM_HalfWordSwap

Half word from write data bus are swapped

enum _pwm_output_configuration

PWM Output Configuration.

Values:

enumerator kPWM_SetAtRolloverAndClearAtcomparison

Output pin is set at rollover and cleared at comparison

enumerator kPWM_ClearAtRolloverAndSetAtcomparison

Output pin is cleared at rollover and set at comparison

enumerator kPWM_NoConfigure
PWM output is disconnected

enum _pwm_sample_repeat
PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.

Values:

enumerator kPWM_EachSampleOnce
Use each sample once

enumerator kPWM_EachSampletwice
Use each sample twice

enumerator kPWM_EachSampleFourTimes
Use each sample four times

enumerator kPWM_EachSampleEightTimes
Use each sample eight times

enum _pwm_interrupt_enable
List of PWM interrupt options.

Values:

enumerator kPWM_FIFOEmptyInterruptEnable
This bit controls the generation of the FIFO Empty interrupt.

enumerator kPWM_RolloverInterruptEnable
This bit controls the generation of the Rollover interrupt.

enumerator kPWM_CompareInterruptEnable
This bit controls the generation of the Compare interrupt

enum _pwm_status_flags
List of PWM status flags.

Values:

enumerator kPWM_FIFOEmptyFlag
This bit indicates the FIFO data level in comparison to the water level set by FWM field in the control register.

enumerator kPWM_RolloverFlag
This bit shows that a roll-over event has occurred.

enumerator kPWM_CompareFlag
This bit shows that a compare event has occurred.

enumerator kPWM_FIFOWriteErrorFlag
This bit shows that an attempt has been made to write FIFO when it is full.

enum _pwm_fifo_available
List of PWM FIFO available.

Values:

enumerator kPWM_NoDataInFIFOFlag
No data available

enumerator kPWM_OneWordInFIFOFlag
1 word of data in FIFO

enumerator kPWM_TwoWordsInFIFOFlag
2 word of data in FIFO

enumerator kPWM_ThreeWordsInFIFOFlag
3 word of data in FIFO

enumerator kPWM_FourWordsInFIFOFlag
4 word of data in FIFO

typedef enum *_pwm_clock_source* pwm_clock_source_t
PWM clock source select.

typedef enum *_pwm_fifo_water_mark* pwm_fifo_water_mark_t
PWM FIFO water mark select. Sets the data level at which the FIFO empty flag will be set.

typedef enum *_pwm_byte_data_swap* pwm_byte_data_swap_t
PWM byte data swap select. It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

typedef enum *_pwm_half_word_data_swap* pwm_half_word_data_swap_t
PWM half-word data swap select.

typedef enum *_pwm_output_configuration* pwm_output_configuration_t
PWM Output Configuration.

typedef enum *_pwm_sample_repeat* pwm_sample_repeat_t
PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.

typedef enum *_pwm_interrupt_enable* pwm_interrupt_enable_t
List of PWM interrupt options.

typedef enum *_pwm_status_flags* pwm_status_flags_t
List of PWM status flags.

typedef enum *_pwm_fifo_available* pwm_fifo_available_t
List of PWM FIFO available.

typedef struct *_pwm_config* pwm_config_t

static inline void PWM_SoftwareReset(PWM_Type *base)
Sofrware reset.

PWM is reset when this bit is set to 1. It is a self clearing bit. Setting this bit resets all the registers to their reset values except for the STOPEN, DOZEN, WAITEN, and DBGEN bits in this control register.

Parameters

- base – PWM peripheral base address

static inline void PWM_SetPeriodValue(PWM_Type *base, uint32_t value)
Sets the PWM period value.

Parameters

- base – PWM peripheral base address
- value – The period value. The PWM period register (PWM_PWMPR) determines the period of the PWM output signal. Writing 0xFFFF to this register will achieve the same result as writing 0xFFFE. PWMO (Hz) = PCLK(Hz) / (period +2)

static inline uint32_t PWM_GetPeriodValue(PWM_Type *base)

Gets the PWM period value.

Parameters

- base – PWM peripheral base address

Returns

The period value. The PWM period register (PWM_PWMPR) determines the period of the PWM output signal.

static inline uint32_t PWM_GetCounterValue(PWM_Type *base)

Gets the PWM counter value.

Parameters

- base – PWM peripheral base address

Returns

The counter value. The current count value.

struct __pwm_config

#include <fsl_pwm.h>

Public Members

bool enableStopMode

True: PWM continues to run in stop mode; False: PWM is paused in stop mode.

bool enableDozeMode

True: PWM continues to run in doze mode; False: PWM is paused in doze mode.

bool enableWaitMode

True: PWM continues to run in wait mode; False: PWM is paused in wait mode.

bool enableDebugMode

True: PWM continues to run in debug mode; False: PWM is paused in debug mode.

uint16_t prescale

Pre-scaler to divide down the clock The prescaler value is not more than 0xFFF. Divide by (value + 1)

pwm_clock_source_t clockSource

Clock source for the counter

pwm_output_configuration_t outputConfig

Set the mode of the PWM output on the output pin.

pwm_fifo_water_mark_t fifoWater

Set the data level for FIFO.

pwm_sample_repeat_t sampleRepeat

The number of times each sample from the FIFO is to be used.

pwm_byte_data_swap_t byteSwap

It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

pwm_half_word_data_swap_t halfWordSwap

It determines which half word data from the 32-bit IP Bus interface is written into the lower 16 bits of the sample register.

2.25 RDC: Resource Domain Controller

enum _rdc_interrupts

RDC interrupts.

Values:

enumerator kRDC_RestoreCompleteInterrupt

Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions.

enum _rdc_flags

RDC status.

Values:

enumerator kRDC_PowerDownDomainOn

Power down domain is ON.

enum _rdc_access_policy

Access permission policy.

Values:

enumerator kRDC_NoAccess

Could not read or write.

enumerator kRDC_WriteOnly

Write only.

enumerator kRDC_ReadOnly

Read only.

enumerator kRDC_ReadWrite

Read and write.

typedef struct *_rdc_hardware_config* rdc_hardware_config_t

RDC hardware configuration.

typedef struct *_rdc_domain_assignment* rdc_domain_assignment_t

Master domain assignment.

typedef struct *_rdc_periph_access_config* rdc_periph_access_config_t

Peripheral domain access permission configuration.

typedef struct *_rdc_mem_access_config* rdc_mem_access_config_t

Memory region domain access control configuration.

Note that when setting the rdc_mem_access_config_t::baseAddress and rdc_mem_access_config_t::endAddress, should be aligned to the region resolution, see rdc_mem_t definitions.

typedef struct *_rdc_mem_status* rdc_mem_status_t

Memory region access violation status.

void RDC_Init(RDC_Type *base)

Initializes the RDC module.

This function enables the RDC clock.

Parameters

- base – RDC peripheral base address.

void RDC_Deinit(RDC_Type *base)

De-initializes the RDC module.

This function disables the RDC clock.

Parameters

- base – RDC peripheral base address.

void RDC_GetHardwareConfig(RDC_Type *base, *rdc_hardware_config_t* *config)

Gets the RDC hardware configuration.

This function gets the RDC hardware configurations, including number of bus masters, number of domains, number of memory regions and number of peripherals.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the structure to get the configuration.

static inline void RDC_EnableInterrupts(RDC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – RDC peripheral base address.
- mask – Interrupts to enable, it is OR'ed value of enum _rdc_interrupts.

static inline void RDC_DisableInterrupts(RDC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – RDC peripheral base address.
- mask – Interrupts to disable, it is OR'ed value of enum _rdc_interrupts.

static inline uint32_t RDC_GetInterruptStatus(RDC_Type *base)

Get the interrupt pending status.

Parameters

- base – RDC peripheral base address.

Returns

Interrupts pending status, it is OR'ed value of enum _rdc_interrupts.

static inline void RDC_ClearInterruptStatus(RDC_Type *base, uint32_t mask)

Clear interrupt pending status.

Parameters

- base – RDC peripheral base address.
- mask – Status to clear, it is OR'ed value of enum _rdc_interrupts.

static inline uint32_t RDC_GetStatus(RDC_Type *base)

Get RDC status.

Parameters

- base – RDC peripheral base address.

Returns

mask RDC status, it is OR'ed value of enum _rdc_flags.

static inline void RDC_ClearStatus(RDC_Type *base, uint32_t mask)

Clear RDC status.

Parameters

- base – RDC peripheral base address.
- mask – RDC status to clear, it is OR'ed value of enum _rdc_flags.

void RDC_SetMasterDomainAssignment(RDC_Type *base, rdc_master_t master, const *rdc_domain_assignment_t* *domainAssignment)

Set master domain assignment.

Parameters

- base – RDC peripheral base address.
- master – Which master to set.
- domainAssignment – Pointer to the assignment.

void RDC_GetDefaultMasterDomainAssignment(*rdc_domain_assignment_t* *domainAssignment)

Get default master domain assignment.

The default configuration is:

```
assignment->domainId = 0U;
assignment->lock = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment.

static inline void RDC_LockMasterDomainAssignment(RDC_Type *base, rdc_master_t master)

Lock master domain assignment.

Once locked, it could not be unlocked until next reset.

Parameters

- base – RDC peripheral base address.
- master – Which master to lock.

void RDC_SetPeriphAccessConfig(RDC_Type *base, const *rdc_periph_access_config_t* *config)

Set peripheral access policy.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

void RDC_GetDefaultPeriphAccessConfig(*rdc_periph_access_config_t* *config)

Get default peripheral access policy.

The default configuration is:

```
config->lock = false;
config->enableSema = false;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

- config – Pointer to the policy configuration.

static inline void RDC_LockPeriphAccessConfig(RDC_Type *base, rdc_periph_t periph)

Lock peripheral access policy configuration.

Once locked, it could not be unlocked until reset.

Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to lock.

static inline uint8_t RDC_GetPeriphAccessPolicy(RDC_Type *base, rdc_periph_t periph, uint8_t domainId)

Get the peripheral access policy for specific domain.

Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to get.
- domainId – Get policy for which domain.

Returns

Access policy, see _rdc_access_policy.

void RDC_SetMemAccessConfig(RDC_Type *base, const *rdc_mem_access_config_t* *config)

Set memory region access policy.

Note that when setting the baseAddress and endAddress in config, should be aligned to the region resolution, see rdc_mem_t definitions.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

void RDC_GetDefaultMemAccessConfig(*rdc_mem_access_config_t* *config)

Get default memory region access policy.

The default configuration is:

```
config->lock = false;
config->baseAddress = 0;
config->endAddress = 0;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

- config – Pointer to the policy configuration.

static inline void RDC_LockMemAccessConfig(RDC_Type *base, rdc_mem_t mem)

Lock memory access policy configuration.

Once locked, it could not be unlocked until reset. After locked, you can only call RDC_SetMemAccessValid to enable the configuration, but can not disable it or change other settings.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to lock.

static inline void RDC_SetMemAccessValid(RDC_Type *base, rdc_mem_t mem, bool valid)

Enable or disable memory access policy configuration.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to operate.
- valid – Pass in true to valid, false to invalid.

void RDC_GetMemViolationStatus(RDC_Type *base, rdc_mem_t mem, *rdc_mem_status_t* *status)

Get the memory region violation status.

The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID(s).

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to get.
- status – The returned status.

static inline void RDC_ClearMemViolationFlag(RDC_Type *base, rdc_mem_t mem)

Clear the memory region violation flag.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to clear.

static inline uint8_t RDC_GetMemAccessPolicy(RDC_Type *base, rdc_mem_t mem, uint8_t domainId)

Get the memory region access policy for specific domain.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to get.
- domainId – Get policy for which domain.

Returns

Access policy, see _rdc_access_policy.

static inline uint8_t RDC_GetCurrentMasterDomainId(RDC_Type *base)

Gets the domain ID of the current bus master.

This function returns the domain ID of the current bus master.

Parameters

- base – RDC peripheral base address.

Returns

Domain ID of current bus master.

FSL_RDC_DRIVER_VERSION

RDC_ACCESS_POLICY(domainID, policy)

struct __rdc_hardware_config

#include <fsl_rdc.h> RDC hardware configuration.

Public Members

uint32_t domainNumber
: Number of domains.

uint32_t masterNumber
: Number of bus masters.

uint32_t periphNumber
: Number of peripherals.

uint32_t memNumber
: Number of memory regions.

struct __rdc_domain_assignment
: *#include <fsl_rdc.h>* Master domain assignment.

Public Members

uint32_t domainId
: Domain ID.

uint32_t ___pad0___
: Reserved.

uint32_t lock
: Lock the domain assignment.

struct __rdc_periph_access_config
: *#include <fsl_rdc.h>* Peripheral domain access permission configuration.

Public Members

rdc_periph_t periph
: Peripheral name.

bool lock
: Lock the permission until reset.

bool enableSema
: Enable semaphore or not, when enabled, master should call RDC_SEMA42_Lock to lock the semaphore gate accordingly before access the peripheral.

uint16_t policy
: Access policy.

struct __rdc_mem_access_config
: *#include <fsl_rdc.h>* Memory region domain access control configuration.

  Note that when setting the rdc_mem_access_config_t::baseAddress and rdc_mem_access_config_t::endAddress, should be aligned to the region resolution, see rdc_mem_t definitions.

Public Members

rdc_mem_t mem
: Memory region descriptor name.

bool lock
: Lock the configuration.

uint64_t baseAddress
: Start address of the memory region.

uint64_t endAddress
: End address of the memory region.

uint16_t policy
: Access policy.

struct _rdc_mem_status
: *#include <fsl_rdc.h>* Memory region access violation status.

Public Members

bool hasViolation
: Violating happens or not.

uint8_t domainID
: Violating Domain ID.

uint64_t address
: Violating Address.

2.26 RDC_SEMA42: Hardware Semaphores Driver

FSL_RDC_SEMA42_DRIVER_VERSION
: RDC_SEMA42 driver version.

void RDC_SEMA42_Init(RDC_SEMAPHORE_Type *base)
: Initializes the RDC_SEMA42 module.

  This function initializes the RDC_SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either RDC_SEMA42_ResetGate or RDC_SEMA42_ResetAllGates function.

  Parameters

  - base – RDC_SEMA42 peripheral base address.

void RDC_SEMA42_Deinit(RDC_SEMAPHORE_Type *base)
: De-initializes the RDC_SEMA42 module.

  This function de-initializes the RDC_SEMA42 module. It only disables the clock.

  Parameters

  - base – RDC_SEMA42 peripheral base address.

status_t RDC_SEMA42_TryLock(RDC_SEMAPHORE_Type *base, uint8_t gateNum, uint8_t masterIndex, uint8_t domainId)
: Tries to lock the RDC_SEMA42 gate.

  This function tries to lock the specific RDC_SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

  Parameters

  - base – RDC_SEMA42 peripheral base address.

- gateNum – Gate number to lock.
- masterIndex – Current processor master index.
- domainId – Current processor domain ID.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_Failed – Sema42 gate has been locked by another processor.

void RDC_SEMA42_Lock(RDC_SEMAPHORE_Type *base, uint8_t gateNum, uint8_t masterIndex, uint8_t domainId)

Locks the RDC_SEMA42 gate.

This function locks the specific RDC_SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- masterIndex – Current processor master index.
- domainId – Current processor domain ID.

static inline void RDC_SEMA42_Unlock(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Unlocks the RDC_SEMA42 gate.

This function unlocks the specific RDC_SEMA42 gate. It only writes unlock value to the RDC_SEMA42 gate register. However, it does not check whether the RDC_SEMA42 gate is locked by the current processor or not. As a result, if the RDC_SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline int32_t RDC_SEMA42_GetLockMasterIndex(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Gets which master has currently locked the gate.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is not locked by any master, otherwise return the master index.

int32_t RDC_SEMA42_GetLockDomainID(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Gets which domain has currently locked the gate.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is not locked by any domain, otherwise return the domain ID.

status_t RDC_SEMA42_ResetGate(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Resets the RDC_SEMA42 gate to an unlocked status.

This function resets a RDC_SEMA42 gate to an unlocked status.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – RDC_SEMA42 gate is reset successfully.
- kStatus_Failed – Some other reset process is ongoing.

static inline *status_t* RDC_SEMA42_ResetAllGates(RDC_SEMAPHORE_Type *base)

Resets all RDC_SEMA42 gates to an unlocked status.

This function resets all RDC_SEMA42 gate to an unlocked status.

Parameters

- base – RDC_SEMA42 peripheral base address.

Return values

- kStatus_Success – RDC_SEMA42 is reset successfully.
- kStatus_RDC_SEMA42_Reseting – Some other reset process is ongoing.

RDC_SEMA42_GATE_NUM_RESET_ALL

The number to reset all RDC_SEMA42 gates.

RDC_SEMA42_GATEn(base, n)

RDC_SEMA42 gate n register address.

RDC_SEMA42_GATE_COUNT

RDC_SEMA42 gate count.

RDC_SEMAPHORE_GATE_GTFSM_MASK

2.27 SAI: Serial Audio Interface

2.28 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

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

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_SetMasterClockConfig(I2S_Type *base, *sai_master_clock_t* *config)

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

void SAI_TxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly    = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

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

static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns
SAI Rx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

void SAI_TxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer

- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.

- channel – Which data channel used.

Returns
data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns
data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

- data – Data needs to be written.

void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

void SAI_TransferTxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Tx handle.

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

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.

- userData – User parameter passed to the callback function

void SAI_TransferRxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Rx handle.

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

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

void SAI_TransferTxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – tranmitter configurations.

void SAI_TransferRxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.

- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

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

status_t SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

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

void SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

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

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

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

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.

- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.10

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy
SAI Tx is busy.

enumerator kStatus_SAI_RxBusy
SAI Rx is busy.

enumerator kStatus_SAI_TxError
SAI Tx FIFO error.

enumerator kStatus_SAI_RxError
SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull
SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle
SAI Tx is idle

enumerator kStatus_SAI_RxIdle
SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.

Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.

Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.

Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity

SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh

Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow

Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge

Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge

Sample inputs on rising edge

enum _sai_sync_mode

Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync

Asynchronous mode

enumerator kSAI_ModeSync

Synchronous mode (with receiver or transmit)

enum _sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1

Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable
FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable
FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing
The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate
Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz
Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz
Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz
Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz
Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz
Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz
Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz
Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz
Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz
Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz
Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width
Audio word width.

Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_data_pin_state
sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine
sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled
sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead
sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite
sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite
sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead
sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite
sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type
sai transceiver type

Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

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

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol

Audio bus protocol in SAI

sai_sync_mode_t syncMode

SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource

Bit Clock source

sai_master_slave_t masterSlave

Master or slave

struct _sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz

Sample rate of audio data

uint32_t bitWidth

Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo

Mono or stereo

uint32_t masterClockHz
: Master clock frequency in Hz

uint8_t watermark
: Watermark value

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct __sai__master__clock
: *#include <fsl_sai.h>* master clock configurations

Public Members

bool mclkOutputEnable
: master clock output enable

uint32_t mclkHz
: target mclk frequency

uint32_t mclkSourceClkHz
: mclk source frequency

struct __sai__fifo
: *#include <fsl_sai.h>* sai fifo configurations

Public Members

bool fifoContinueOneError
: fifo continues when error occur

sai_fifo_combine_t fifoCombine
: fifo combine mode

sai_fifo_packing_t fifoPacking
: fifo packing mode

uint8_t fifoWatermark
: fifo watermark

struct __sai__bit__clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkInputDelay
: bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
: bit clock polarity

sai_bclk_source_t bclkSource
: bit Clock source

struct __sai__frame__sync
: *#include <fsl_sai.h>* sai frame sync configurations

Public Members

uint8_t frameSyncWidth
: frame sync width in number of bit clocks

bool frameSyncEarly
: TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
: internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity
: frame sync polarity

struct __sai__serial__data
: *#include <fsl_sai.h>* sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
: sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
: configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
: configure the number of bits in the first word in each frame

uint8_t dataWordNLength
: configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
: used to record the data length for dma transfer

uint8_t dataFirstBitShifted
: Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
: configure the number of words in each frame

uint32_t dataMaskedWord
: configure whether the transmit word is masked

struct __sai__transceiver
: *#include <fsl_sai.h>* sai transceiver configurations

Public Members

sai_serial_data_t serialData
: serial data configurations

sai_frame_sync_t frameSync
: ws configurations

sai_bit_clock_t bitClock
: bit clock configurations

sai_fifo_t fifo
: fifo configurations

sai_master_slave_t masterSlave
: transceiver is master or slave

sai_sync_mode_t syncMode
: transceiver sync mode

uint8_t startChannel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

struct __sai__transfer
: *#include <fsl_sai.h>* SAI transfer structure.

Public Members

uint8_t *data
: Data start address to transfer.

size_t dataSize
: Transfer size.

struct __sai__handle
: *#include <fsl_sai.h>* SAI handle structure.

Public Members

I2S_Type *base
: base address

uint32_t state
: Transfer status

sai_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

uint8_t bitWidth
: Bit width for transfer, 8/16/24/32 bits

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

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

uint8_t watermark
: Watermark value

2.29 SAI SDMA Driver

void SAI_TransferTxCreateHandleSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_sdma_callback_t* callback, void *userData, *sdma_handle_t* *dmaHandle, uint32_t eventSource)

Initializes the SAI SDMA handle.

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

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.
- base – SAI peripheral base address.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – SAI event source number.

void SAI_TransferRxCreateHandleSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_sdma_callback_t* callback, void *userData, *sdma_handle_t* *dmaHandle, uint32_t eventSource)

Initializes the SAI Rx SDMA handle.

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.
- base – SAI peripheral base address.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – SAI event source number.

status_t SAI_TransferSendSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

Note: This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SAI SDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SAI is busy sending data.

status_t SAI_TransferReceiveSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using SDMA.

Note: This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer
- handle – SAI SDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SAI SDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SAI is busy receiving data.

void SAI_TransferAbortSendSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Aborts a SAI transfer using SDMA.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

void SAI_TransferAbortReceiveSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Aborts a SAI receive using SDMA.

Parameters

- base – SAI base pointer
- handle – SAI SDMA handle pointer.

void SAI_TransferTerminateReceiveSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Terminate all the SAI sdma receive transfer.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

void SAI_TransferTerminateSendSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Terminate all the SAI sdma send transfer.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

void SAI_TransferRxSetConfigSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

brief Configures the SAI RX.

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

void SAI_TransferTxSetConfigSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

brief Configures the SAI Tx.

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

FSL_SAI_SDMA_DRIVER_VERSION

Version 2.6.0

typedef struct *_sai_sdma_handle* sai_sdma_handle_t

typedef void (*sai_sdma_callback_t)(I2S_Type *base, *sai_sdma_handle_t* *handle, *status_t* status, void *userData)

SAI SDMA transfer callback function for finish and error.

struct _sai_sdma_handle

#include <fsl_sai_sdma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

sdma_handle_t *dmaHandle
DMA handler for SAI send

uint8_t bytesPerFrame
Bytes in a frame

uint8_t channel
start data channel

uint8_t channelNums
total transfer channel numbers, used for multififo

uint8_t channelMask
enabled channel mask value, refernece _sai_channel_mask

uint8_t fifoOffset
fifo address offset between multifo

uint32_t count
The transfer data count in a DMA request

uint32_t state
Internal state for SAI SDMA transfer

uint32_t eventSource
SAI event source number

sai_sdma_callback_t callback
Callback for users while transfer finish or error occurs

void *userData
User callback parameter

sdma_buffer_descriptor_t bdPool[(4U)]
BD pool for SDMA transfer.

sai_transfer_t saiQueue[(4U)]
Transfer queue storing queued transfer.

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer.

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

2.30 SDMA: Smart Direct Memory Access (SDMA) Controller Driver

void SDMA_Init(SDMAARM_Type *base, const *sdma_config_t* *config)
Initializes the SDMA peripheral.

This function ungates the SDMA clock and configures the SDMA peripheral according to the configuration structure.

Note: This function enables the minor loop map feature.

Parameters

- base – SDMA peripheral base address.
- config – A pointer to the configuration structure, see "sdma_config_t".

void SDMA_Deinit(SDMAARM_Type *base)

Deinitializes the SDMA peripheral.

This function gates the SDMA clock.

Parameters

- base – SDMA peripheral base address.

void SDMA_GetDefaultConfig(*sdma_config_t* *config)

Gets the SDMA default configuration structure.

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

```
config.enableRealTimeDebugPin = false;
config.isSoftwareResetClearLock = true;
config.ratio = kSDMA_HalfARMClockFreq;
```

Parameters

- config – A pointer to the SDMA configuration structure.

void SDMA_ResetModule(SDMAARM_Type *base)

Sets all SDMA core register to reset status.

If only reset ARM core, SDMA register cannot return to reset value, shall call this function to reset all SDMA register to reset value. But the internal status cannot be reset.

Parameters

- base – SDMA peripheral base address.

static inline void SDMA_EnableChannelErrorInterrupts(SDMAARM_Type *base, uint32_t channel)

Enables the interrupt source for the SDMA error.

Enable this will trigger an interrupt while SDMA occurs error while executing scripts.

Parameters

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

static inline void SDMA_DisableChannelErrorInterrupts(SDMAARM_Type *base, uint32_t channel)

Disables the interrupt source for the SDMA error.

Parameters

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

void SDMA_ConfigBufferDescriptor(*sdma_buffer_descriptor_t* *bd, uint32_t srcAddr, uint32_t destAddr, *sdma_transfer_size_t* busWidth, size_t bufferSize, bool isLast, bool enableInterrupt, bool isWrap, *sdma_transfer_type_t* type)

Sets buffer descriptor contents.

This function sets the descriptor contents such as source, dest address and status bits.

Parameters

- bd – Pointer to the buffer descriptor structure.
- srcAddr – Source address for the buffer descriptor.
- destAddr – Destination address for the buffer descriptor.
- busWidth – The transfer width, it only can be a member of sdma_transfer_size_t.
- bufferSize – Buffer size for this descriptor, this number shall less than 0xFFFF. If need to transfer a big size, shall divide into several buffer descriptors.
- isLast – Is the buffer descriptor the last one for the channel to transfer. If only one descriptor used for the channel, this bit shall set to TRUE.
- enableInterrupt – If trigger an interrupt while this buffer descriptor transfer finished.
- isWrap – Is the buffer descriptor need to be wrapped. While this bit set to true, it will automatically wrap to the first buffer descrtiptor to do transfer.
- type – Transfer type, memory to memory, peripheral to memory or memory to peripheral.

static inline void SDMA_SetChannelPriority(SDMAARM_Type *base, uint32_t channel, uint8_t priority)

Set SDMA channel priority.

This function sets the channel priority. The default value is 0 for all channels, priority 0 will prevents channel from starting, so the priority must be set before start a channel.

Parameters

- base – SDMA peripheral base address.
- channel – SDMA channel number.
- priority – SDMA channel priority.

static inline void SDMA_SetSourceChannel(SDMAARM_Type *base, uint32_t source, uint32_t channelMask)

Set SDMA request source mapping channel.

This function sets which channel will be triggered by the dma request source.

Parameters

- base – SDMA peripheral base address.
- source – SDMA dma request source number.
- channelMask – SDMA channel mask. 1 means channel 0, 2 means channel 1, 4 means channel 3. SDMA supports an event trigger multi-channel. A channel can also be triggered by several source events.

static inline void SDMA_StartChannelSoftware(SDMAARM_Type *base, uint32_t channel)

Start a SDMA channel by software trigger.

This function start a channel.

Parameters

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

static inline void SDMA_StartChannelEvents(SDMAARM_Type *base, uint32_t channel)

Start a SDMA channel by hardware events.

This function start a channel.

Parameters

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

static inline void SDMA_StopChannel(SDMAARM_Type *base, uint32_t channel)

Stop a SDMA channel.

This function stops a channel.

Parameters

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

void SDMA_SetContextSwitchMode(SDMAARM_Type *base, *sdma_context_switch_mode_t* mode)

Set the SDMA context switch mode.

Parameters

- base – SDMA peripheral base address.
- mode – SDMA context switch mode.

static inline uint32_t SDMA_GetChannelInterruptStatus(SDMAARM_Type *base)

Gets the SDMA interrupt status of all channels.

Parameters

- base – SDMA peripheral base address.

Returns

The interrupt status for all channels. Check the relevant bits for specific channel.

static inline void SDMA_ClearChannelInterruptStatus(SDMAARM_Type *base, uint32_t mask)

Clear the SDMA channel interrupt status of specific channels.

Parameters

- base – SDMA peripheral base address.
- mask – The interrupt status need to be cleared.

static inline uint32_t SDMA_GetChannelStopStatus(SDMAARM_Type *base)

Gets the SDMA stop status of all channels.

Parameters

- base – SDMA peripheral base address.

Returns

The stop status for all channels. Check the relevant bits for specific channel.

static inline void SDMA_ClearChannelStopStatus(SDMAARM_Type *base, uint32_t mask)

Clear the SDMA channel stop status of specific channels.

Parameters

- base – SDMA peripheral base address.
- mask – The stop status need to be cleared.

static inline uint32_t SDMA_GetChannelPendStatus(SDMAARM_Type *base)

Gets the SDMA channel pending status of all channels.

Parameters

- base – SDMA peripheral base address.

Returns

The pending status for all channels. Check the relevant bits for specific channel.

static inline void SDMA_ClearChannelPendStatus(SDMAARM_Type *base, uint32_t mask)

Clear the SDMA channel pending status of specific channels.

Parameters

- base – SDMA peripheral base address.
- mask – The pending status need to be cleared.

static inline uint32_t SDMA_GetErrorStatus(SDMAARM_Type *base)

Gets the SDMA channel error status.

SDMA channel error flag is asserted while an incoming DMA request was detected and it triggers a channel that is already pending or being serviced. This probably means there is an overflow of data for that channel.

Parameters

- base – SDMA peripheral base address.

Returns

The error status for all channels. Check the relevant bits for specific channel.

bool SDMA_GetRequestSourceStatus(SDMAARM_Type *base, uint32_t source)

Gets the SDMA request source pending status.

Parameters

- base – SDMA peripheral base address.
- source – DMA request source number.

Returns

True means the request source is pending, otherwise not pending.

void SDMA_CreateHandle(*sdma_handle_t* *handle, SDMAARM_Type *base, uint32_t channel, *sdma_context_data_t* *context)

Creates the SDMA handle.

This function is called if using the transactional API for SDMA. This function initializes the internal state of the SDMA handle.

Parameters

- handle – SDMA handle pointer. The SDMA handle stores callback function and parameters.
- base – SDMA peripheral base address.
- channel – SDMA channel number.

- context – Context structure for the channel to download into SDMA. Users shall make sure the context located in a non-cacheable memory, or it will cause SDMA run fail. Users shall not touch the context contents, it only be filled by SDMA driver in SDMA_SubmitTransfer function.

void SDMA_InstallBDMemory(*sdma_handle_t* *handle, *sdma_buffer_descriptor_t* *BDPool, uint32_t BDCount)

Installs the BDs memory pool into the SDMA handle.

This function is called after the SDMA_CreateHandle to use multi-buffer feature.

Parameters

- handle – SDMA handle pointer.
- BDPool – A memory pool to store BDs. It must be located in non-cacheable address.
- BDCount – The number of BD slots.

void SDMA_SetCallback(*sdma_handle_t* *handle, *sdma_callback* callback, void *userData)

Installs a callback function for the SDMA transfer.

This callback is called in the SDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

- handle – SDMA handle pointer.
- callback – SDMA callback function pointer.
- userData – A parameter for the callback function.

void SDMA_SetMultiFifoConfig(*sdma_transfer_config_t* *config, uint32_t fifoNums, uint32_t fifoOffset)

multi fifo configurations.

This api is used to support multi fifo for SDMA, if user want to get multi fifo data, then this api shoule be called before submit transfer.

Parameters

- config – transfer configurations.
- fifoNums – fifo numbers that multi fifo operation perform, support up to 15 fifo numbers.
- fifoOffset – fifoOffset = fifo address offset / sizeof(uint32_t) - 1.

void SDMA_EnableSwDone(SDMAARM_Type *base, *sdma_transfer_config_t* *config, uint8_t sel, *sdma_peripheral_t* type)

enable sdma sw done feature.

Deprecated:

Do not use this function. It has been superceded by SDMA_SetDoneConfig.

Parameters

- base – SDMA base.
- config – transfer configurations.
- sel – sw done selector.
- type – peripheral type is used to determine the corresponding peripheral sw done selector bit.

void SDMA_SetDoneConfig(SDMAARM_Type *base, *sdma_transfer_config_t* *config, *sdma_peripheral_t* type, *sdma_done_src_t* doneSrc)

sdma channel done configurations.

Parameters

- base – SDMA base.
- config – transfer configurations.
- type – peripheral type.
- doneSrc – reference sdma_done_src_t.

void SDMA_LoadScript(SDMAARM_Type *base, uint32_t destAddr, void *srcAddr, size_t bufferSizeBytes)

load script to sdma program memory.

Parameters

- base – SDMA base.
- destAddr – dest script address, should be SDMA program memory address.
- srcAddr – source address of target script.
- bufferSizeBytes – bytes size of script.

void SDMA_DumpScript(SDMAARM_Type *base, uint32_t srcAddr, void *destAddr, size_t bufferSizeBytes)

dump script from sdma program memory.

Parameters

- base – SDMA base.
- srcAddr – should be SDMA program memory address.
- destAddr – address to store scripts.
- bufferSizeBytes – bytes size of script.

static inline const char *SDMA_GetRamScriptVersion(SDMAARM_Type *base)

Get RAM script version.

Parameters

- base – SDMA base.

Returns

The script version of RAM.

void SDMA_PrepareTransfer(*sdma_transfer_config_t* *config, uint32_t srcAddr, uint32_t destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferSize, uint32_t eventSource, *sdma_peripheral_t* peripheral, *sdma_transfer_type_t* type)

Prepares the SDMA 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.

Parameters

- config – The user configuration structure of type sdma_transfer_t.
- srcAddr – SDMA transfer source address.

- destAddr – SDMA transfer destination address.
- srcWidth – SDMA transfer source address width(bytes).
- destWidth – SDMA transfer destination address width(bytes).
- bytesEachRequest – SDMA transfer bytes per channel request.
- transferSize – SDMA transfer bytes to be transferred.
- eventSource – Event source number for the transfer, if use software trigger, just write 0.
- peripheral – Peripheral type, used to decide if need to use some special scripts.
- type – SDMA transfer type. Used to decide the correct SDMA script address in SDMA ROM.

void SDMA_PrepareP2PTransfer(*sdma_transfer_config_t* *config, uint32_t srcAddr, uint32_t destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferSize, uint32_t eventSource, uint32_t eventSource1, *sdma_peripheral_t* peripheral, *sdma_p2p_config_t* *p2p)

Prepares the SDMA P2P 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.

Parameters

- config – The user configuration structure of type sdma_transfer_t.
- srcAddr – SDMA transfer source address.
- destAddr – SDMA transfer destination address.
- srcWidth – SDMA transfer source address width(bytes).
- destWidth – SDMA transfer destination address width(bytes).
- bytesEachRequest – SDMA transfer bytes per channel request.
- transferSize – SDMA transfer bytes to be transferred.
- eventSource – Event source number for the transfer.
- eventSource1 – Event source1 number for the transfer.
- peripheral – Peripheral type, used to decide if need to use some special scripts.
- p2p – sdma p2p configuration pointer.

void SDMA_SubmitTransfer(*sdma_handle_t* *handle, const *sdma_transfer_config_t* *config)

Submits the SDMA transfer request.

This function submits the SDMA transfer request according to the transfer configuration structure.

Parameters

- handle – SDMA handle pointer.
- config – Pointer to SDMA transfer configuration structure.

void SDMA_StartTransfer(*sdma_handle_t* *handle)

SDMA 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 – SDMA handle pointer.

void SDMA_StopTransfer(*sdma_handle_t* *handle)

SDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call SDMA_StartTransfer() again to resume the transfer.

Parameters

- handle – SDMA handle pointer.

void SDMA_AbortTransfer(*sdma_handle_t* *handle)

SDMA 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.

uint32_t SDMA_GetTransferredBytes(*sdma_handle_t* *handle)

Get transferred bytes while not using BD pools.

This function returns the buffer descriptor count value if not using buffer descriptor. While do a simple transfer, which only uses one descriptor, the SDMA driver inside handle the buffer descriptor. In uart receive case, it can tell users how many data already received, also it can tells users how many data transfferd while error occurred. Notice, the count would not change while transfer is on-going using default SDMA script.

Parameters

- handle – DMA handle pointer.

Returns

Transferred bytes.

bool SDMA_IsPeripheralInSPBA(uint32_t addr)

Judge if address located in SPBA.

Parameters

- addr – Address which need to judge.

Return values

True – means located in SPBA, false means not.

void SDMA_HandleIRQ(*sdma_handle_t* *handle)

SDMA IRQ handler for complete a buffer descriptor transfer.

This function clears the interrupt flags and also handle the CCB for the channel.

Parameters

- handle – SDMA handle pointer.

FSL_SDMA_DRIVER_VERSION

SDMA driver version.

Version 2.4.3.

enum _sdma_transfer_size

SDMA transfer configuration.

Values:

enumerator kSDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kSDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kSDMA_TransferSize3Bytes

Source/Destination data transfer size is 3 bytes every time

enumerator kSDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enum _sdma_bd_status

SDMA buffer descriptor status.

Values:

enumerator kSDMA_BDStatusDone

BD ownership, 0 means ARM core owns the BD, while 1 means SDMA owns BD.

enumerator kSDMA_BDStatusWrap

While this BD is last one, the next BD will be the first one

enumerator kSDMA_BDStatusContinuous

Buffer is allowed to transfer/receive to/from multiple buffers

enumerator kSDMA_BDStatusInterrupt

While this BD finished, send an interrupt.

enumerator kSDMA_BDStatusError

Error occurred on buffer descriptor command.

enumerator kSDMA_BDStatusLast

This BD is the last BD in this array. It means the transfer ended after this buffer

enumerator kSDMA_BDStatusExtend

Buffer descriptor extend status for SDMA scripts

enum _sdma_bd_command

SDMA buffer descriptor command.

Values:

enumerator kSDMA_BDCommandSETDM

Load SDMA data memory from ARM core memory buffer.

enumerator kSDMA_BDCommandGETDM

Copy SDMA data memory to ARM core memory buffer.

enumerator kSDMA_BDCommandSETPM

Load SDMA program memory from ARM core memory buffer.

enumerator kSDMA_BDCommandGETPM

Copy SDMA program memory to ARM core memory buffer.

enumerator kSDMA_BDCommandSETCTX

Load context for one channel into SDMA RAM from ARM platform memory buffer.

enumerator kSDMA_BDCommandGETCTX
Copy context for one channel from SDMA RAM to ARM platform memory buffer.

enum _sdma_context_switch_mode
SDMA context switch mode.

Values:

enumerator kSDMA_ContextSwitchModeStatic
SDMA context switch mode static

enumerator kSDMA_ContextSwitchModeDynamicLowPower
SDMA context switch mode dynamic with low power

enumerator kSDMA_ContextSwitchModeDynamicWithNoLoop
SDMA context switch mode dynamic with no loop

enumerator kSDMA_ContextSwitchModeDynamic
SDMA context switch mode dynamic

enum _sdma_clock_ratio
SDMA core clock frequency ratio to the ARM DMA interface.

Values:

enumerator kSDMA_HalfARMClockFreq
SDMA core clock frequency half of ARM platform

enumerator kSDMA_ARMClockFreq
SDMA core clock frequency equals to ARM platform

enum _sdma_transfer_type
SDMA transfer type.

Values:

enumerator kSDMA_MemoryToMemory
Transfer from memory to memory

enumerator kSDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kSDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kSDMA_PeripheralToPeripheral
Transfer from peripheral to peripheral

enum sdma_peripheral
Peripheral type use SDMA.

Values:

enumerator kSDMA_PeripheralTypeMemory
Peripheral DDR memory

enumerator kSDMA_PeripheralTypeUART
UART use SDMA

enumerator kSDMA_PeripheralTypeUART_SP
UART instance in SPBA use SDMA

enumerator kSDMA_PeripheralTypeSPDIF
SPDIF use SDMA

enumerator kSDMA_PeripheralNormal
Normal peripheral use SDMA

enumerator kSDMA_PeripheralNormal_SP
Normal peripheral in SPBA use SDMA

enumerator kSDMA_PeripheralMultiFifoPDM
multi fifo PDM

enumerator kSDMA_PeripheralMultiFifoSaiRX
multi fifo sai rx use SDMA

enumerator kSDMA_PeripheralMultiFifoSaiTX
multi fifo sai tx use SDMA

enumerator kSDMA_PeripheralASRCM2P
asrc m2p

enumerator kSDMA_PeripheralASRCP2M
asrc p2m

enumerator kSDMA_PeripheralASRCP2P
asrc p2p

_sdma_transfer_status SDMA transfer status

Values:

enumerator kStatus_SDMA_ERROR
SDMA context error.

enumerator kStatus_SDMA_Busy
Channel is busy and can't handle the transfer request.

_sdma_multi_fifo_mask SDMA multi fifo mask

Values:

enumerator kSDMA_MultiFifoWatermarkLevelMask
multi fifo watermark level mask

enumerator kSDMA_MultiFifoNumsMask
multi fifo nums mask

enumerator kSDMA_MultiFifoOffsetMask
multi fifo offset mask

enumerator kSDMA_MultiFifoSwDoneMask
multi fifo sw done mask

enumerator kSDMA_MultiFifoSwDoneSelectorMask
multi fifo sw done selector mask

_sdma_multi_fifo_shift SDMA multi fifo shift

Values:

enumerator kSDMA_MultiFifoWatermarkLevelShift
multi fifo watermark level shift

enumerator kSDMA_MultiFifoNumsShift
multi fifo nums shift

enumerator kSDMA_MultiFifoOffsetShift
multi fifo offset shift

enumerator kSDMA_MultiFifoSwDoneShift
multi fifo sw done shift

enumerator kSDMA_MultiFifoSwDoneSelectorShift
multi fifo sw done selector shift

_sdma_done_channel SDMA done channel

Values:

enumerator kSDMA_DoneChannel0
SDMA done channel 0

enumerator kSDMA_DoneChannel1
SDMA done channel 1

enumerator kSDMA_DoneChannel2
SDMA done channel 2

enumerator kSDMA_DoneChannel3
SDMA done channel 3

enumerator kSDMA_DoneChannel4
SDMA done channel 4

enumerator kSDMA_DoneChannel5
SDMA done channel 5

enumerator kSDMA_DoneChannel6
SDMA done channel 6

enumerator kSDMA_DoneChannel7
SDMA done channel 7

enum _sdma_done_src
SDMA done source.

Values:

enumerator kSDMA_DoneSrcSW
software done

enumerator kSDMA_DoneSrcHwEvent0U
HW event 0 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent1U
HW event 1 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent2U
HW event 2 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent3U
HW event 3 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent4U
HW event 4 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent5U
HW event 5 is used for DONE event

enumerator kSDMA_DoneSrCHwEvent6U
HW event 6 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent7U
HW event 7 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent8U
HW event 8 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent9U
HW event 9 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent10U
HW event 10 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent11U
HW event 11 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent12U
HW event 12 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent13U
HW event 13 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent14U
HW event 14 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent15U
HW event 15 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent16U
HW event 16 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent17U
HW event 17 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent18U
HW event 18 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent19U
HW event 19 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent20U
HW event 20 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent21U
HW event 21 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent22U
HW event 22 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent23U
HW event 23 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent24U
HW event 24 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent25U
HW event 25 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent26U
HW event 26 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent27U
HW event 27 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent28U
HW event 28 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent29U
HW event 29 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent30U
HW event 30 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent31U
HW event 31 is used for DONE event

typedef enum *_sdma_transfer_size* sdma_transfer_size_t
SDMA transfer configuration.

typedef enum *_sdma_bd_status* sdma_bd_status_t
SDMA buffer descriptor status.

typedef enum *_sdma_bd_command* sdma_bd_command_t
SDMA buffer descriptor command.

typedef enum *_sdma_context_switch_mode* sdma_context_switch_mode_t
SDMA context switch mode.

typedef enum *_sdma_clock_ratio* sdma_clock_ratio_t
SDMA core clock frequency ratio to the ARM DMA interface.

typedef enum *_sdma_transfer_type* sdma_transfer_type_t
SDMA transfer type.

typedef enum *sdma_peripheral* sdma_peripheral_t
Peripheral type use SDMA.

typedef enum *_sdma_done_src* sdma_done_src_t
SDMA done source.

typedef struct *_sdma_config* sdma_config_t
SDMA global configuration structure.

typedef struct *_sdma_multi_fifo_config* sdma_multi_fifo_config_t
SDMA multi fifo configurations.

typedef struct *_sdma_sw_done_config* sdma_sw_done_config_t
SDMA sw done configurations.

typedef struct *_sdma_p2p_config* sdma_p2p_config_t
SDMA peripheral to peripheral R7 config.

typedef struct *_sdma_transfer_config* sdma_transfer_config_t
SDMA transfer configuration.

This structure configures the source/destination transfer attribute.

typedef struct *_sdma_buffer_descriptor* sdma_buffer_descriptor_t
SDMA buffer descriptor structure.

This structure is a buffer descriptor, this structure describes the buffer start address and other options

typedef struct *_sdma_channel_control* sdma_channel_control_t

SDMA channel control descriptor structure.

typedef struct *_sdma_context_data* sdma_context_data_t

SDMA context structure for each channel. This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

typedef void (*sdma_callback)(struct *_sdma_handle* *handle, void *userData, bool transferDone, uint32_t bdIndex)

Define callback function for SDMA.

typedef struct *_sdma_handle* sdma_handle_t

SDMA transfer handle structure.

SDMA_DRIVER_LOAD_RAM_SCRIPT

struct _sdma_config

#include <fsl_sdma.h> SDMA global configuration structure.

Public Members

bool enableRealTimeDebugPin

If enable real-time debug pin, default is closed to reduce power consumption.

bool isSoftwareResetClearLock

If software reset clears the LOCK bit which prevent writing SDMA scripts into SDMA.

sdma_clock_ratio_t ratio

SDMA core clock ratio to ARM platform DMA interface

struct _sdma_multi_fifo_config

#include <fsl_sdma.h> SDMA multi fifo configurations.

Public Members

uint8_t fifoNums

fifo numbers

uint8_t fifoOffset

offset between multi fifo data register address

struct _sdma_sw_done_config

#include <fsl_sdma.h> SDMA sw done configurations.

Public Members

bool enableSwDone

true is enable sw done, false is disable

uint8_t swDoneSel

sw done channel number per peripheral type

struct _sdma_p2p_config

#include <fsl_sdma.h> SDMA peripheral to peripheral R7 config.

Public Members

uint8_t sourceWatermark
: lower watermark value

uint8_t destWatermark
: higher water makr value

bool continuousTransfer
: 0: the amount of samples to be transferred is equal to the cont field of mode word 1: the amount of samples to be transferred is unknown and script will keep on transferring as long as both events are detected and script must be stopped by application.

struct __sdma__transfer__config

#include <fsl_sdma.h> SDMA transfer configuration.

This structure configures the source/destination transfer attribute.

Public Members

uint32_t srcAddr
: Source address of the transfer

uint32_t destAddr
: Destination address of the transfer

sdma_transfer_size_t srcTransferSize
: Source data transfer size.

sdma_transfer_size_t destTransferSize
: Destination data transfer size.

uint32_t bytesPerRequest
: Bytes to transfer in a minor loop

uint32_t transferSzie
: Bytes to transfer for this descriptor

uint32_t scriptAddr
: SDMA script address located in SDMA ROM.

uint32_t eventSource
: Event source number for the channel. 0 means no event, use software trigger

uint32_t eventSource1
: event source 1

bool isEventIgnore
: True means software trigger, false means hardware trigger

bool isSoftTriggerIgnore
: If ignore the HE bit, 1 means use hardware events trigger, 0 means software trigger

sdma_transfer_type_t type
: Transfer type, transfer type used to decide the SDMA script.

sdma_multi_fifo_config_t multiFifo
: multi fifo configurations

sdma_sw_done_config_t swDone
: sw done selector

uint32_t watermarkLevel
watermark level

uint32_t eventMask0
event mask 0

uint32_t eventMask1
event mask 1

struct __sdma__buffer__descriptor
#include <fsl_sdma.h> SDMA buffer descriptor structure.

This structure is a buffer descriptor, this structure describes the buffer start address and other options

Public Members

uint32_t count
Bytes of the buffer length for this buffer descriptor.

uint32_t status
E,R,I,C,W,D status bits stored here

uint32_t command
command mostlky used for channel 0

uint32_t bufferAddr
Buffer start address for this descriptor.

uint32_t extendBufferAddr
External buffer start address, this is an optional for a transfer.

struct __sdma__channel__control
#include <fsl_sdma.h> SDMA channel control descriptor structure.

Public Members

uint32_t currentBDAddr
Address of current buffer descriptor processed

uint32_t baseBDAddr
The start address of the buffer descriptor array

uint32_t channelDesc
Optional for transfer

uint32_t status
Channel status

struct __sdma__context__data
#include <fsl_sdma.h> SDMA context structure for each channel. This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

Public Members

uint32_t GeneralReg[8]
8 general regsiters used for SDMA RISC core

struct __sdma__handle
#include <fsl_sdma.h> SDMA transfer handle structure.

Public Members

sdma_callback callback
Callback function for major count exhausted.

void *userData
Callback function parameter.

SDMAARM_Type *base
SDMA peripheral base address.

sdma_buffer_descriptor_t *BDPool
Pointer to memory stored BD arrays.

uint32_t bdCount
How many buffer descriptor

uint32_t bdIndex
How many buffer descriptor

uint32_t eventSource
Event source count for the channel

uint32_t eventSource1
Event source 1 count for the channel

sdma_context_data_t *context
Channel context to exectute in SDMA

uint8_t channel
SDMA channel number.

uint8_t priority
SDMA channel priority

uint8_t flags
The status of the current channel.

2.31 SEMA4: Hardware Semaphores Driver

FSL_SEMA4_DRIVER_VERSION
SEMA4 driver version.

void SEMA4_Init(SEMA4_Type *base)
Initializes the SEMA4 module.

This function initializes the SEMA4 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA4_ResetGate or SEMA4_ResetAllGates function.

Parameters

- base – SEMA4 peripheral base address.

void SEMA4_Deinit(SEMA4_Type *base)
De-initializes the SEMA4 module.

This function de-initializes the SEMA4 module. It only disables the clock.

Parameters

- base – SEMA4 peripheral base address.

status_t SEMA4_TryLock(SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)

Tries to lock the SEMA4 gate.

This function tries to lock the specific SEMA4 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema4 gate successfully.
- kStatus_Fail – Sema4 gate has been locked by another processor.

status_t SEMA4_Lock(SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA4 gate.

This function locks the specific SEMA4 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA4_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns

status_t

static inline void SEMA4_Unlock(SEMA4_Type *base, uint8_t gateNum)

Unlocks the SEMA4 gate.

This function unlocks the specific SEMA4 gate. It only writes unlock value to the SEMA4 gate register. However, it does not check whether the SEMA4 gate is locked by the current processor or not. As a result, if the SEMA4 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to unlock.

static inline int32_t SEMA4_GetLockProc(SEMA4_Type *base, uint8_t gateNum)

Gets the status of the SEMA4 gate.

This function checks the lock status of a specific SEMA4 gate.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is unlocked, otherwise return the processor number which has locked the gate.

status_t SEMA4_ResetGate(SEMA4_Type *base, uint8_t gateNum)

Resets the SEMA4 gate to an unlocked status.

This function resets a SEMA4 gate to an unlocked status.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – SEMA4 gate is reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

static inline *status_t* SEMA4_ResetAllGates(SEMA4_Type *base)

Resets all SEMA4 gates to an unlocked status.

This function resets all SEMA4 gate to an unlocked status.

Parameters

- base – SEMA4 peripheral base address.

Return values

- kStatus_Success – SEMA4 is reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

static inline void SEMA4_EnableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum, uint16_t mask)

Enable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1.

static inline void SEMA4_DisableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum, uint16_t mask)

Disable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1.

static inline uint32_t SEMA4_GetGateNotifyStatus(SEMA4_Type *base, uint8_t procNum)

Get the gate notification flags.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle. The status flags are cleared automatically when the gate is locked by current core or locked again before the other core.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.

Returns

OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1 flags are pending.

status_t SEMA4_ResetGateNotify(SEMA4_Type *base, uint8_t gateNum)

Resets the SEMA4 gate IRQ notification.

This function resets a SEMA4 gate IRQ notification.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – Reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

static inline *status_t* SEMA4_ResetAllGateNotify(SEMA4_Type *base)

Resets all SEMA4 gates IRQ notification.

This function resets all SEMA4 gate IRQ notifications.

Parameters

- base – SEMA4 peripheral base address.

Return values

- kStatus_Success – Reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

SEMA4_GATE_NUM_RESET_ALL

The number to reset all SEMA4 gates.

SEMA4_GATEn(base, n)

SEMA4 gate n register address.

SEMA4_BUSY_POLL_COUNT

Maximum polling iterations for SEMA4 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA4 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA4 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.32 SNVS: Secure Non-Volatile Storage

2.33 Secure Non-Volatile Storage High-Power

void SNVS_HP_Init(SNVS_Type *base)

Initialize the SNVS.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_Deinit(SNVS_Type *base)

Deinitialize the SNVS.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_RTC_Init(SNVS_Type *base, const *snvs_hp_rtc_config_t* *config)

Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address
- config – Pointer to the user's SNVS configuration structure.

void SNVS_HP_RTC_Deinit(SNVS_Type *base)

Stops the RTC and SRTC timers.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_RTC_GetDefaultConfig(*snvs_hp_rtc_config_t* *config)

Fills in the SNVS config struct with the default settings.

The default values are as follows.

```
config->rtccalenable = false;
config->rtccalvalue = 0U;
config->PIFreq = 0U;
```

Parameters

- config – Pointer to the user's SNVS configuration structure.

status_t SNVS_HP_RTC_SetDatetime(SNVS_Type *base, const *snvs_hp_rtc_datetime_t* *datetime)

Sets the SNVS RTC date and time according to the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the SNVS RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void SNVS_HP_RTC_GetDatetime(SNVS_Type *base, *snvs_hp_rtc_datetime_t* *datetime)

Gets the SNVS RTC time and stores it in the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t SNVS_HP_RTC_SetAlarm(SNVS_Type *base, const *snvs_hp_rtc_datetime_t* *alarmTime)

Sets the SNVS RTC alarm time.

The function sets the RTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

- base – SNVS peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the SNVS RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

void SNVS_HP_RTC_GetAlarm(SNVS_Type *base, *snvs_hp_rtc_datetime_t* *datetime)

Returns the SNVS RTC alarm time.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

static inline void SNVS_HP_RTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)

Enables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_hp_interrupts_t

static inline void SNVS_HP_RTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)

Disables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration :: _snvs_hp_interrupts_t

uint32_t SNVS_HP_RTC_GetEnabledInterrupts(SNVS_Type *base)

Gets the enabled SNVS interrupts.

Parameters

- base – SNVS peripheral base address

Returns
The enabled interrupts. This is the logical OR of members of the enumeration :: _snvs_hp_interrupts_t

uint32_t SNVS_HP_RTC_GetStatusFlags(SNVS_Type *base)

Gets the SNVS status flags.

Parameters

- base – SNVS peripheral base address

Returns
The status flags. This is the logical OR of members of the enumeration :: _snvs_hp_status_flags_t

static inline void SNVS_HP_RTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clears the SNVS status flags.

Parameters

- base – SNVS peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration :: _snvs_hp_status_flags_t

static inline void SNVS_HP_RTC_StartTimer(SNVS_Type *base)

Starts the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_RTC_StopTimer(SNVS_Type *base)

Stops the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_EnableHighAssuranceCounter(SNVS_Type *base, bool enable)

Enable or disable the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_HP_StartHighAssuranceCounter(SNVS_Type *base, bool start)

Start or stop the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- start – Pass true to start, false to stop.

static inline void SNVS_HP_SetHighAssuranceCounterInitialValue(SNVS_Type *base, uint32_t value)

Set the High Assurance Counter (HAC) initialize value.

Parameters

- base – SNVS peripheral base address
- value – The initial value to set.

static inline void SNVS_HP_LoadHighAssuranceCounter(SNVS_Type *base)

Load the High Assurance Counter (HAC)

This function loads the HAC initialize value to counter register.

Parameters

- base – SNVS peripheral base address

static inline uint32_t SNVS_HP_GetHighAssuranceCounter(SNVS_Type *base)

Get the current High Assurance Counter (HAC) value.

Parameters

- base – SNVS peripheral base address

Returns

HAC currnet value.

static inline void SNVS_HP_ClearHighAssuranceCounter(SNVS_Type *base)

Clear the High Assurance Counter (HAC)

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_LockHighAssuranceCounter(SNVS_Type *base)

Lock the High Assurance Counter (HAC)

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

Parameters

- base – SNVS peripheral base address

FSL_SNVS_HP_DRIVER_VERSION

Version 2.3.2

enum _snvs_hp_interrupts

List of SNVS interrupts.

Values:

enumerator kSNVS_RTC_AlarmInterrupt

RTC time alarm

enumerator kSNVS_RTC_PeriodicInterrupt

RTC periodic interrupt

enum _snvs_hp_status_flags

List of SNVS flags.

Values:

enumerator kSNVS_RTC_AlarmInterruptFlag

RTC time alarm flag

enumerator kSNVS_RTC_PeriodicInterruptFlag
RTC periodic interrupt flag

enumerator kSNVS_ZMK_ZeroFlag
The ZMK is zero

enumerator kSNVS_OTPMK_ZeroFlag
The OTPMK is zero

enum _snvs_hp_sv_status_flags
List of SNVS security violation flags.

Values:

enumerator kSNVS_LP_ViolationFlag
Low Power section Security Violation

enumerator kSNVS_ZMK_EccFailFlag
Zeroizable Master Key Error Correcting Code Check Failure

enumerator kSNVS_LP_SoftwareViolationFlag
LP Software Security Violation

enumerator kSNVS_FatalSoftwareViolationFlag
Software Fatal Security Violation

enumerator kSNVS_SoftwareViolationFlag
Software Security Violation

enumerator kSNVS_Violation0Flag
Security Violation 0

enumerator kSNVS_Violation1Flag
Security Violation 1

enumerator kSNVS_Violation2Flag
Security Violation 2

enumerator kSNVS_Violation4Flag
Security Violation 4

enumerator kSNVS_Violation5Flag
Security Violation 5

enum _snvs_hp_ssm_state
List of SNVS Security State Machine State.

Values:

enumerator kSNVS_SSMInit
Init

enumerator kSNVS_SSMHardFail
Hard Fail

enumerator kSNVS_SSMSoftFail
Soft Fail

enumerator kSNVS_SSMInitInter
Init Intermediate (transition state between Init and Check)

enumerator kSNVS_SSMCheck
Check

enumerator kSNVS_SSMNonSecure
Non-Secure

enumerator kSNVS_SSMTrusted
Trusted

enumerator kSNVS_SSMSecure
Secure

typedef enum *_snvs_hp_interrupts* snvs_hp_interrupts_t
List of SNVS interrupts.

typedef enum *_snvs_hp_status_flags* snvs_hp_status_flags_t
List of SNVS flags.

typedef enum *_snvs_hp_sv_status_flags* snvs_hp_sv_status_flags_t
List of SNVS security violation flags.

typedef struct *_snvs_hp_rtc_datetime* snvs_hp_rtc_datetime_t
Structure is used to hold the date and time.

typedef struct *_snvs_hp_rtc_config* snvs_hp_rtc_config_t
SNVS config structure.

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

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

typedef enum *_snvs_hp_ssm_state* snvs_hp_ssm_state_t
List of SNVS Security State Machine State.

static inline void SNVS_HP_EnableMasterKeySelection(SNVS_Type *base, bool enable)
Enable or disable master key selection.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_HP_ProgramZeroizableMasterKey(SNVS_Type *base)
Trigger to program Zeroizable Master Key.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_ChangeSSMState(SNVS_Type *base)
Trigger SSM State Transition.

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State –> Trusted (when Secure Boot or in Fab Configuration)
- Trusted State –> Secure
- Secure State –> Trusted
- Soft Fail –> Non-Secure

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_SetSoftwareFatalSecurityViolation(SNVS_Type *base)

Trigger Software Fatal Security Violation.

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_SetSoftwareSecurityViolation(SNVS_Type *base)

Trigger Software Security Violation.

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline *snvs_hp_ssm_state_t* SNVS_HP_GetSSMState(SNVS_Type *base)

Get current SSM State.

Parameters

- base – SNVS peripheral base address

Returns

Current SSM state

static inline void SNVS_HP_ResetLP(SNVS_Type *base)

Reset the SNVS LP section.

Reset the LP section except SRTC and Time alarm.

Parameters

- base – SNVS peripheral base address

static inline uint32_t SNVS_HP_GetStatusFlags(SNVS_Type *base)

Get the SNVS HP status flags.

The flags are returned as the OR'ed value f the enumeration :: _snvs_hp_status_flags_t.

Parameters

- base – SNVS peripheral base address

Returns

The OR'ed value of status flags.

static inline void SNVS_HP_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: _snvs_hp_status_flags_t. Only these flags could be cleared using this API.

- kSNVS_RTC_PeriodicInterruptFlag

- kSNVS_RTC_AlarmInterruptFlag

Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

static inline uint32_t SNVS_HP_GetSecurityViolationStatusFlags(SNVS_Type *base)

Get the SNVS HP security violation status flags.

The flags are returned as the OR'ed value of the enumeration :: _snvs_hp_sv_status_flags_t.

Parameters

- base – SNVS peripheral base address

Returns

The OR'ed value of security violation status flags.

static inline void SNVS_HP_ClearSecurityViolationStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP security violation status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: _snvs_hp_sv_status_flags_t. Only these flags could be cleared using this API.

- kSNVS_ZMK_EccFailFlag
- kSNVS_Violation0Flag
- kSNVS_Violation1Flag
- kSNVS_Violation2Flag
- kSNVS_Violation3Flag
- kSNVS_Violation4Flag
- kSNVS_Violation5Flag

Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

SNVS_HPSVSR_SV0_MASK

SNVS_HPSVSR_SV1_MASK

SNVS_HPSVSR_SV2_MASK

SNVS_HPSVSR_SV4_MASK

SNVS_HPSVSR_SV5_MASK

SNVS_MAKE_HP_SV_FLAG(x)

Macro to make security violation flag.

Macro help to make security violation flag kSNVS_Violation0Flag to kSNVS_Violation5Flag, For example, SNVS_MAKE_HP_SV_FLAG(0) is kSNVS_Violation0Flag.

struct _snvs_hp_rtc_datetime

#include <fsl_snvs_hp.h> Structure is used to hold the date and time.

Public Members

uint16_t year
: Range from 1970 to 2099.

uint8_t month
: Range from 1 to 12.

uint8_t day
: Range from 1 to 31 (depending on month).

uint8_t hour
: Range from 0 to 23.

uint8_t minute
: Range from 0 to 59.

uint8_t second
: Range from 0 to 59.

struct _snvs_hp_rtc_config
: *#include <fsl_snvs_hp.h>* SNVS config structure.

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

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

Public Members

bool rtcCalEnable
: true: RTC calibration mechanism is enabled; false:No calibration is used

uint32_t rtcCalValue
: Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15

uint32_t periodicInterruptFreq
: Defines frequency of the periodic interrupt; Range from 0 to 15

2.34 Secure Non-Volatile Storage Low-Power

void SNVS_LP_Init(SNVS_Type *base)
: Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_Deinit(SNVS_Type *base)
: Deinit the SNVS LP section.

Parameters

- base – SNVS peripheral base address

status_t SNVS_LP_SRTC_SetDatetime(SNVS_Type *base, const *snvs_lp_srtc_datetime_t* *datetime)

Sets the SNVS SRTC date and time according to the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the SNVS SRTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void SNVS_LP_SRTC_GetDatetime(SNVS_Type *base, *snvs_lp_srtc_datetime_t* *datetime)

Gets the SNVS SRTC time and stores it in the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t SNVS_LP_SRTC_SetAlarm(SNVS_Type *base, const *snvs_lp_srtc_datetime_t* *alarmTime)

Sets the SNVS SRTC alarm time.

The function sets the SRTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error. Please note, that SRTC alarm has limited resolution because only 32 most significant bits of SRTC counter are compared to SRTC Alarm register. If the alarm time is beyond SRTC resolution, the function does not set the alarm and returns an error.

Parameters

- base – SNVS peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the SNVS SRTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed or is beyond resolution

void SNVS_LP_SRTC_GetAlarm(SNVS_Type *base, *snvs_lp_srtc_datetime_t* *datetime)

Returns the SNVS SRTC alarm time.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

static inline void SNVS_LP_SRTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)

Enables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

static inline void SNVS_LP_SRTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)

Disables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

uint32_t SNVS_LP_SRTC_GetEnabledInterrupts(SNVS_Type *base)

Gets the enabled SNVS interrupts.

Parameters

- base – SNVS peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

uint32_t SNVS_LP_SRTC_GetStatusFlags(SNVS_Type *base)

Gets the SNVS status flags.

Parameters

- base – SNVS peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration :: _snvs_lp_srtc_status_flags

static inline void SNVS_LP_SRTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clears the SNVS status flags.

Parameters

- base – SNVS peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_status_flags

static inline void SNVS_LP_SRTC_StartTimer(SNVS_Type *base)

Starts the SNVS SRTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_LP_SRTC_StopTimer(SNVS_Type *base)

Stops the SNVS SRTC time counter.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_PassiveTamperPin_GetDefaultConfig(*snvs_lp_passive_tamper_t* *config)

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->polarity = 0U; config->filterenable = 0U; if available on SoC config->filter = 0U; if available on SoC endcode

Parameters

- config – Pointer to the user's SNVS configuration structure.

static inline void SNVS_LP_EnableMonotonicCounter(SNVS_Type *base, bool enable)

Enable or disable the Monotonic Counter.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

uint64_t SNVS_LP_GetMonotonicCounter(SNVS_Type *base)

Get the current Monotonic Counter.

Parameters

- base – SNVS peripheral base address

Returns

Current Monotonic Counter value.

static inline void SNVS_LP_IncreaseMonotonicCounter(SNVS_Type *base)

Increase the Monotonic Counter.

Increase the Monotonic Counter by 1.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_WriteZeroizableMasterKey(SNVS_Type *base, uint32_t ZMKey[8U])

Write Zeroizable Master Key (ZMK) to the SNVS registers.

Parameters

- base – SNVS peripheral base address
- ZMKey – The ZMK write to the SNVS register.

static inline void SNVS_LP_SetZeroizableMasterKeyValid(SNVS_Type *base, bool valid)

Set Zeroizable Master Key valid.

This API could only be called when using software programming mode. After writing ZMK using SNVS_LP_WriteZeroizableMasterKey, call this API to make the ZMK valid.

Parameters

- base – SNVS peripheral base address
- valid – Pass true to set valid, false to set invalid.

static inline bool SNVS_LP_GetZeroizableMasterKeyValid(SNVS_Type *base)

Get Zeroizable Master Key valid status.

In hardware programming mode, call this API to check whether the ZMK is valid.

Parameters

- base – SNVS peripheral base address

Returns

true if valid, false if invalid.

static inline void SNVS_LP_SetZeroizableMasterKeyProgramMode(SNVS_Type *base, *snvs_lp_zmk_program_mode_t* mode)

Set Zeroizable Master Key programming mode.

Parameters

- base – SNVS peripheral base address
- mode – ZMK programming mode.

static inline void SNVS_LP_EnableZeroizableMasterKeyECC(SNVS_Type *base, bool enable)

Enable or disable Zeroizable Master Key ECC.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_LP_SetMasterKeyMode(SNVS_Type *base, *snvs_lp_master_key_mode_t* mode)

Set SNVS Master Key mode.

Note: When kSNVS_ZMK or kSNVS_CMK used, the SNVS_HP must be configured to enable the master key selection.

Parameters

- base – SNVS peripheral base address
- mode – Master Key mode.

FSL_SNVS_LP_DRIVER_VERSION

Version 2.4.6

enum _snvs_lp_srtc_interrupts

List of SNVS_LP interrupts.

Values:

enumerator kSNVS_SRTC_AlarmInterrupt

SRTC time alarm.

enum _snvs_lp_srtc_status_flags

List of SNVS_LP flags.

Values:

enumerator kSNVS_SRTC_AlarmInterruptFlag

SRTC time alarm flag

enum _snvs_lp_external_tamper_status

List of SNVS_LP external tampers status.

Values:

enumerator kSNVS_TamperNotDetected

enumerator kSNVS_TamperDetected

enum _snvs_lp_external_tamper_polarity

SNVS_LP external tamper polarity.

Values:

enumerator kSNVS_ExternalTamperActiveLow

enumerator kSNVS_ExternalTamperActiveHigh

enum _snvs_lp_zmk_program_mode

SNVS_LP Zeroizable Master Key programming mode.

Values:

enumerator kSNVS_ZMKSoftwareProgram
Software programming mode.

enumerator kSNVS_ZMKHardwareProgram
Hardware programming mode.

enum _snvs_lp_master_key_mode
SNVS_LP Master Key mode.

Values:

enumerator kSNVS_OTPMK
One Time Programmable Master Key.

enumerator kSNVS_ZMK
Zeroizable Master Key.

enumerator kSNVS_CMK
Combined Master Key, it is XOR of OPTMK and ZMK.

typedef enum *_snvs_lp_srtc_interrupts* snvs_lp_srtc_interrupts_t
List of SNVS_LP interrupts.

typedef enum *_snvs_lp_srtc_status_flags* snvs_lp_srtc_status_flags_t
List of SNVS_LP flags.

typedef enum *_snvs_lp_external_tamper_status* snvs_lp_external_tamper_status_t
List of SNVS_LP external tampers status.

typedef enum *_snvs_lp_external_tamper_polarity* snvs_lp_external_tamper_polarity_t
SNVS_LP external tamper polarity.

typedef struct *_snvs_lp_srtc_datetime* snvs_lp_srtc_datetime_t
Structure is used to hold the date and time.

typedef struct *_snvs_lp_srtc_config* snvs_lp_srtc_config_t
SNVS_LP config structure.

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

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

typedef enum *_snvs_lp_zmk_program_mode* snvs_lp_zmk_program_mode_t
SNVS_LP Zeroizable Master Key programming mode.

typedef enum *_snvs_lp_master_key_mode* snvs_lp_master_key_mode_t
SNVS_LP Master Key mode.

void SNVS_LP_SRTC_Init(SNVS_Type *base, const *snvs_lp_srtc_config_t* *config)
Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address
- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_SRTC_Deinit(SNVS_Type *base)

Stops the SRTC timer.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_SRTC_GetDefaultConfig(*snvs_lp_srtc_config_t* *config)

Fills in the SNVS_LP config struct with the default settings.

The default values are as follows.

```
config->srtccalenable = false;
config->srtccalvalue = 0U;
```

Parameters

- config – Pointer to the user's SNVS configuration structure.

SNVS_ZMK_REG_COUNT

Define of SNVS_LP Zeroizable Master Key registers.

SNVS_LP_MAX_TAMPER

Define of SNVS_LP Max possible tamper.

struct snvs_lp_passive_tamper_t

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP passive tamper pins.

struct _snvs_lp_srtc_datetime

#include <fsl_snvs_lp.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _snvs_lp_srtc_config

#include <fsl_snvs_lp.h> SNVS_LP config structure.

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

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

Public Members

bool srtcCalEnable

true: SRTC calibration mechanism is enabled; false: No calibration is used

uint32_t srtcCalValue

Defines signed calibration value for SRTC; This is a 5-bit 2's complement value, range from -16 to +15

2.35 SPDIF: Sony/Philips Digital Interface

void SPDIF_Init(SPDIF_Type *base, const *spdif_config_t* *config)

Initializes the SPDIF peripheral.

Ungates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by SPDIF_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

- base – SPDIF base pointer
- config – SPDIF configuration structure.

void SPDIF_GetDefaultConfig(*spdif_config_t* *config)

Sets the SPDIF configuration structure to default values.

This API initializes the configuration structure for use in SPDIF_Init. The initialized structure can remain unchanged in SPDIF_Init, or it can be modified before calling SPDIF_Init. This is an example.

```
spdif_config_t config;
SPDIF_GetDefaultConfig(&config);
```

Parameters

- config – pointer to master configuration structure

void SPDIF_Deinit(SPDIF_Type *base)

De-initializes the SPDIF peripheral.

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

Parameters

- base – SPDIF base pointer

uint32_t SPDIF_GetInstance(SPDIF_Type *base)

Get the instance number for SPDIF.

Parameters

- base – SPDIF base pointer.

static inline void SPDIF_TxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Tx.

This function makes Tx FIFO in reset mode.

Parameters

- base – SPDIF base pointer

static inline void SPDIF_RxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Rx.

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

- base – SPDIF base pointer

void SPDIF_TxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Tx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Tx, false means disable.

static inline void SPDIF_RxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Rx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Rx, false means disable.

static inline uint32_t SPDIF_GetStatusFlag(SPDIF_Type *base)

Gets the SPDIF status flag state.

Parameters

- base – SPDIF base pointer

Returns

SPDIF status flag value. Use the _spdif_interrupt_enable_t to get the status value needed.

static inline void SPDIF_ClearStatusFlags(SPDIF_Type *base, uint32_t mask)

Clears the SPDIF status flag state.

Parameters

- base – SPDIF base pointer
- mask – State mask. It can be a combination of the _spdif_interrupt_enable_t member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits:
  - kSPDIF_UChannelReceiveRegisterFull
  - kSPDIF_QChannelReceiveRegisterFull
  - kSPDIF_TxFIFOEmpty
  - kSPDIF_RxFIFOFull

static inline void SPDIF_EnableInterrupts(SPDIF_Type *base, uint32_t mask)

Enables the SPDIF Tx interrupt requests.

Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF_WordStartInterruptEnable
  - kSPDIF_SyncErrorInterruptEnable
  - kSPDIF_FIFOWarningInterruptEnable
  - kSPDIF_FIFORequestInterruptEnable
  - kSPDIF_FIFOErrorInterruptEnable

static inline void SPDIF_DisableInterrupts(SPDIF_Type *base, uint32_t mask)

Disables the SPDIF Tx interrupt requests.

Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF_WordStartInterruptEnable
  - kSPDIF_SyncErrorInterruptEnable
  - kSPDIF_FIFOWarningInterruptEnable
  - kSPDIF_FIFORequestInterruptEnable
  - kSPDIF_FIFOErrorInterruptEnable

static inline void SPDIF_EnableDMA(SPDIF_Type *base, uint32_t mask, bool enable)

Enables/disables the SPDIF DMA requests.

Parameters

- base – SPDIF base pointer
- mask – SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined
  - kSPDIF_RxDMAEnable
  - kSPDIF_TxDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t SPDIF_TxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns

data register address.

static inline uint32_t SPDIF_TxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

static inline uint32_t SPDIF_RxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

static inline uint32_t SPDIF_RxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

void SPDIF_TxSetSampleRate(SPDIF_Type *base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz)

Configures the SPDIF Tx sample rate.

The audio format can be changed at run-time. This function configures the sample rate.

Parameters

- base – SPDIF base pointer.
- sampleRate_Hz – SPDIF sample rate frequency in Hz.
- sourceClockFreq_Hz – SPDIF tx clock source frequency in Hz.

uint32_t SPDIF_GetRxSampleRate(SPDIF_Type *base, uint32_t clockSourceFreq_Hz)

Configures the SPDIF Rx audio format.

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – SPDIF base pointer.
- clockSourceFreq_Hz – SPDIF system clock frequency in hz.

void SPDIF_WriteBlocking(SPDIF_Type *base, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SPDIF base pointer.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SPDIF_WriteLeftData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteRightData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusHigh(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusLow(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

void SPDIF_ReadBlocking(SPDIF_Type *base, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SPDIF base pointer.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SPDIF_ReadLeftData(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadRightData(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadChannelStatusHigh(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadChannelStatusLow(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadQChannel(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadUChannel(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

void SPDIF_TransferTxCreateHandle(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_callback_t* callback, void *userData)

Initializes the SPDIF Tx handle.

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

Parameters

- base – SPDIF base pointer
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SPDIF_TransferRxCreateHandle(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_callback_t* callback, void *userData)

Initializes the SPDIF Rx handle.

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

Parameters

- base – SPDIF base pointer.
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.

- userData – User parameter passed to the callback function.

status_t SPDIF_TransferSendNonBlocking(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- xfer – Pointer to the spdif_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SPDIF_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SPDIF_TransferReceiveNonBlocking(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- xfer – Pointer to the spdif_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SPDIF_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SPDIF_TransferGetSendCount(SPDIF_Type *base, *spdif_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

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

status_t SPDIF_TransferGetReceiveCount(SPDIF_Type *base, *spdif_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

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

void SPDIF_TransferAbortSend(SPDIF_Type *base, *spdif_handle_t* *handle)

Aborts the current send.

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

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.

void SPDIF_TransferAbortReceive(SPDIF_Type *base, *spdif_handle_t* *handle)

Aborts the current IRQ receive.

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

Parameters

- base – SPDIF base pointer
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.

void SPDIF_TransferTxHandleIRQ(SPDIF_Type *base, *spdif_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure.

void SPDIF_TransferRxHandleIRQ(SPDIF_Type *base, *spdif_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure.

FSL_SPDIF_DRIVER_VERSION
Version 2.0.7

SPDIF return status.

Values:

enumerator kStatus_SPDIF_RxDPLLLocked
SPDIF Rx PLL locked.

enumerator kStatus_SPDIF_TxFIFOError
SPDIF Tx FIFO error.

enumerator kStatus_SPDIF_TxFIFOResync
SPDIF Tx left and right FIFO resync.

enumerator kStatus_SPDIF_RxCnew
SPDIF Rx status channel value updated.

enumerator kStatus_SPDIF_ValidatyNoGood
SPDIF validaty flag not good.

enumerator kStatus_SPDIF_RxIllegalSymbol
SPDIF Rx receive illegal symbol.

enumerator kStatus_SPDIF_RxParityBitError
SPDIF Rx parity bit error.

enumerator kStatus_SPDIF_UChannelOverrun
SPDIF receive U channel overrun.

enumerator kStatus_SPDIF_QChannelOverrun
SPDIF receive Q channel overrun.

enumerator kStatus_SPDIF_UQChannelSync
SPDIF U/Q channel sync found.

enumerator kStatus_SPDIF_UQChannelFrameError
SPDIF U/Q channel frame error.

enumerator kStatus_SPDIF_RxFIFOError
SPDIF Rx FIFO error.

enumerator kStatus_SPDIF_RxFIFOResync
SPDIF Rx left and right FIFO resync.

enumerator kStatus_SPDIF_LockLoss
SPDIF Rx PLL clock lock loss.

enumerator kStatus_SPDIF_TxIdle
SPDIF Tx is idle

enumerator kStatus_SPDIF_RxIdle
SPDIF Rx is idle

enumerator kStatus_SPDIF_QueueFull
SPDIF queue full

enum _spdif_rxfull_select

SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

Values:

enumerator kSPDIF_RxFull1Sample

Rx full at least 1 sample in left and right FIFO

enumerator kSPDIF_RxFull4Samples

Rx full at least 4 sample in left and right FIFO

enumerator kSPDIF_RxFull8Samples

Rx full at least 8 sample in left and right FIFO

enumerator kSPDIF_RxFull16Samples

Rx full at least 16 sample in left and right FIFO

enum _spdif_txempty_select

SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

Values:

enumerator kSPDIF_TxEmpty0Sample

Tx empty at most 0 sample in left and right FIFO

enumerator kSPDIF_TxEmpty4Samples

Tx empty at most 4 sample in left and right FIFO

enumerator kSPDIF_TxEmpty8Samples

Tx empty at most 8 sample in left and right FIFO

enumerator kSPDIF_TxEmpty12Samples

Tx empty at most 12 sample in left and right FIFO

enum _spdif_uchannel_source

SPDIF U channel source.

Values:

enumerator kSPDIF_NoUChannel

No embedded U channel

enumerator kSPDIF_UChannelFromRx

U channel from receiver, it is CD mode

enumerator kSPDIF_UChannelFromTx

U channel from on chip tx

enum _spdif_gain_select

SPDIF clock gain.

Values:

enumerator kSPDIF_GAIN_24

Gain select is 24

enumerator kSPDIF_GAIN_16

Gain select is 16

enumerator kSPDIF_GAIN_12

Gain select is 12

enumerator kSPDIF_GAIN_8

Gain select is 8

enumerator kSPDIF_GAIN_6
Gain select is 6

enumerator kSPDIF_GAIN_4
Gain select is 4

enumerator kSPDIF_GAIN_3
Gain select is 3

enum _spdif_tx_source
SPDIF tx data source.

Values:

enumerator kSPDIF_txFromReceiver
Tx data directly through SPDIF receiver

enumerator kSPDIF_txNormal
Normal operation, data from processor

enum _spdif_validity_config
SPDIF tx data source.

Values:

enumerator kSPDIF_validityFlagAlwaysSet
Outgoing validity flags always set

enumerator kSPDIF_validityFlagAlwaysClear
Outgoing validity flags always clear

The SPDIF interrupt enable flag.

Values:

enumerator kSPDIF_RxDPLLLocked
SPDIF DPLL locked

enumerator kSPDIF_TxFIFOError
Tx FIFO underrun or overrun

enumerator kSPDIF_TxFIFOResync
Tx FIFO left and right channel resync

enumerator kSPDIF_RxControlChannelChange
SPDIF Rx control channel value changed

enumerator kSPDIF_ValidityFlagNoGood
SPDIF validity flag no good

enumerator kSPDIF_RxIllegalSymbol
SPDIF receiver found illegal symbol

enumerator kSPDIF_RxParityBitError
SPDIF receiver found parity bit error

enumerator kSPDIF_UChannelReceiveRegisterFull
SPDIF U channel revceive register full

enumerator kSPDIF_UChannelReceiveRegisterOverrun
SPDIF U channel receive register overrun

enumerator kSPDIF_QChannelReceiveRegisterFull
SPDIF Q channel receive reigster full

enumerator kSPDIF_QChannelReceiveRegisterOverrun
SPDIF Q channel receive register overrun

enumerator kSPDIF_UQChannelSync
SPDIF U/Q channel sync found

enumerator kSPDIF_UQChannelFrameError
SPDIF U/Q channel frame error

enumerator kSPDIF_RxFIFOError
SPDIF Rx FIFO underrun/overrun

enumerator kSPDIF_RxFIFOResync
SPDIF Rx left and right FIFO resync

enumerator kSPDIF_LockLoss
SPDIF receiver loss of lock

enumerator kSPDIF_TxFIFOEmpty
SPDIF Tx FIFO empty

enumerator kSPDIF_RxFIFOFull
SPDIF Rx FIFO full

enumerator kSPDIF_AllInterrupt
all interrupt

The DMA request sources.

Values:

enumerator kSPDIF_RxDMAEnable
Rx FIFO full

enumerator kSPDIF_TxDMAEnable
Tx FIFO empty

typedef enum *_spdif_rxfull_select* spdif_rxfull_select_t
SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

typedef enum *_spdif_txempty_select* spdif_txempty_select_t
SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

typedef enum *_spdif_uchannel_source* spdif_uchannel_source_t
SPDIF U channel source.

typedef enum *_spdif_gain_select* spdif_gain_select_t
SPDIF clock gain.

typedef enum *_spdif_tx_source* spdif_tx_source_t
SPDIF tx data source.

typedef enum *_spdif_validity_config* spdif_validity_config_t
SPDIF tx data source.

typedef struct *_spdif_config* spdif_config_t
SPDIF user configuration structure.

typedef struct *_spdif_transfer* spdif_transfer_t
SPDIF transfer structure.

typedef struct *_spdif_handle* spdif_handle_t

typedef void (*spdif_transfer_callback_t)(SPDIF_Type *base, *spdif_handle_t* *handle, *status_t* status, void *userData)
SPDIF transfer callback prototype.

SPDIF_XFER_QUEUE_SIZE
SPDIF transfer queue size, user can refine it according to use case.

struct _spdif_config
#include <fsl_spdif.h> SPDIF user configuration structure.

Public Members

bool isTxAutoSync
If auto sync mechanism open

bool isRxAutoSync
If auto sync mechanism open

uint8_t DPLLClkSource
SPDIF DPLL clock source, range from 0~15, meaning is chip-specific

uint8_t txClkSource
SPDIF tx clock source, range from 0~7, meaning is chip-specific

spdif_rxfull_select_t rxFullSelect
SPDIF rx buffer full select

spdif_txempty_select_t txFullSelect
SPDIF tx buffer empty select

spdif_uchannel_source_t uChannelSrc
U channel source

spdif_tx_source_t txSource
SPDIF tx data source

spdif_validity_config_t validityConfig
Validity flag config

spdif_gain_select_t gain
Rx receive clock measure gain parameter.

struct _spdif_transfer
#include <fsl_spdif.h> SPDIF transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

uint8_t *qdata
Data buffer for Q channel

uint8_t *udata
Data buffer for C channel

size_t dataSize
Transfer size.

struct __spdif_handle
#include <fsl_spdif.h> SPDIF handle structure.

Public Members

uint32_t state
Transfer status

spdif_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

spdif_transfer_t spdifQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

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

uint8_t watermark
Watermark value

2.36 SRC: System Reset Controller Driver

FSL_SRC_DRIVER_VERSION
SRC driver version 2.0.1.

enum __src_reset_status_flags
SRC reset status flags.

Values:

enumerator kSRC_WarmBootIndicationFlag
WARM boot indication shows that WARM boot was initiated by software.

enumerator kSRC_TemperatureSensorResetFlag
Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt needs to be served before this bit can be cleaned.

enumerator kSRC_JTAGSoftwareResetFlag
Indicates whether the reset was the result of setting SJC_GPCCR bit 31.

enumerator kSRC_JTAGGeneratedResetFlag
Indicates a reset has been caused by JTAG selection of certain IR codes: EXTEST or HIGHZ.

enumerator kSRC_WatchdogResetFlag
Indicates a reset has been caused by the watchdog timer timing out. This reset source can be blocked by disabling the watchdog.

enum _src_warm_reset_bypass_count

Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Values:

enumerator kSRC_WarmResetWaitAlways

System will wait until MMDC acknowledge is asserted.

enumerator kSRC_WarmResetWaitClk16

Wait 16 32KHz clock cycles before switching the reset.

enumerator kSRC_WarmResetWaitClk32

Wait 32 32KHz clock cycles before switching the reset.

enumerator kSRC_WarmResetWaitClk64

Wait 64 32KHz clock cycles before switching the reset.

typedef enum *_src_warm_reset_bypass_count* src_warm_reset_bypass_count_t

Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

static inline void SRC_EnableWDOGReset(SRC_Type *base, bool enable)

Enable the WDOG Reset in SRC.

WDOG Reset is enabled in SRC by default. If the WDOG event to SRC is masked, it would not create a reset to the chip. During the time the WDOG event is masked, when the WDOG event flag is asserted, it would remain asserted regardless of servicing the WDOG module. The only way to clear that bit is the hardware reset.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline void SRC_SetWarmResetBypassCount(SRC_Type *base, *src_warm_reset_bypass_count_t* option)

Set the delay count of waiting MMDC's acknowledge.

This function would define the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Parameters

- base – SRC peripheral base address.
- option – The option of setting mode, see to src_warm_reset_bypass_count_t.

static inline void SRC_EnableWarmReset(SRC_Type *base, bool enable)

Enable the WARM reset.

Only when the WARM reset is enabled, the WARM reset requests would be served by WARM reset. Otherwise, all the WARM reset sources would generate COLD reset.

Parameters

- base – SRC peripheral base address.
- enable – Enable the WARM reset or not.

static inline uint32_t SRC_GetBootModeWord1(SRC_Type *base)

Get the boot mode register 1 value.

The Boot Mode register contains bits that reflect the status of BOOT_CFGx pins of the chip. See to chip-specific document for detail information about value.

Parameters

- base – SRC peripheral base address.

Returns

status of BOOT_CFGx pins of the chip.

static inline uint32_t SRC_GetBootModeWord2(SRC_Type *base)

Get the boot mode register 2 value.

The Boot Mode register contains bits that reflect the status of BOOT_MODEx Pins and fuse values that controls boot of the chip. See to chip-specific document for detail information about value.

Parameters

- base – SRC peripheral base address.

Returns

status of BOOT_MODEx Pins and fuse values that controls boot of the chip.

static inline void SRC_SetWarmBootIndication(SRC_Type *base, bool enable)

Set the warm boot indication flag.

WARM boot indication shows that WARM boot was initiated by software. This indicates to the software that it saved the needed information in the memory before initiating the WARM reset. In this case, software will set this bit to '1', before initiating the WARM reset. The warm_boot bit should be used as indication only after a warm_reset sequence. Software should clear this bit after warm_reset to indicate that the next warm_reset is not performed with warm_boot.

Parameters

- base – SRC peripheral base address.
- enable – Assert the flag or not.

static inline uint32_t SRC_GetResetStatusFlags(SRC_Type *base)

Get the status flags of SRC.

Parameters

- base – SRC peripheral base address.

Returns

Mask value of status flags, see to _src_reset_status_flags.

void SRC_ClearResetStatusFlags(SRC_Type *base, uint32_t flags)

Clear the status flags of SRC.

Parameters

- base – SRC peripheral base address.
- flags – value of status flags to be cleared, see to _src_reset_status_flags.

static inline void SRC_SetGeneralPurposeRegister(SRC_Type *base, uint32_t index, uint32_t value)

Set value to general purpose registers.

General purpose registers (GPRx) would hold the value during reset process. Wakeup function could be kept in these register. For example, the GPR1 holds the entry function for

waking-up from Partial SLEEP mode while the GPR2 holds the argument. Other GPRx register would store the arbitray values.

Parameters

- base – SRC peripheral base address.
- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.
- value – Setting value for GPRx register.

static inline uint32_t SRC_GetGeneralPurposeRegister(SRC_Type *base, uint32_t index)

Get the value from general purpose registers.

Parameters

- base – SRC peripheral base address.
- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.

Returns

The setting value for GPRx register.

2.37 TMU: Thermal Management Unit Driver

enum _tmu_interrupt_enable

TMU interrupt enable.

Values:

enumerator kTMU_ImmediateTemperatureInterruptEnable

Immediate temperature threshold exceeded interrupt enable.

enumerator kTMU_AverageTemperatureInterruptEnable

Average temperature threshold exceeded interrupt enable.

enumerator kTMU_AverageTemperatureCriticalInterruptEnable

Average temperature critical threshold exceeded interrupt enable. >

enum _tmu_interrupt_status_flags

TMU interrupt status flags.

Values:

enumerator kTMU_ImmediateTemperatureStatusFlags

Immediate temperature threshold exceeded(ITTE).

enumerator kTMU_AverageTemperatureStatusFlags

Average temperature threshold exceeded(ATTE).

enumerator kTMU_AverageTemperatureCriticalStatusFlags

Average temperature critical threshold exceeded.(ATCTE)

enum _tmu_average_low_pass_filter

Average low pass filter setting.

Values:

enumerator kTMU_AverageLowPassFilter1_0

Average low pass filter = 1.

enumerator kTMU_AverageLowPassFilter0_5
Average low pass filter = 0.5.

enumerator kTMU_AverageLowPassFilter0_25
Average low pass filter = 0.25.

enumerator kTMU_AverageLowPassFilter0_125
Average low pass filter = 0.125.

enum _tmu_amplifier_gain
Amplifier gain setting.

Values:

enumerator kTMU_AmplifierGain6_34
TMU amplifier gain voltage 6.34mV.

enumerator kTMU_AmplifierGain6_485
TMU amplifier gain voltage 6.485mV.

enumerator kTMU_AmplifierGain6_63
TMU amplifier gain voltage 6.63mV.

enumerator kTMU_AmplifierGain6_775
TMU amplifier gain voltage 6.775mV.

enumerator kTMU_AmplifierGain6_92
TMU amplifier gain voltage 6.92mV.

enumerator kTMU_AmplifierGain7_065
TMU amplifier gain voltage 7.065mV.

enumerator kTMU_AmplifierGain7_21
TMU amplifier gain voltage 7.21mV.

enumerator kTMU_AmplifierGain7_355
TMU amplifier gain voltage 7.355mV.

enumerator kTMU_AmplifierGain7_5
TMU amplifier gain voltage 7.5mV.

enumerator kTMU_AmplifierGain7_645
TMU amplifier gain voltage 7.645mV.

enumerator kTMU_AmplifierGain7_79
TMU amplifier gain voltage 7.79mV.

enumerator kTMU_AmplifierGain7_935
TMU amplifier gain voltage 7.935mV.

enumerator kTMU_AmplifierGain8_08
TMU amplifier gain voltage 8.08mV(default).

enumerator kTMU_AmplifierGain8_225
TMU amplifier gain voltage 8.225mV.

enumerator kTMU_AmplifierGain8_37
TMU amplifier gain voltage 8.37mV.

enumerator kTMU_AmplifierGain8_515
TMU amplifier gain voltage 8.515mV.

enum _tmu_amplifier_reference_voltage

Amplifier reference voltage setting.

Values:

enumerator kTMU_AmplifierReferenceVoltage510
TMU amplifier reference voltage 510mV.

enumerator kTMU_AmplifierReferenceVoltage517_5
TMU amplifier reference voltage 517.5mV.

enumerator kTMU_AmplifierReferenceVoltage525
TMU amplifier reference voltage 525mV.

enumerator kTMU_AmplifierReferenceVoltage532_5
TMU amplifier reference voltage 532.5mV.

enumerator kTMU_AmplifierReferenceVoltage540
TMU amplifier reference voltage 540mV.

enumerator kTMU_AmplifierReferenceVoltage547_5
TMU amplifier reference voltage 547.5mV.

enumerator kTMU_AmplifierReferenceVoltage555
TMU amplifier reference voltage 555mV.

enumerator kTMU_AmplifierReferenceVoltage562_5
TMU amplifier reference voltage 562.5mV.

enumerator kTMU_AmplifierReferenceVoltage570
TMU amplifier reference voltage 570mV.

enumerator kTMU_AmplifierReferenceVoltage577_5
TMU amplifier reference voltage 577.5mV.

enumerator kTMU_AmplifierReferenceVoltage585
TMU amplifier reference voltage 585mV.

enumerator kTMU_AmplifierReferenceVoltage592_5
TMU amplifier reference voltage 592.5mV.

enumerator kTMU_AmplifierReferenceVoltage600
TMU amplifier reference voltage 600mV.

enumerator kTMU_AmplifierReferenceVoltage607_5
TMU amplifier reference voltage 607.5mV.

enumerator kTMU_AmplifierReferenceVoltage615
TMU amplifier reference voltage 615mV.

enumerator kTMU_AmplifierReferenceVoltage622_5
TMU amplifier reference voltage 622.5mV.

enumerator kTMU_AmplifierReferenceVoltage630
TMU amplifier reference voltage 630mV.

enumerator kTMU_AmplifierReferenceVoltage637_5
TMU amplifier reference voltage 637.5mV.

enumerator kTMU_AmplifierReferenceVoltage645
TMU amplifier reference voltage 645mV.

enumerator kTMU_AmplifierReferenceVoltage652_5
TMU amplifier reference voltage 652.5mV(default).

enumerator kTMU_AmplifierReferenceVoltage660
TMU amplifier reference voltage 660mV.

enumerator kTMU_AmplifierReferenceVoltage667_5
TMU amplifier reference voltage 667.5mV.

enumerator kTMU_AmplifierReferenceVoltage675
TMU amplifier reference voltage 675mV.

enumerator kTMU_AmplifierReferenceVoltage682_5
TMU amplifier reference voltage 682.5mV.

enumerator kTMU_AmplifierReferenceVoltage690
TMU amplifier reference voltage 690mV.

enumerator kTMU_AmplifierReferenceVoltage697_5
TMU amplifier reference voltage 697.5mV.

enumerator kTMU_AmplifierReferenceVoltage705
TMU amplifier reference voltage 705mV.

enumerator kTMU_AmplifierReferenceVoltage712_5
TMU amplifier reference voltage 712.5mV.

enumerator kTMU_AmplifierReferenceVoltage720
TMU amplifier reference voltage 720mV.

enumerator kTMU_AmplifierReferenceVoltage727_5
TMU amplifier reference voltage 727.5mV.

enumerator kTMU_AmplifierReferenceVoltage735
TMU amplifier reference voltage 735mV.

enumerator kTMU_AmplifierReferenceVoltage742_5
TMU amplifier reference voltage 742.5mV.

typedef struct *_tmu_thresold_config* tmu_thresold_config_t
configuration for TMU thresold.

typedef struct *_tmu_interrupt_status* tmu_interrupt_status_t
TMU interrupt status.

typedef enum *_tmu_average_low_pass_filter* tmu_average_low_pass_filter_t
Average low pass filter setting.

typedef enum *_tmu_amplifier_gain* tmu_amplifier_gain_t
Amplifier gain setting.

typedef enum *_tmu_amplifier_reference_voltage* tmu_amplifier_reference_voltage_t
Amplifier reference voltage setting.

typedef struct *_tmu_config* tmu_config_t
Configuration for TMU module.

void TMU_Init(TMU_Type *base, const *tmu_config_t* *config)
Enable the access to TMU registers and Initialize TMU module.

Parameters

- base – TMU peripheral base address.

- config – Pointer to configuration structure. Refer to “tmu_config_t” structure.

void TMU_Deinit(TMU_Type *base)

De-initialize TMU module and Disable the access to DCDC registers.

Parameters

- base – TMU peripheral base address.

void TMU_GetDefaultConfig(*tmu_config_t* *config)

Gets the default configuration for TMU.

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

Example:

```
config->averageLPF = kTMU_AverageLowPassFilter0_5;
```

Parameters

- config – Pointer to TMU configuration structure.

static inline void TMU_Enable(TMU_Type *base, bool enable)

Enable/Disable monitoring the temperature sensor.

Parameters

- base – TMU peripheral base address.
- enable – Switcher to enable/disable TMU.

static inline void TMU_EnableInterrupts(TMU_Type *base, uint32_t mask)

Enable the TMU interrupts.

Parameters

- base – TMU peripheral base address.
- mask – The interrupt mask. Refer to “_tmu_interrupt_enable” enumeration.

static inline void TMU_DisableInterrupts(TMU_Type *base, uint32_t mask)

Disable the TMU interrupts.

Parameters

- base – TMU peripheral base address.
- mask – The interrupt mask. Refer to “_tmu_interrupt_enable” enumeration.

void TMU_GetInterruptStatusFlags(TMU_Type *base, *tmu_interrupt_status_t* *status)

Get interrupt status flags.

Parameters

- base – TMU peripheral base address.
- status – The pointer to interrupt status structure. Record the current interrupt status. Please refer to “tmu_interrupt_status_t” structure.

void TMU_ClearInterruptStatusFlags(TMU_Type *base, uint32_t mask)

Clear interrupt status flags.

Parameters

- base – TMU peripheral base address.

- mask – The mask of interrupt status flags. Refer to "_tmu_interrupt_status_flags" enumeration.

status_t TMU_GetImmediateTemperature(TMU_Type *base, uint32_t *temperature)

Get the last immediate temperature at site.

Parameters

- base – TMU peripheral base address.
- temperature – Last immediate temperature reading at site when V=1.

Return values

- kStatus_Success – Temperature reading is valid.
- kStatus_Fail – Temperature reading is not valid because temperature out of sensor range or first measurement still pending.

Returns

Execution status.

status_t TMU_GetAverageTemperature(TMU_Type *base, uint32_t *temperature)

Get the last average temperature at site.

Parameters

- base – TMU peripheral base address.
- temperature – Last average temperature reading at site.

Return values

- kStatus_Success – Temperature reading is valid.
- kStatus_Fail – Temperature reading is not valid because temperature out of sensor range or first measurement still pending.

Returns

Execution status.

void TMU_SetHighTemperatureThresold(TMU_Type *base, const *tmu_thresold_config_t* *config)

Configure the high temperature thresold value and enable/disable relevant thresold.

Parameters

- base – TMU peripheral base address.
- config – Pointer to configuration structure. Refer to "tmu_thresold_config_t" structure.

FSL_TMU_DRIVER_VERSION

TMU driver version.

Version 2.1.1.

struct _tmu_thresold_config

#include <fsl_tmu.h> configuration for TMU thresold.

Public Members

bool immediateThresoldEnable

Enable high temperature immediate threshold.

bool AverageThresoldEnable

Enable high temperature average threshold.

bool AverageCriticalThresoldEnable

Enable high temperature average critical threshold.

uint8_t immediateThresoldValue

Range:10U-125U. Valid when corresponding threshold is enabled. High temperature immediate threshold value. Determines the current upper temperature threshold, for any enabled monitored site.

uint8_t averageThresoldValue

Range:10U-125U. Valid when corresponding threshold is enabled. High temperature average threshold value. Determines the average upper temperature threshold, for any enabled monitored site.

uint8_t averageCriticalThresoldValue

Range:10U-125U. Valid when corresponding threshold is enabled. High temperature average critical threshold value. Determines the average upper critical temperature threshold, for any enabled monitored site.

struct __tmu__interrupt__status

#include <fsl_tmu.h> TMU interrupt status.

Public Members

uint32_t interruptDetectMask

The mask of interrupt status flags. Refer to "_tmu_interrupt_status_flags" enumeration.

struct __tmu__config

#include <fsl_tmu.h> Configuration for TMU module.

Public Members

tmu_average_low_pass_filter_t averageLPF

The average temperature is calculated as: ALPF x Current_Temp + (1 - ALPF) x Average_Temp. For proper operation, this field should only change when monitoring is disabled.

2.38 UART: Universal Asynchronous Receiver/Transmitter Driver

2.39 UART Driver

static inline void UART__SoftwareReset(UART_Type *base)

Resets the UART using software.

This function resets the transmit and receive state machines, all FIFOs and register USR1, USR2, UBIR, UBMR, UBRC , URXD, UTXD and UTS[6-3]

Parameters

- base – UART peripheral base address.

status_t UART_Init(UART_Type *base, const *uart_config_t* *config, uint32_t srcClock_Hz)

Initializes an UART instance with the user configuration structure and the peripheral clock.

This function configures the UART module with user-defined settings. Call the UART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the UART.

```
uart_config_t uartConfig;
uartConfig.baudRate_Bps = 115200U;
uartConfig.parityMode = kUART_ParityDisabled;
uartConfig.dataBitsCount = kUART_EightDataBits;
uartConfig.stopBitCount = kUART_OneStopBit;
uartConfig.txFifoWatermark = 2;
uartConfig.rxFifoWatermark = 1;
uartConfig.enableAutoBaudrate = false;
uartConfig.enableTx = true;
uartConfig.enableRx = true;
UART_Init(UART1, &uartConfig, 24000000U);
```

Parameters

- base – UART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – UART clock source frequency in HZ.

Return values

kStatus_Success – UART initialize succeed

void UART_Deinit(UART_Type *base)

Deinitializes a UART instance.

This function waits for transmit to complete, disables TX and RX, and disables the UART clock.

Parameters

- base – UART peripheral base address.

void UART_GetDefaultConfig(*uart_config_t* *config)

Gets the default configuration structure.

l

This function initializes the UART configuration structure to a default value. The default values are: uartConfig->baudRate_Bps = 115200U; uartConfig->parityMode = kUART_ParityDisabled; uartConfig->dataBitsCount = kUART_EightDataBits; uartConfig->stopBitCount = kUART_OneStopBit; uartConfig->txFifoWatermark = 2; uartConfig->rxFifoWatermark = 1; uartConfig->enableAutoBaudrate = flase; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t UART_SetBaudRate(UART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the UART instance baud rate.

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the UART_Init.

```
UART_SetBaudRate(UART1, 115200U, 20000000U);
```

Parameters

- base – UART peripheral base address.
- baudRate_Bps – UART baudrate to be set.
- srcClock_Hz – UART clock source frequency in Hz.

Return values

- kStatus_UART_BaudrateNotSupport – Baudrate is not support in the current clock source.
- kStatus_Success – Set baudrate succeeded.

static inline void UART_Enable(UART_Type *base)

This function is used to Enable the UART Module.

Parameters

- base – UART base pointer.

static inline void UART_SetIdleCondition(UART_Type *base, *uart_idle_condition_t* condition)

This function is used to configure the IDLE line condition.

Parameters

- base – UART base pointer.
- condition – IDLE line detect condition of the enumerators in uart_idle_condition_t.

static inline void UART_Disable(UART_Type *base)

This function is used to Disable the UART Module.

Parameters

- base – UART base pointer.

bool UART_GetStatusFlag(UART_Type *base, uint32_t flag)

This function is used to get the current status of specific UART status flag(including interrupt flag). The available status flag can be select from uart_status_flag_t enumeration.

Parameters

- base – UART base pointer.
- flag – Status flag to check.

Return values

current – state of corresponding status flag.

void UART_ClearStatusFlag(UART_Type *base, uint32_t flag)

This function is used to clear the current status of specific UART status flag. The available status flag can be select from uart_status_flag_t enumeration.

Parameters

- base – UART base pointer.
- flag – Status flag to clear.

void UART_EnableInterrupts(UART_Type *base, uint32_t mask)

Enables UART interrupts according to the provided mask.

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _uart_interrupt_enable. For example, to enable TX empty interrupt and RX data ready interrupt, do the following.

```
UART_EnableInterrupts(UART1,kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
```

Parameters

- base – UART peripheral base address.
- mask – The interrupts to enable. Logical OR of _uart_interrupt_enable.

void UART_DisableInterrupts(UART_Type *base, uint32_t mask)

Disables the UART interrupts according to the provided mask.

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _uart_interrupt_enable. For example, to disable TX empty interrupt and RX data ready interrupt do the following.

```
UART_EnableInterrupts(UART1,kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
```

Parameters

- base – UART peripheral base address.
- mask – The interrupts to disable. Logical OR of _uart_interrupt_enable.

uint32_t UART_GetEnabledInterrupts(UART_Type *base)

Gets enabled UART interrupts.

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _uart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _uart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);

if (kUART_TxEmptyEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – UART peripheral base address.

Returns

UART interrupt flags which are logical OR of the enumerators in _uart_interrupt_enable.

static inline void UART_EnableTx(UART_Type *base, bool enable)

Enables or disables the UART transmitter.

This function enables or disables the UART transmitter.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_EnableRx(UART_Type *base, bool enable)

Enables or disables the UART receiver.

This function enables or disables the UART receiver.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_WriteByte(UART_Type *base, uint8_t data)

Writes to the transmitter register.

This function is used to write data to transmitter register. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – UART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t UART_ReadByte(UART_Type *base)

Reads the receiver register.

This function is used to read data from receiver register. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – UART peripheral base address.

Returns

Data read from data register.

status_t UART_WriteBlocking(UART_Type *base, const uint8_t *data, size_t length)

Writes to the TX register using a blocking method.

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

- base – UART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t UART_ReadBlocking(UART_Type *base, uint8_t *data, size_t length)

Read RX data register using a blocking method.

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the TX register.

Parameters

- base – UART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_UART_RxHardwareOverrun – Receiver overrun occurred while receiving data.
- kStatus_UART_NoiseError – A noise error occurred while receiving data.
- kStatus_UART_FramingError – A framing error occurred while receiving data.
- kStatus_UART_ParityError – A parity error occurred while receiving data.
- kStatus_UART_Timeout – Transmission timed out and was aborted.

- kStatus_Success – Successfully received all data.

void UART_TransferCreateHandle(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

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

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

void UART_TransferStartRingBuffer(UART_Type *base, *uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If data is already received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- ringBuffer – Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void UART_TransferStopRingBuffer(UART_Type *base, *uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

size_t UART_TransferGetRxRingBufferLength(*uart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- handle – UART handle pointer.

Returns

Length of received data in RX ring buffer.

status_t UART_TransferSendNonBlocking(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the kStatus_UART_TxIdle as status parameter.

Note: The kStatus_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the kUART_TransmissionCompleteFlag to ensure that the TX is finished.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART transfer structure. See uart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished; data not all written to TX register yet.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortSend(UART_Type *base, *uart_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 – UART peripheral base address.
- handle – UART handle pointer.

status_t UART_TransferGetSendCount(UART_Type *base, *uart_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the UART TX register.

This function gets the number of bytes written to the UART TX register by using the interrupt method.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – The parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t UART_TransferReceiveNonBlocking(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using an interrupt method.

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in 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 to read, the receive request is saved by the UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the xfer->data and this function returns with the parameter receivedBytes set to 5. For the left 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART 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 the xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into transmit queue.
- kStatus_UART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortReceive(UART_Type *base, *uart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

status_t UART_TransferGetReceiveCount(UART_Type *base, *uart_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 – UART peripheral base address.
- handle – UART 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 UART_TransferHandleIRQ(UART_Type *base, void *irqHandle)

UART IRQ handle function.

This function handles the UART transmit and receive IRQ request.

Parameters

- base – UART peripheral base address.
- irqHandle – UART handle pointer.

static inline void UART_EnableTxDMA(UART_Type *base, bool enable)

Enables or disables the UART transmitter DMA request.

This function enables or disables the transmit request when the transmitter has one or more slots available in the TxFIFO. The fill level in the TxFIFO that generates the DMA request is controlled by the TXTL bits.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_EnableRxDMA(UART_Type *base, bool enable)

Enables or disables the UART receiver DMA request.

This function enables or disables the receive request when the receiver has data in the RxFIFO. The fill level in the RxFIFO at which a DMA request is generated is controlled by the RXTL bits .

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_SetTxFifoWatermark(UART_Type *base, uint8_t watermark)

This function is used to set the watermark of UART Tx FIFO. A maskable interrupt is generated whenever the data level in the TxFIFO falls below the Tx FIFO watermark.

Parameters

- base – UART base pointer.
- watermark – The Tx FIFO watermark.

static inline void UART_SetRxRTSWatermark(UART_Type *base, uint8_t watermark)

This function is used to set the watermark of UART RTS deassertion.

The RTS signal deasserts whenever the data count in RxFIFO reaches the Rx RTS watermark.

Parameters

- base – UART base pointer.
- watermark – The Rx RTS watermark.

static inline void UART_SetRxFifoWatermark(UART_Type *base, uint8_t watermark)

This function is used to set the watermark of UART Rx FIFO. A maskable interrupt is generated whenever the data level in the RxFIFO reaches the Rx FIFO watermark.

Parameters

- base – UART base pointer.
- watermark – The Rx FIFO watermark.

static inline void UART_EnableAutoBaudRate(UART_Type *base, bool enable)

This function is used to set the enable condition of Automatic Baud Rate Detection feature.

Parameters

- base – UART base pointer.
- enable – Enable/Disable Automatic Baud Rate Detection feature.
  - true: Enable Automatic Baud Rate Detection feature.
  - false: Disable Automatic Baud Rate Detection feature.

static inline bool UART_IsAutoBaudRateComplete(UART_Type *base)

This function is used to read if the automatic baud rate detection has finished.

Parameters

- base – UART base pointer.

Returns

- true: Automatic baud rate detection has finished.
- false: Automatic baud rate detection has not finished.

FSL_UART_DRIVER_VERSION

UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_UART_TxBusy

Transmitter is busy.

enumerator kStatus_UART_RxBusy

Receiver is busy.

enumerator kStatus_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_UART_RxIdle

UART receiver is idle.

enumerator kStatus_UART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_UART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_UART_FlagCannotClearManually

UART flag can't be manually cleared.

enumerator kStatus_UART_Error

Error happens on UART.

enumerator kStatus_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_UART_RxHardwareOverrun

UART RX receiver overrun.

enumerator kStatus_UART_NoiseError

UART noise error.

enumerator kStatus_UART_FramingError
UART framing error.

enumerator kStatus_UART_ParityError
UART parity error.

enumerator kStatus_UART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_UART_BreakDetect
Receiver detect BREAK signal

enumerator kStatus_UART_Timeout
UART times out.

enum _uart_data_bits
UART data bits count.

Values:

enumerator kUART_SevenDataBits
Seven data bit

enumerator kUART_EightDataBits
Eight data bit

enum _uart_parity_mode
UART parity mode.

Values:

enumerator kUART_ParityDisabled
Parity disabled

enumerator kUART_ParityEven
Even error check is selected

enumerator kUART_ParityOdd
Odd error check is selected

enum _uart_stop_bit_count
UART stop bit count.

Values:

enumerator kUART_OneStopBit
One stop bit

enumerator kUART_TwoStopBit
Two stop bits

enum _uart_idle_condition
UART idle condition detect.

Values:

enumerator kUART_IdleFor4Frames
Idle for more than 4 frames

enumerator kUART_IdleFor8Frames
Idle for more than 8 frames

enumerator kUART_IdleFor16Frames
Idle for more than 16 frames

enumerator kUART_IdleFor32Frames
Idle for more than 32 frames

enum _uart_interrupt_enable
This structure contains the settings for all of the UART interrupt configurations.

Values:

enumerator kUART_AutoBaudEnable

enumerator kUART_TxReadyEnable

enumerator kUART_IdleEnable

enumerator kUART_RxReadyEnable

enumerator kUART_TxEmptyEnable

enumerator kUART_RtsDeltaEnable

enumerator kUART_EscapeEnable

enumerator kUART_RtsEnable

enumerator kUART_AgingTimerEnable

enumerator kUART_DtrEnable

enumerator kUART_ParityErrorEnable

enumerator kUART_FrameErrorEnable

enumerator kUART_DcdEnable

enumerator kUART_RiEnable

enumerator kUART_RxDsEnable

enumerator kUART_tAirWakeEnable

enumerator kUART_AwakeEnable

enumerator kUART_DtrDeltaEnable

enumerator kUART_AutoBaudCntEnable

enumerator kUART_IrEnable

enumerator kUART_WakeEnable

enumerator kUART_TxCompleteEnable

enumerator kUART_BreakDetectEnable

enumerator kUART_RxOverrunEnable

enumerator kUART_RxDataReadyEnable

enumerator kUART_RxDmaIdleEnable

enumerator kUART_AllInterruptsEnable

UART status flags.

This provides constants for the UART status flags for use in the UART functions.

Values:

enumerator kUART_RxCharReadyFlag
Rx Character Ready Flag.

enumerator kUART_RxErrorFlag
Rx Error Detect Flag.

enumerator kUART_RxOverrunErrorFlag
Rx Overrun Flag.

enumerator kUART_RxFrameErrorFlag
Rx Frame Error Flag.

enumerator kUART_RxBreakDetectFlag
Rx Break Detect Flag.

enumerator kUART_RxParityErrorFlag
Rx Parity Error Flag.

enumerator kUART_ParityErrorFlag
Parity Error Interrupt Flag.

enumerator kUART_RtsStatusFlag
RTS_B Pin Status Flag.

enumerator kUART_TxReadyFlag
Transmitter Ready Interrupt/DMA Flag.

enumerator kUART_RtsDeltaFlag
RTS Delta Flag.

enumerator kUART_EscapeFlag
Escape Sequence Interrupt Flag.

enumerator kUART_FrameErrorFlag
Frame Error Interrupt Flag.

enumerator kUART_RxReadyFlag
Receiver Ready Interrupt/DMA Flag.

enumerator kUART_AgingTimerFlag
Aging Timer Interrupt Flag.

enumerator kUART_DtrDeltaFlag
DTR Delta Flag.

enumerator kUART_RxDsFlag
Receiver IDLE Interrupt Flag.

enumerator kUART_tAirWakeFlag
Asynchronous IR WAKE Interrupt Flag.

enumerator kUART_AwakeFlag
Asynchronous WAKE Interrupt Flag.

enumerator kUART_Rs485SlaveAddrMatchFlag
RS-485 Slave Address Detected Interrupt Flag.

enumerator kUART_AutoBaudFlag
Automatic Baud Rate Detect Complete Flag.

enumerator kUART_TxEmptyFlag
Transmit Buffer FIFO Empty.

enumerator kUART_DtrFlag
DTR edge triggered interrupt flag.

enumerator kUART_IdleFlag
Idle Condition Flag.

enumerator kUART_AutoBaudCntStopFlag
Auto-baud Counter Stopped Flag.

enumerator kUART_RiDeltaFlag
Ring Indicator Delta Flag.

enumerator kUART_RiFlag
Ring Indicator Input Flag.

enumerator kUART_IrFlag
Serial Infrared Interrupt Flag.

enumerator kUART_WakeFlag
Wake Flag.

enumerator kUART_DcdDeltaFlag
Data Carrier Detect Delta Flag.

enumerator kUART_DcdFlag
Data Carrier Detect Input Flag.

enumerator kUART_RtsFlag
RTS Edge Triggered Interrupt Flag.

enumerator kUART_TxCompleteFlag
Transmitter Complete Flag.

enumerator kUART_BreakDetectFlag
BREAK Condition Detected Flag.

enumerator kUART_RxOverrunFlag
Overrun Error Flag.

enumerator kUART_RxDataReadyFlag
Receive Data Ready Flag.

typedef enum *_uart_data_bits* uart_data_bits_t
UART data bits count.

typedef enum *_uart_parity_mode* uart_parity_mode_t
UART parity mode.

typedef enum *_uart_stop_bit_count* uart_stop_bit_count_t
UART stop bit count.

typedef enum *_uart_idle_condition* uart_idle_condition_t
UART idle condition detect.

typedef struct *_uart_config* uart_config_t
UART configuration structure.

typedef struct *_uart_transfer* uart_transfer_t
UART transfer structure.

typedef struct *_uart_handle* uart_handle_t
Forward declaration of the handle typedef.

typedef void (*uart_transfer_callback_t)(UART_Type *base, *uart_handle_t* *handle, *status_t* status, void *userData)
UART transfer callback function.

typedef void (*uart_isr_t)(UART_Type *base, void *handle)

const IRQn_Type s_uartIRQ[]

uart_isr_t s_uartIsr

void *s_uartHandle[]
Pointers to uart handles for each instance.

uint32_t UART_GetInstance(UART_Type *base)
Get the UART instance from peripheral base address.

Parameters

- base – UART peripheral base address.

Returns
UART instance.

UART_RETRY_TIMES
Retry times for waiting flag.

struct uart_config
#include <fsl_uart.h> UART configuration structure.

Public Members

uint32_t baudRate_Bps
UART baud rate.

uart_parity_mode_t parityMode
Parity error check mode of this module.

uart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

uart_stop_bit_count_t stopBitCount
Number of stop bits in one frame.

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

uint8_t rxRTSWatermark
RX RTS watermark, RX FIFO data count being larger than this triggers RTS deassertion

bool enableAutoBaudRate
Enable automatic baud rate detection

bool enableTx
Enable TX

bool enableRx
: Enable RX

bool enableRxRTS
: RX RTS enable

bool enableTxCTS
: TX CTS enable

struct __uart__transfer
: *#include <fsl_uart.h>* UART transfer structure.

Public Members

size_t dataSize
: The byte count to be transfer.

struct __uart__handle
: *#include <fsl_uart.h>* UART handle structure.

Public Members

const uint8_t *volatile txData
: Address of remaining data to send.

volatile size_t txDataSize
: Size of the remaining data to send.

size_t txDataSizeAll
: Size of the data to send out.

uint8_t *volatile rxData
: Address of remaining data to receive.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t rxDataSizeAll
: Size of the data to receive.

uint8_t *rxRingBuffer
: Start address of the receiver ring buffer.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

uart_transfer_callback_t callback
: Callback function.

void *userData
: UART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
RX transfer state

union ___unnamed15___

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

2.40 UART FreeRTOS Driver

2.41 UART SDMA Driver

void UART_TransferCreateHandleSDMA(UART_Type *base, *uart_sdma_handle_t* *handle, *uart_sdma_transfer_callback_t* callback, void *userData, *sdma_handle_t* *txSdmaHandle, *sdma_handle_t* *rxSdmaHandle, uint32_t eventSourceTx, uint32_t eventSourceRx)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.
- callback – UART callback, NULL means no callback.
- userData – User callback function data.
- rxSdmaHandle – User-requested DMA handle for RX DMA transfer.
- txSdmaHandle – User-requested DMA handle for TX DMA transfer.
- eventSourceTx – Eventsource for TX DMA transfer.
- eventSourceRx – Eventsource for RX DMA transfer.

status_t UART_SendSDMA(UART_Type *base, *uart_sdma_handle_t* *handle, *uart_transfer_t* *xfer)

Sends data using sDMA.

This function sends data using sDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART sDMA transfer structure. See uart_transfer_t.

Return values

- kStatus_Success – if succeeded; otherwise failed.

- kStatus_UART_TxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

status_t UART_ReceiveSDMA(UART_Type *base, *uart_sdma_handle_t* *handle, *uart_transfer_t* *xfer)

Receives data using sDMA.

This function receives data using sDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.
- xfer – UART sDMA transfer structure. See uart_transfer_t.

Return values

- kStatus_Success – if succeeded; otherwise failed.
- kStatus_UART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortSendSDMA(UART_Type *base, *uart_sdma_handle_t* *handle)

Aborts the sent data using sDMA.

This function aborts sent data using sDMA.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.

void UART_TransferAbortReceiveSDMA(UART_Type *base, *uart_sdma_handle_t* *handle)

Aborts the receive data using sDMA.

This function aborts receive data using sDMA.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.

void UART_TransferSdmaHandleIRQ(UART_Type *base, void *uartSdmaHandle)

UART IRQ handle function.

This function handles the UART transmit complete IRQ request and invoke user callback.

Parameters

- base – UART peripheral base address.
- uartSdmaHandle – UART handle pointer.

FSL_UART_SDMA_DRIVER_VERSION

UART SDMA driver version.

typedef struct *_uart_sdma_handle* uart_sdma_handle_t

typedef void (*uart_sdma_transfer_callback_t)(UART_Type *base, *uart_sdma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct _uart_sdma_handle

#include <fsl_uart_sdma.h> UART sDMA handle.

Public Members

uart_sdma_transfer_callback_t callback
Callback function.

void *userData
UART callback function parameter.

size_t rxDataSizeAll
Size of the data to receive.

size_t txDataSizeAll
Size of the data to send out.

sdma_handle_t *txSdmaHandle
The sDMA TX channel used.

sdma_handle_t *rxSdmaHandle
The sDMA RX channel used.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.42 WDOG: Watchdog Timer Driver

void WDOG_GetDefaultConfig(*wdog_config_t* *config)
Initializes the WDOG configuration structure.

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
wdogConfig->enableWdog = true;
wdogConfig->workMode.enableWait = true;
wdogConfig->workMode.enableStop = true;
wdogConfig->workMode.enableDebug = true;
wdogConfig->enableInterrupt = false;
wdogConfig->enablePowerdown = false;
wdogConfig->resetExtension = flase;
wdogConfig->timeoutValue = 0xFFU;
wdogConfig->interruptTimeValue = 0x04u;
```

See also:

wdog_config_t

Parameters

- config – Pointer to the WDOG configuration structure.

void WDOG_Init(WDOG_Type *base, const *wdog_config_t* *config)
Initializes the WDOG.

This function initializes the WDOG. When called, the WDOG runs according to the configuration.

This is an example.

```
wdog_config_t config;
WDOG_GetDefaultConfig(&config);
config.timeoutValue = 0xffU;
config->interruptTimeValue = 0x04u;
WDOG_Init(wdog_base,&config);
```

Parameters

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

void WDOG_Deinit(WDOG_Type *base)

Shuts down the WDOG.

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

static inline void WDOG_Enable(WDOG_Type *base)

Enables the WDOG module.

This function writes a value into the WDOG_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_Disable(WDOG_Type *base)

Disables the WDOG module.

This function writes a value into the WDOG_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write,once the bit is set. only debug mode exception

Parameters

- base – WDOG peripheral base address

static inline void WDOG_TriggerSystemSoftwareReset(WDOG_Type *base)

Trigger the system software reset.

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to “1” after it has been asserted to “0”. Note: Calling this API will reset the system right now, please using it with more attention.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_TriggerSoftwareSignal(WDOG_Type *base)

Trigger an output assertion.

This function will write to the WCR[WDA] bit to trigger WDOG_B signal assertion. The WDOG_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG_B signal. Note: The WDOG_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_EnableInterrupts(WDOG_Type *base, uint16_t mask)

Enables the WDOG interrupt.

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

Parameters

- base – WDOG peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined.
  - kWDOG_InterruptEnable

uint16_t WDOG_GetStatusFlags(WDOG_Type *base)

Gets the WDOG all reset status flags.

This function gets all reset status flags.

```
uint16_t status;
status = WDOG_GetStatusFlags (wdog_base);
```

See also:

_wdog_status_flags

- true: a related status flag has been set.
- false: a related status flag is not set.

Parameters

- base – WDOG peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void WDOG_ClearInterruptStatus(WDOG_Type *base, uint16_t mask)

Clears the WDOG flag.

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
WDOG_ClearStatusFlags(wdog_base,KWDOG_InterruptFlag);
```

Parameters

- base – WDOG peripheral base address
- mask – The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

static inline void WDOG_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG timeout value.

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

static inline void WDOG_SetInterrputTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG interrupt count timeout value.

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are wirte-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

static inline void WDOG_DisablePowerDownEnable(WDOG_Type *base)

Disable the WDOG power down enable bit.

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are wirte-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

Parameters

- base – WDOG peripheral base address

void WDOG_Refresh(WDOG_Type *base)

Refreshes the WDOG timer.

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG peripheral base address

FSL_WDOG_DRIVER_VERSION

Defines WDOG driver version.

WDOG_REFRESH_KEY

enum _wdog_interrupt_enable

WDOG interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the WDOG interrupt configurations.

Values:

enumerator kWDOG_InterruptEnable

WDOG timeout generates an interrupt before reset

enum _wdog_status_flags

WDOG status flags.

This structure contains the WDOG status flags for use in the WDOG functions.

Values:

enumerator kWDOG_RunningFlag

Running flag, set when WDOG is enabled

enumerator kWDOG_PowerOnResetFlag

Power On flag, set when reset is the result of a powerOnReset

enumerator kWDOG_TimeoutResetFlag

Timeout flag, set when reset is the result of a timeout

enumerator kWDOG_SoftwareResetFlag
Software flag, set when reset is the result of a software

enumerator kWDOG_InterruptFlag
interrupt flag,whether interrupt has occurred or not

typedef struct *_wdog_work_mode* wdog_work_mode_t
Defines WDOG work mode.

typedef struct *_wdog_config* wdog_config_t
Describes WDOG configuration structure.

struct _wdog_work_mode
#include <fsl_wdog.h> Defines WDOG work mode.

Public Members

bool enableWait
If set to true, WDOG continues in wait mode

bool enableStop
If set to true, WDOG continues in stop mode

bool enableDebug
If set to true, WDOG continues in debug mode

struct _wdog_config
#include <fsl_wdog.h> Describes WDOG configuration structure.

Public Members

bool enableWdog
Enables or disables WDOG

wdog_work_mode_t workMode
Configures WDOG work mode in debug stop and wait mode

bool enableInterrupt
Enables or disables WDOG interrupt

uint16_t timeoutValue
Timeout value

uint16_t interruptTimeValue
Interrupt count timeout value

bool softwareResetExtension
software reset extension

bool enablePowerDown
power down enable bit

bool enableTimeOutAssert
Enable WDOG_B timeout assertion.

Chapter 3

Middleware

3.1 Motor Control

3.1.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called "packet-driven BDM" interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT]((https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to FreeMASTER community or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the src/platforms directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a "middleware" component which may be downloaded along with the example applications from https://mcuxpresso.nxp.com/en/welcome.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- The official FreeMASTER middleware repository.
- Online version of this document

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.

- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample spplications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.

- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.

- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory

block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- "External" single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- "True" single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as "automatic" in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- ***src/platforms*** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- ***src/common*** folder—contains the common driver source files shared by the driver for all supported platforms. All the *.c* files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
  - *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
  - *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
  - *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.

  - *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
  - *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
  - *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
  - *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
  - *freemaster_serial.h* - defines the low-level character-oriented Serial API.
  - *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
  - *freemaster_can.h* - defines the low-level message-oriented CAN API.
  - *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.
  - *freemaster_net.h* - definitions related to the Network transport.
  - *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
  - *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
  - *freemaster_utils.h* - definitions related to utility code.
- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR   [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See *Driver interrupt modes*.

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR_SERIAL** - serial communication protocol
- **FMSTR_CAN** - using CAN communication
- **FMSTR_PDBDM** - using packet-driven BDM communication
- **FMSTR_NET** - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)
- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetSerialBaseAddress()` to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode.
The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the "True" single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.
Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool.
Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables.
Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project.
Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions.
Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr, FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the FMSTR_Poll routine. An application interrupt can occur in the middle of the

Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all *.c files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See *Driver files* for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See *Driver configuration* for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the "idle" time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

*N * Tchar*

where:

- *N* is equal to the length of the receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance *0* which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-langiage symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type)  /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type)  /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type)  /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.

- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT*n*	Unsigned integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_SINT*n*	Signed integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_FRAC*n*	Fractional number of size *n* bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(*m*,*n*)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(*m*,*n*)	Unsigned fractional number in general UQ form (m+n total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FMSTR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")     /* entering a new virtual directory */
```

(continues on next page)

(continued from previous page)

```
/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*

- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
    FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↪
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_xxx and FMSTR_PIPE_SIZE_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the nGranularity value equal to the nLength value, all data are considered as one chunk which is either written successfully as a whole or not at all. The nGranularity value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

FMSTR_TSA_TI	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	The smallest memory entity. On the vast majority of platforms, this is an unsigned 8-bit integer. On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	A single character in the communication buffer. Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

Document references

Links

- This document online: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

3.2 MultiCore

3.2.1 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 25.12.00.
This software package contains components for efficient work with multicore devices as well as for the
multiprocessor communication.

What is new

- eRPC CHANGELOG
- RPMsg-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
  - FRDM-IMXRT1186
- 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
  - FRDM-IMXRT1186

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 <MCSDK_dir>	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call component	<MCSDK_dir>/erpc/
Multicore Manager component	<MCSDK_dir>/mcmgr/
RPMsg-Lite	<MCSDK_dir>/rpmsg_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 (RPMsg-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_dsp
- rpmsg_lite_pingpong_tzm

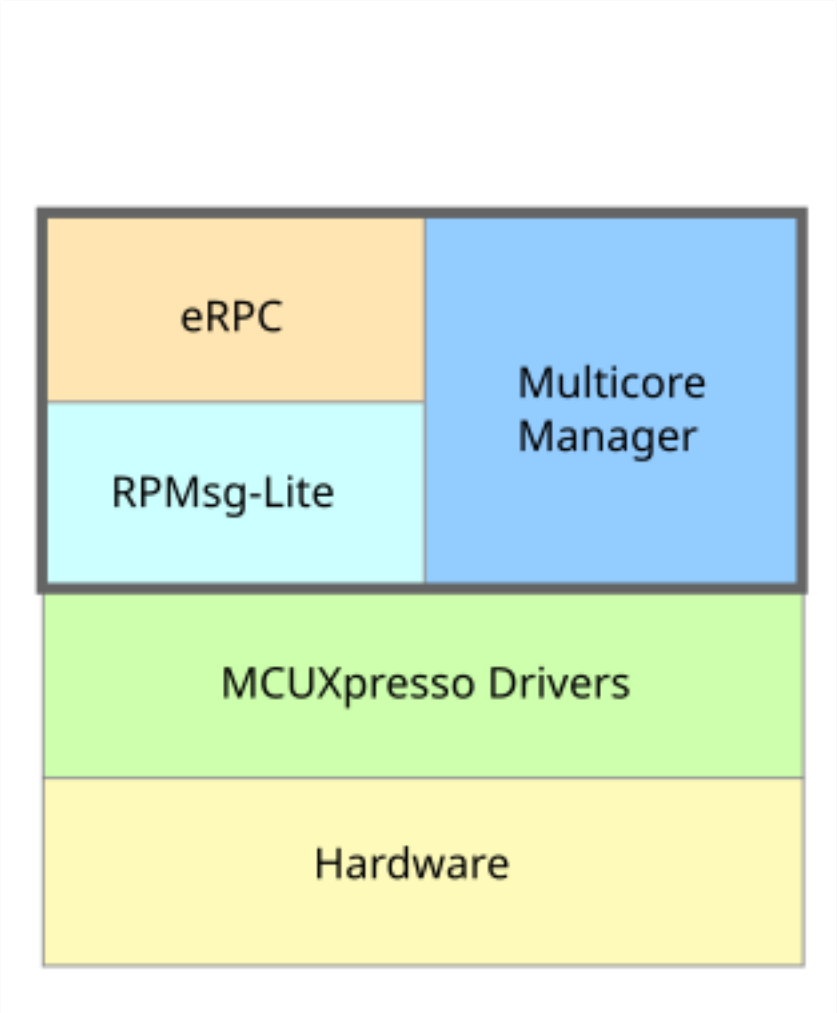
The eRPC multicore component can be leveraged for inter-processor communication and remote procedure calls between SoCs / development boards.
The following multiprocessor demo applications are located together with other MCUXpresso SDK examples in
the <MCUXpressoSDK_install_dir>/examples/multiprocessor_examples 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.

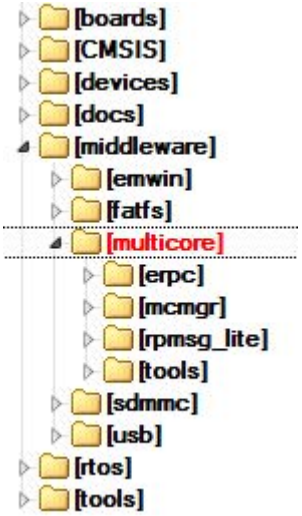


All the MCSDK-related files are located in <MCUXpressoSDK_install_dir>/middleware/multicore folder.

For supported toolchain versions, see the *Multicore SDK v25.12.00 Release Notes* (document MCSDKRN). For the latest version of this and other MCSDK documents, visit www.nxp.com.

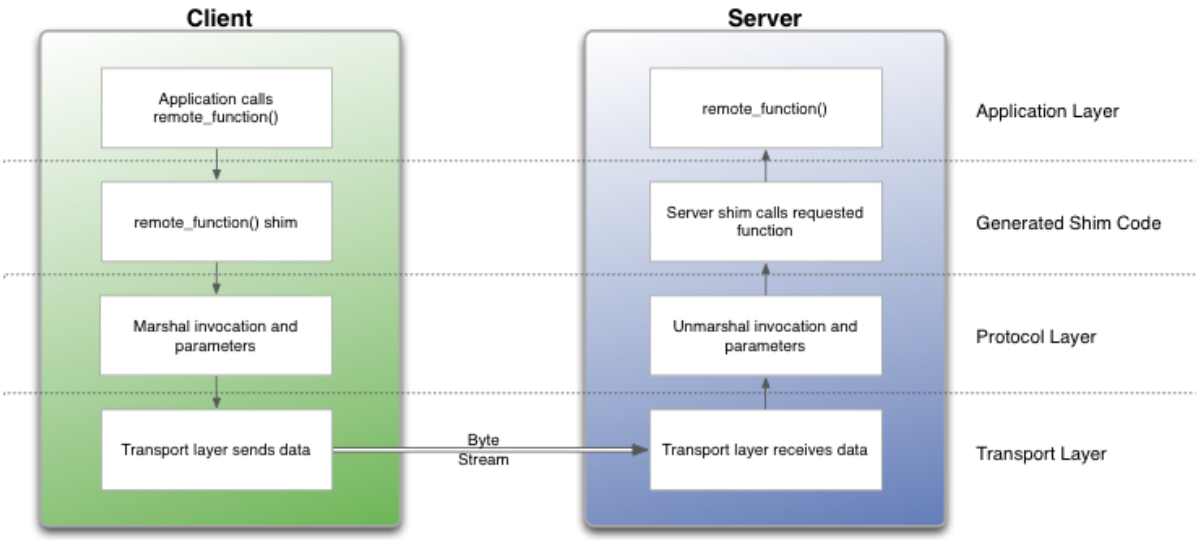
Multicore SDK (MCSDK) components The MCSDK consists of the following software components:

- **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, TPC/IP, UART, and so on). The server unmarshaled 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 example projects	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/
Multiprocessor example projects	<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.

[25.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.2
  - RPMsg-Lite v5.3.0

[25.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0

  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.1
  - RPMsg-Lite v5.2.1

[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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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 / RPMsg 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.

For Further API documentation, please look at doxygen documentation

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 overview please read RPMSG-Lite VirtIO Overview.

For RPMSG-Lite Design Considerations please read RPMSG-Lite Design Considerations.

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.

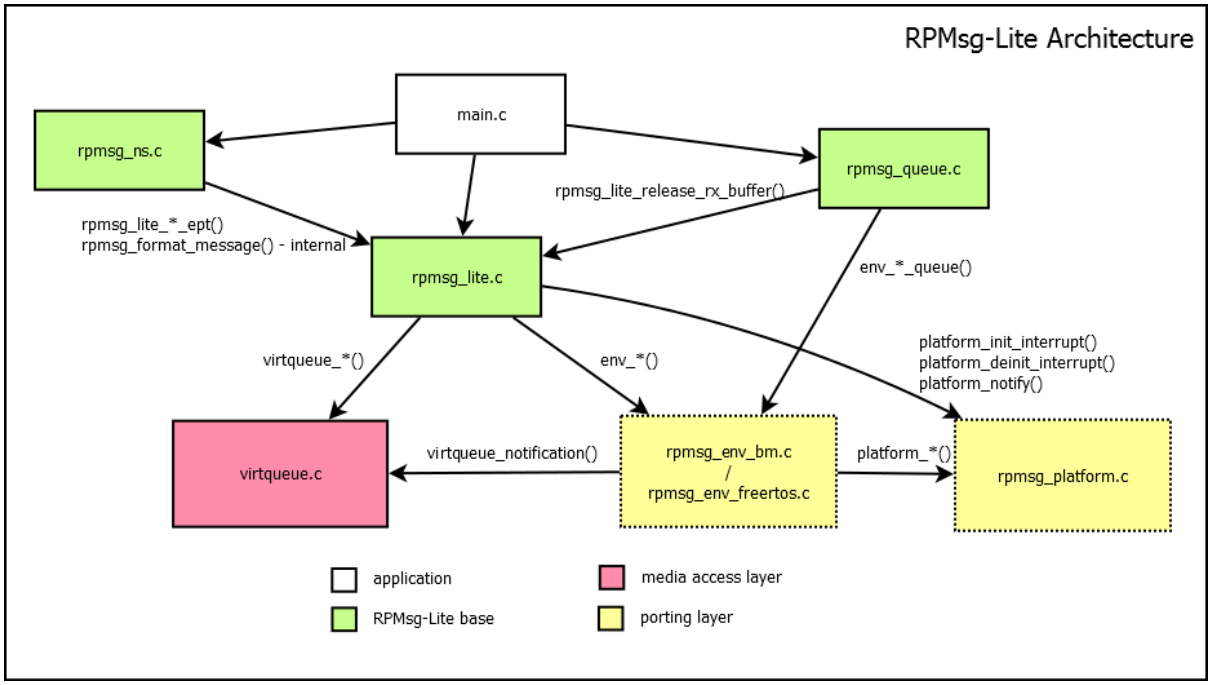
Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property 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 subcomponents, 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).

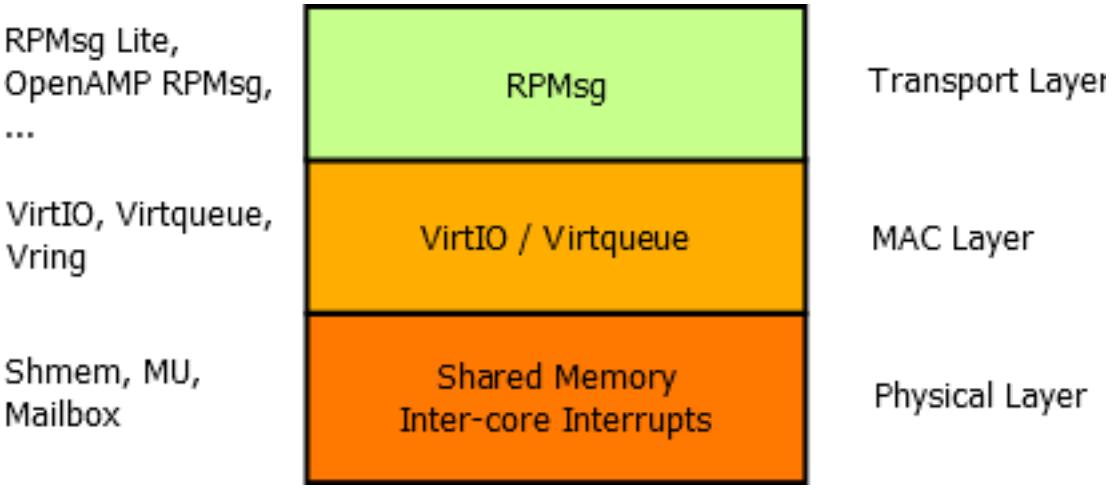
The actual "media access" layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

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 /rpmsg_lite/lib/include directory to the include path and in the application, include either the rpmsg_lite.h header file, or optionally also include the rpmsg_queue.h and/or rpmsg_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, rpmsg_env_myrtos.c) and to include it in the project build.

The initialization of the stack is done by calling the rpmsg_lite_master_init() on the master side and the rpmsg_lite_remote_init() on the remote side. This initialization function must be called prior to any RPMsg-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 rpmsg_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 rpmsg_queue_create() function must be called before calling rpmsg_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 rpmsg_queue_rx_cb(). Then, it is possible to use rpmsg_queue_receive() function to listen on a queue object for incoming messages. The rpmsg_lite_send() function is used to send messages to the other side.

The RPMsg-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.

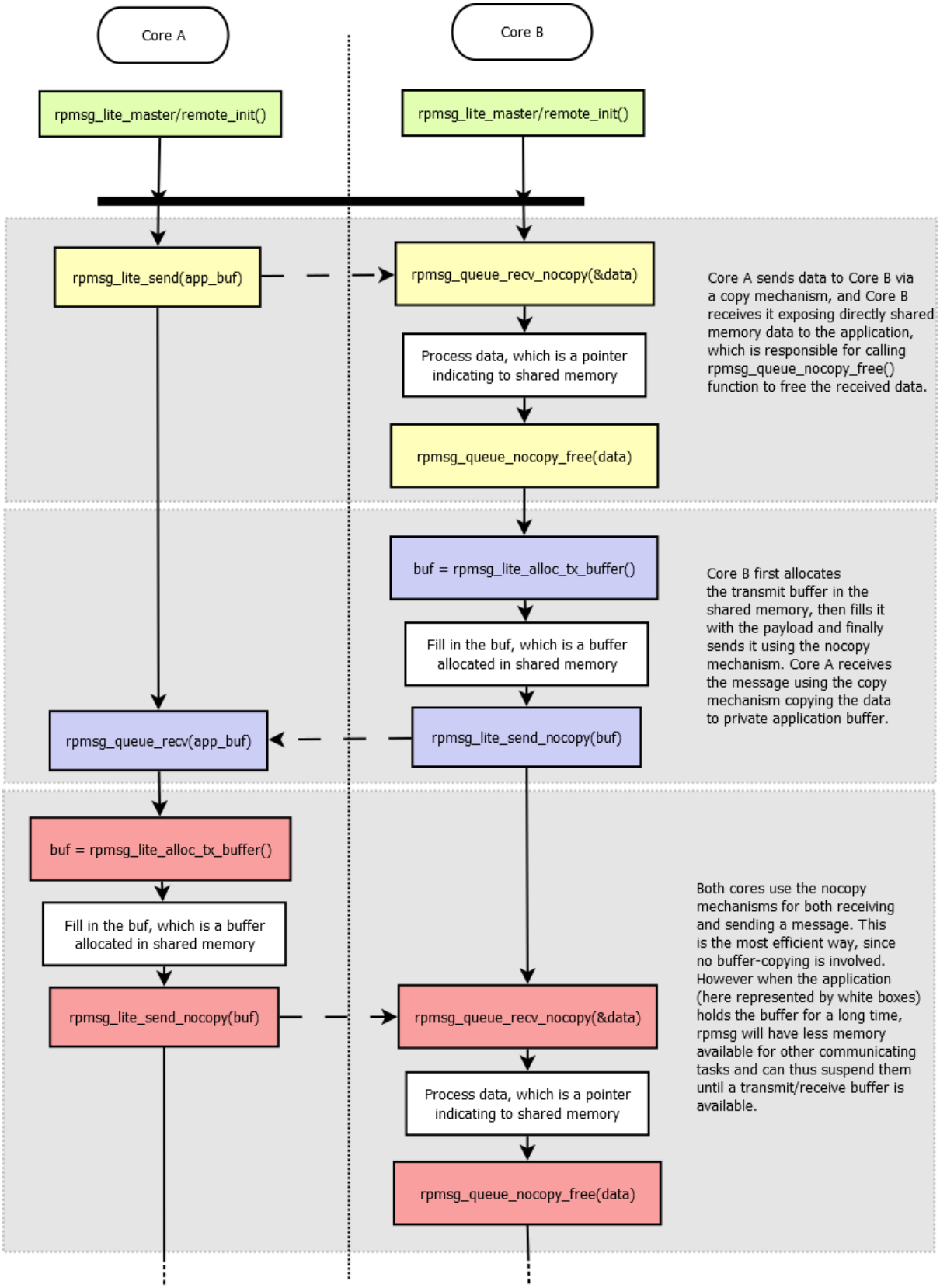
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the rpmsg_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 rpmsg_lite_alloc_tx_buffer() size output parameter).
- Call the rpmsg_lite_send_nocopy() function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the rpmsg_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 rpmsg_queue_recv_nocopy() function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the rpmsg_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 RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function rpmsg_queue_destroy() is used to destroy a queue, rpmsg_lite_destroy_ept() is used to destroy an endpoint and finally, rpmsg_lite_deinit() is used to deinitialize the RPMsg-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. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples RPMsg_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 (RPMsg_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 liker file is also a good practise, it is recommended to use linker files from MCUXpressSDK 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.

Configuration option	Default 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 more than 1 byte, and has to be word align (including rpmsg header size 16 bytes), if not it will be aligned up
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 rpmsg_lite_send_nocopy() and rpmsg_lite_release_rx_buffer() functions (enabled by RL_API_HAS_ZEROCOPY 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_ALLOV	(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_ALLOV	(0)	It allows to define custom shared memory configuration and replacing the shared memory related global settings from rpmsg_config.h 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_ASSER	see rpmsg	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 RPMSG-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.

[v5.3.0]

Added

- RT700 porting layer added support to send rpmsg messages between CM33_0 <-> Hifi1 and CM33_1 <-> Hifi4 cores.
- Add new platform macro RL_PLATFORM_MAX_ISR_COUNT this will set number of IRQ count per platform. This macro is then used in environment layers to set isr_table size where irq handles are registered. It size should match the bit length of VQ_ID so all combinations can fit into table.
- Unit tests updated to improve code coverage, new unit tests added covering static allocations in rtos environment layers.

Fixed

- virtio.h removed typedef uint8_t boolean and in its place use standard C99 bool type to avoid potential type conflicts.
- env_acquire_sync_lock() and env_release_sync_lock() synchronization primitives removed
- Kconfig consolidation, when RL_ALLOW_CUSTOM_SHMEM_CONFIG enabled the platform_get_custom_shmem_config() function needs to be implemented in platform layer to provide custom shared memory configuration for RPMsg-Lite instance.

v5.2.1

Added

- Doc added RPMSG-Lite VirtIO Overview
- Doc added RPSMG-Lite Design Considerations
- Added frdmimxrt1186 unit testing

Changed

- Remove limitation that RL_BUFFER_SIZE needs to be power of 2. It just has to be more than 16 bytes, e.g. 16 bytes of rpmsg header and payload size at least 1 byte and word aligned, if not it will be aligned up.

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.
- Baremental 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.

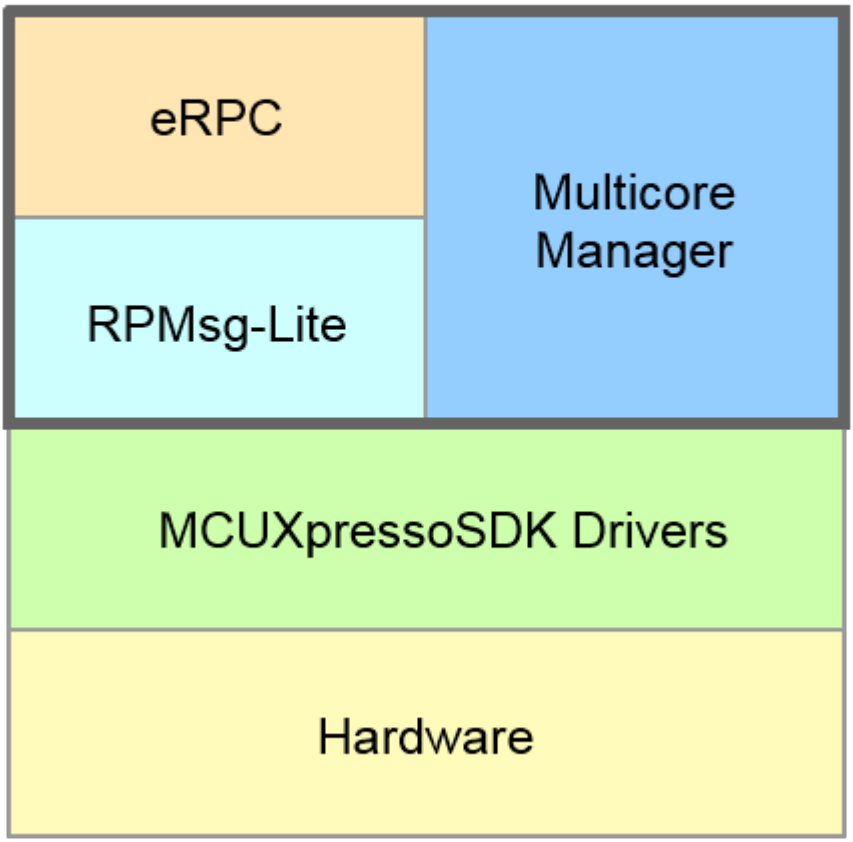
For Further API documentation, please look at doxygen documentation

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();
```

(continues on next page)

(continued from previous page)

```
    /* 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.

It could also happen that the secondary core application stops running correctly and the primary core application does not know about that situation. Therefore, it is beneficial to implement a mechanism for core health monitoring. The *test_heartbeat* unit test can serve as an example how to ensure that: secondary core could periodically send heartbeat signals to the primary core using MCMGR_TriggerEvent() API to indicate that it is alive and functioning properly.

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();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPMsgRemoteReadyEventHandler, (void␣
↪*)RPMsgRemoteReadyEventData);
```

(continues on next page)

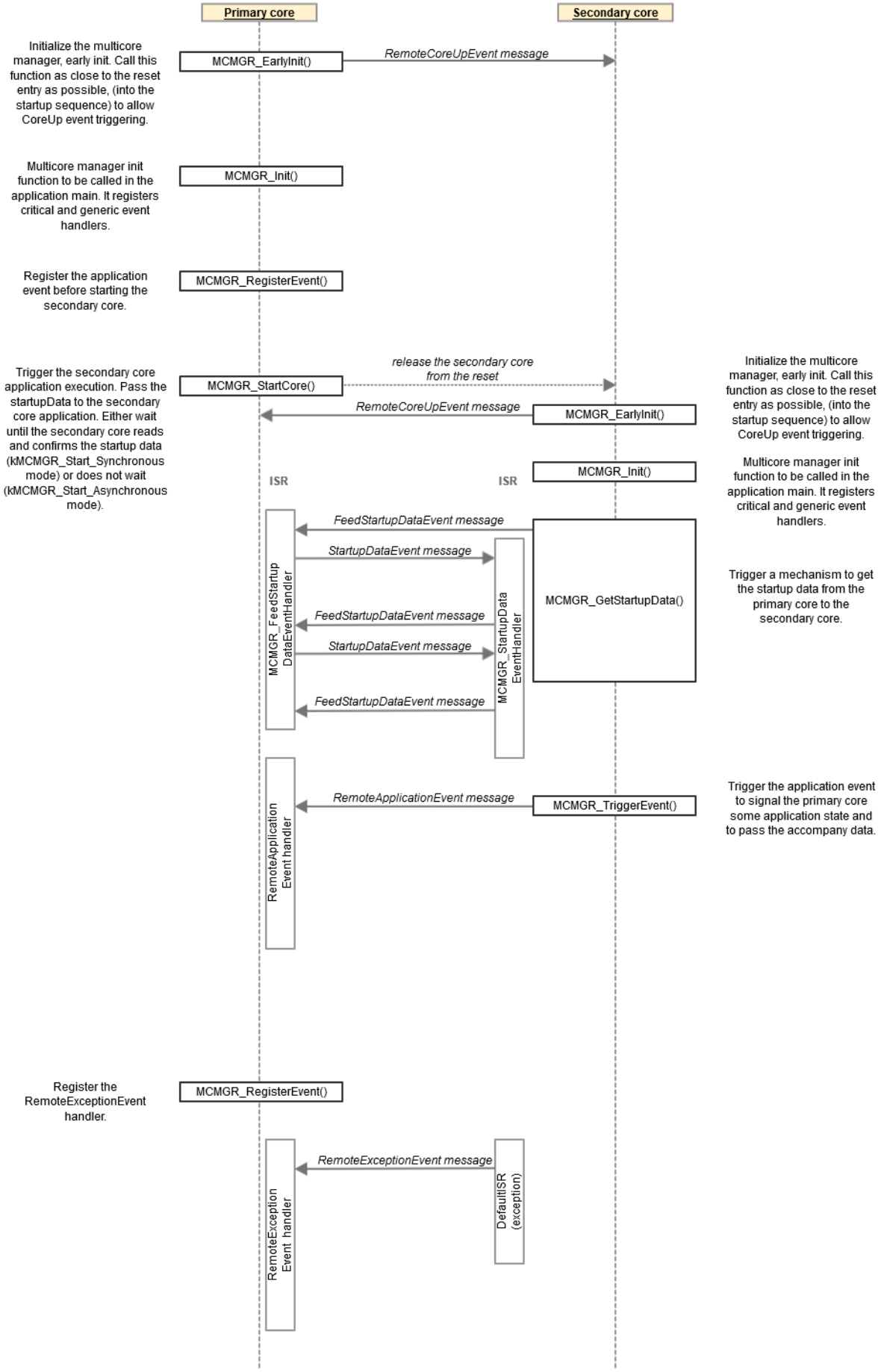
(continued from previous page)

```
    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpmsg_lite_base address␣
↪as startup data, starting synchronously. */
    MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpmsg_lite_
↪base, kMCMGR_Start_Synchronous);

    /* Wait until the secondary core application signals the rpmsg remote has been initialized and is ready to␣
↪communicate. */
    while(APP_RPMSG_READY_EVENT_DATA != RPMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

#define APP_RPMSG_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_RPMSG_
↪READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPMSG_
↪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.


Primary core
Secondary core
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_EarlyInit()
RemoteCoreUpEvent message
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
MCMGR_Init()
Register the application event before starting the secondary core.
MCMGR_RegisterEvent()
Trigger the secondary core application execution. Pass the startupData to the secondary core application. Either wait until the secondary core reads and confirms the startup data (kMCMGR_Start_Synchronous mode) or does not wait (kMCMGR_Start_Asynchronous mode).
MCMGR_StartCore()
release the secondary core from the reset
RemoteCoreUpEvent message
MCMGR_EarlyInit()
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_Init()
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
ISR
ISR
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
MCMGR_FeedStartup DataEventHandler
MCMGR_StartupData EventHandler
MCMGR_GetStartupData()
Trigger a mechanism to get the startup data from the primary core to the secondary core.
RemoteApplicationEvent message
MCMGR_TriggerEvent()
RemoteApplication Event handler
Trigger the application event to signal the primary core some application state and to pass the accompany data.
Register the RemoteExceptionEvent handler.
MCMGR_RegisterEvent()
RemoteExceptionEvent message
RemoteException Event handler
DefaultISR (exception)

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.

[v5.0.2]

Added

- Added gcov options and configs to support mcmgr code coverage
- Added new test_weak_mu_isr testcase for devices with MU peripheral
- Added new test_heartbeat testcase showing heartbeat mechanism between primary and secondary cores using the MCMGR

v5.0.1

Added

- Added frdmimxrt1186 unit testing

Changed

- [KW43] Rename core#1 reset control register

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.
- Fix path to mcmgr headers directory in doxyfile

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);
```

(continues on next page)

(continued from previous page)

```
    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 */
    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
```

(continues on next page)

(continued from previous page)

```
{
    /* 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 */
    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 by 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**, ...
- **Binson** - https://www.gnu.org/software/bison/
- **Flex** - https://github.com/westes/flex/

Requirements when using Make:

- **Make**
- **C/C++ compiler** - **GCC**, **CLANG**, ...
- **Binson** - 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.

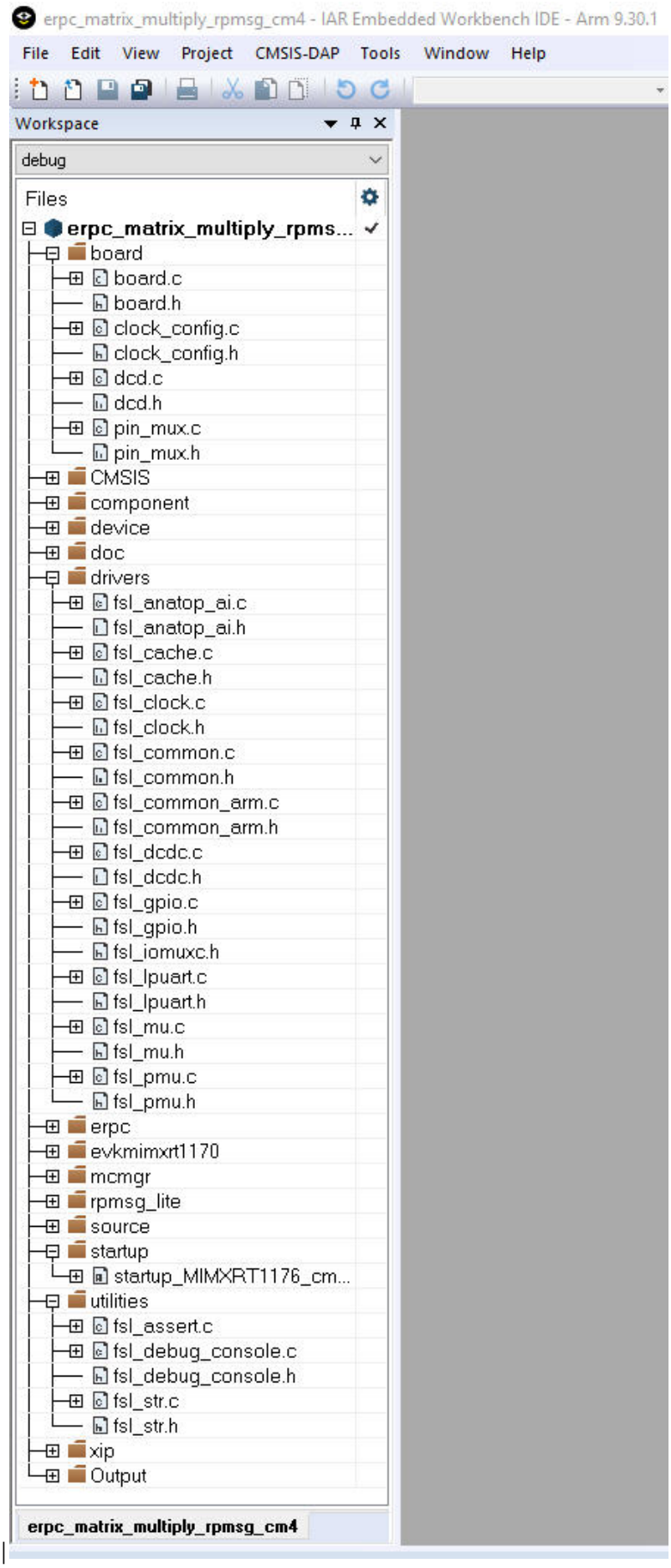
Multicore server application The “Matrix multiply” eRPC server project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/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>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

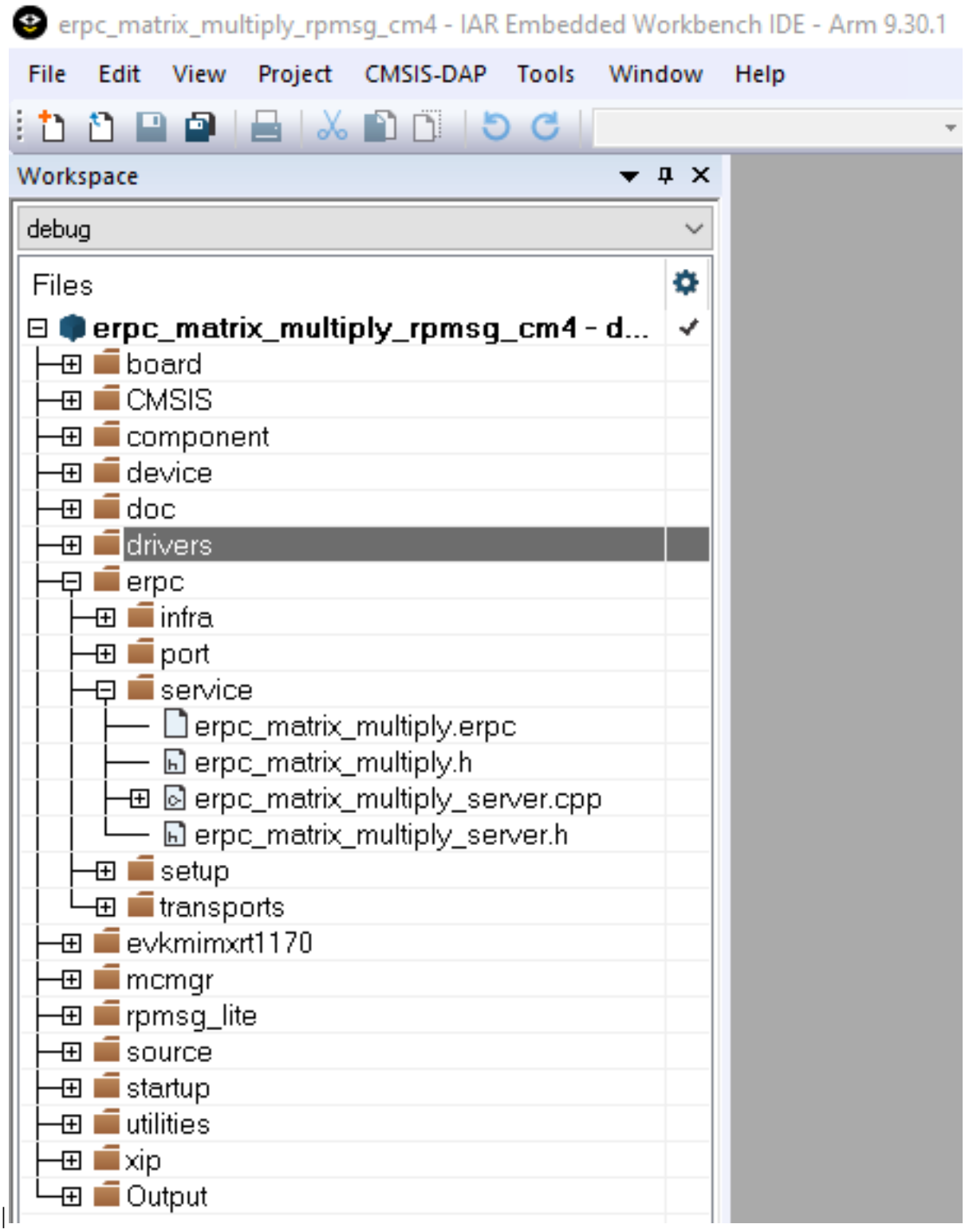
Parent topic: Multicore server application

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/s



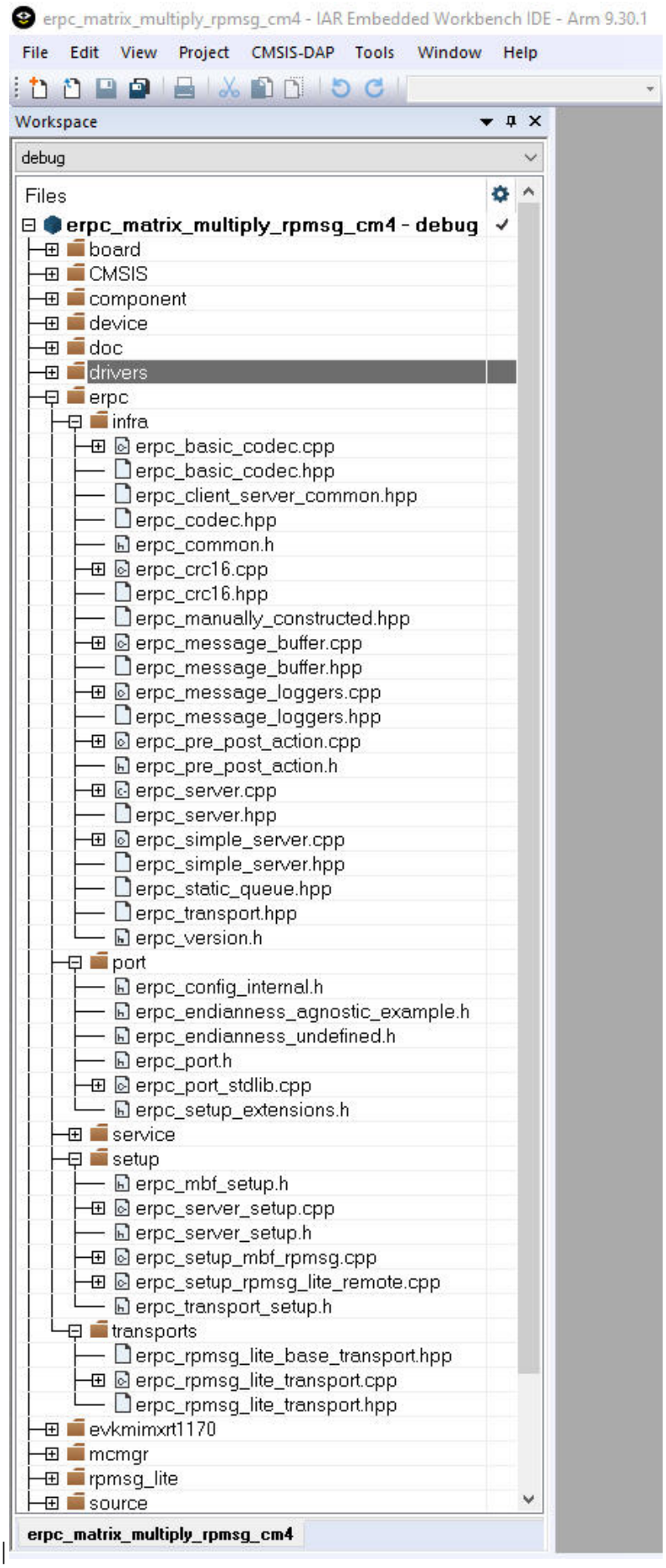
Parent topic: Multicore server application

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 needs 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 needs 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 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 need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

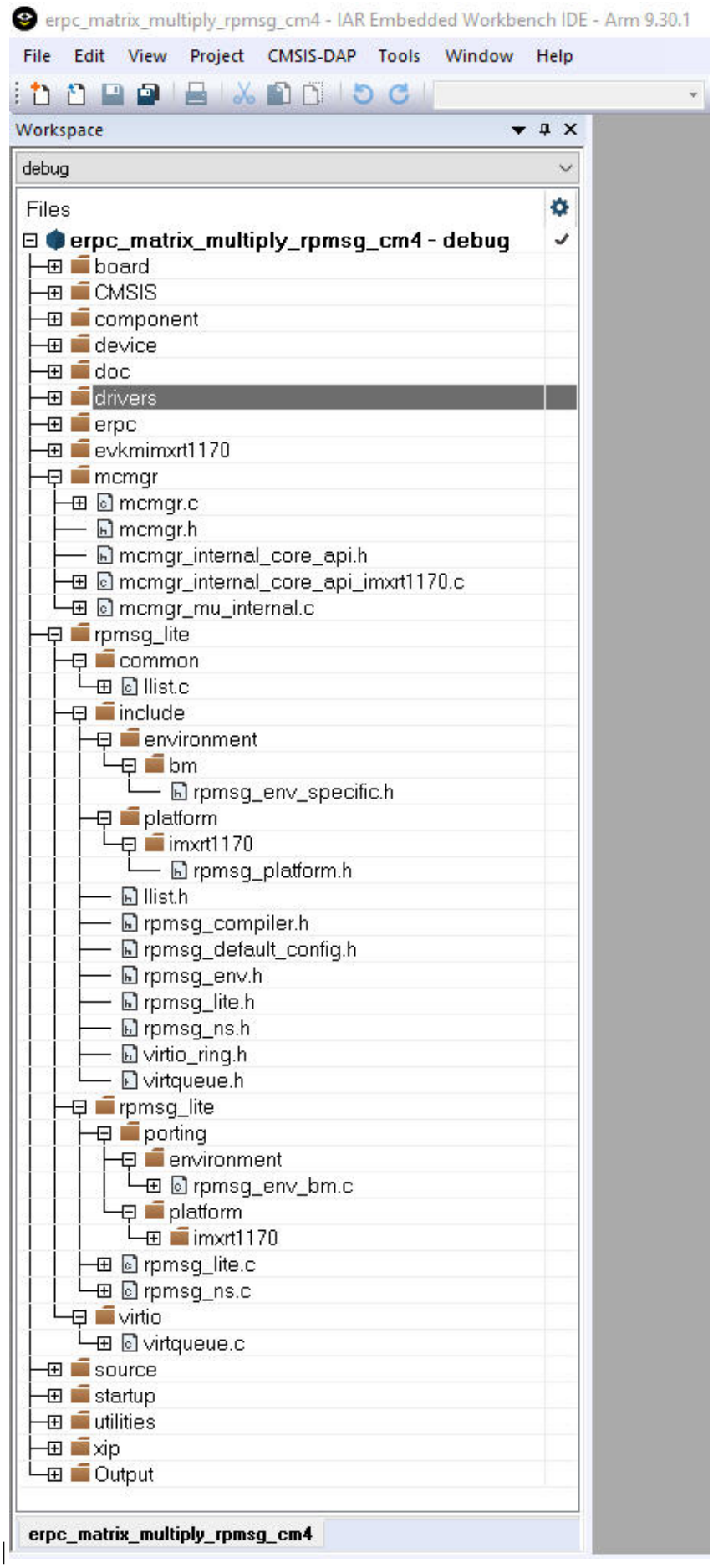
Parent topic: Multicore server application

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/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

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

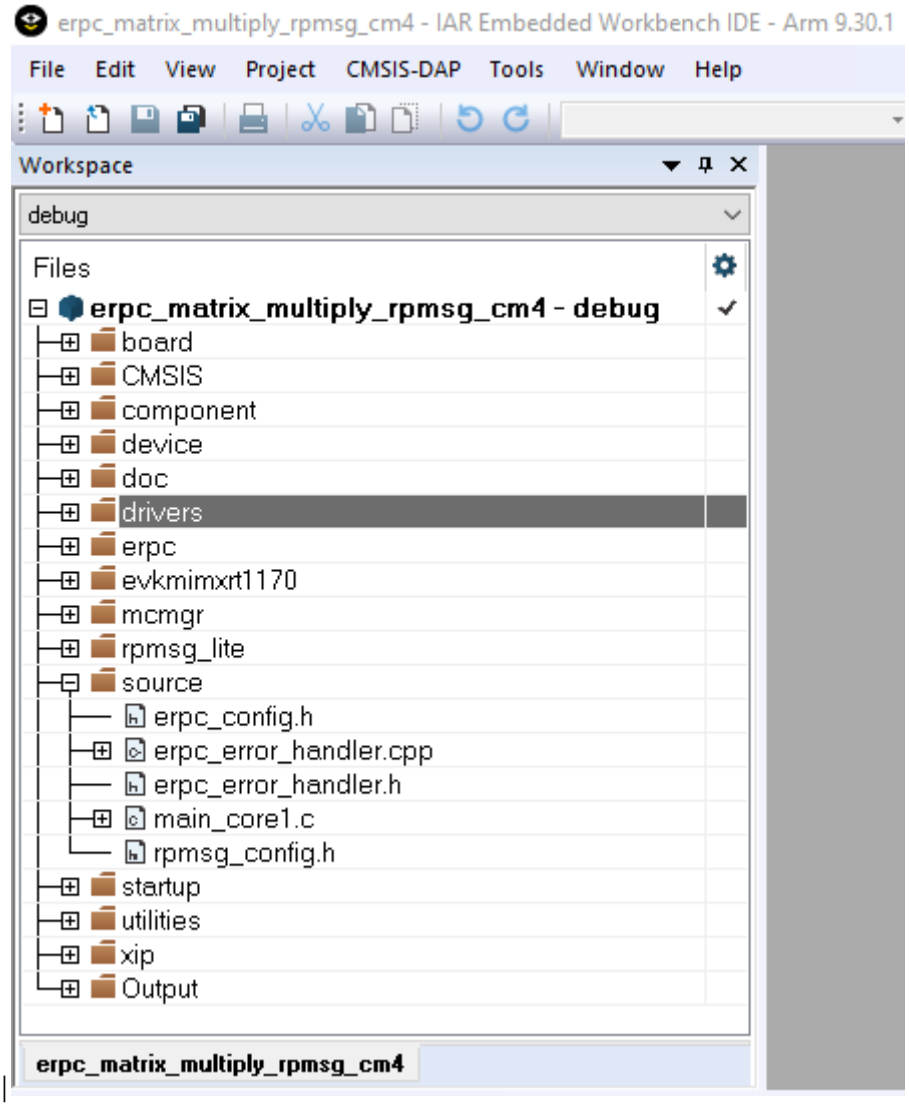
The main_core1.c file contains two functions:

- 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()
{
  ...
  /* RPMsg-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*

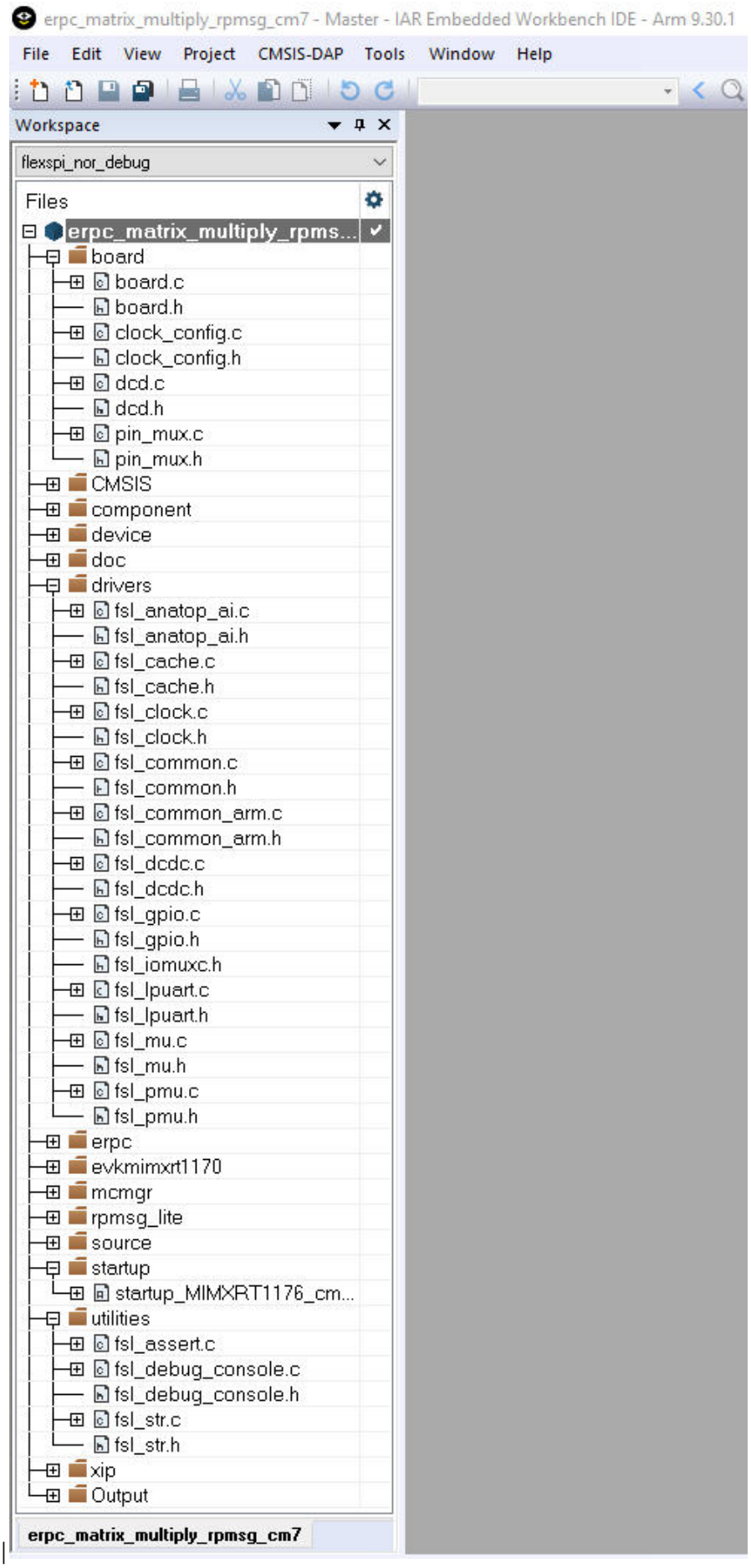
Multicore client application The "Matrix multiply" eRPC client project is located in the following folder:

<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>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

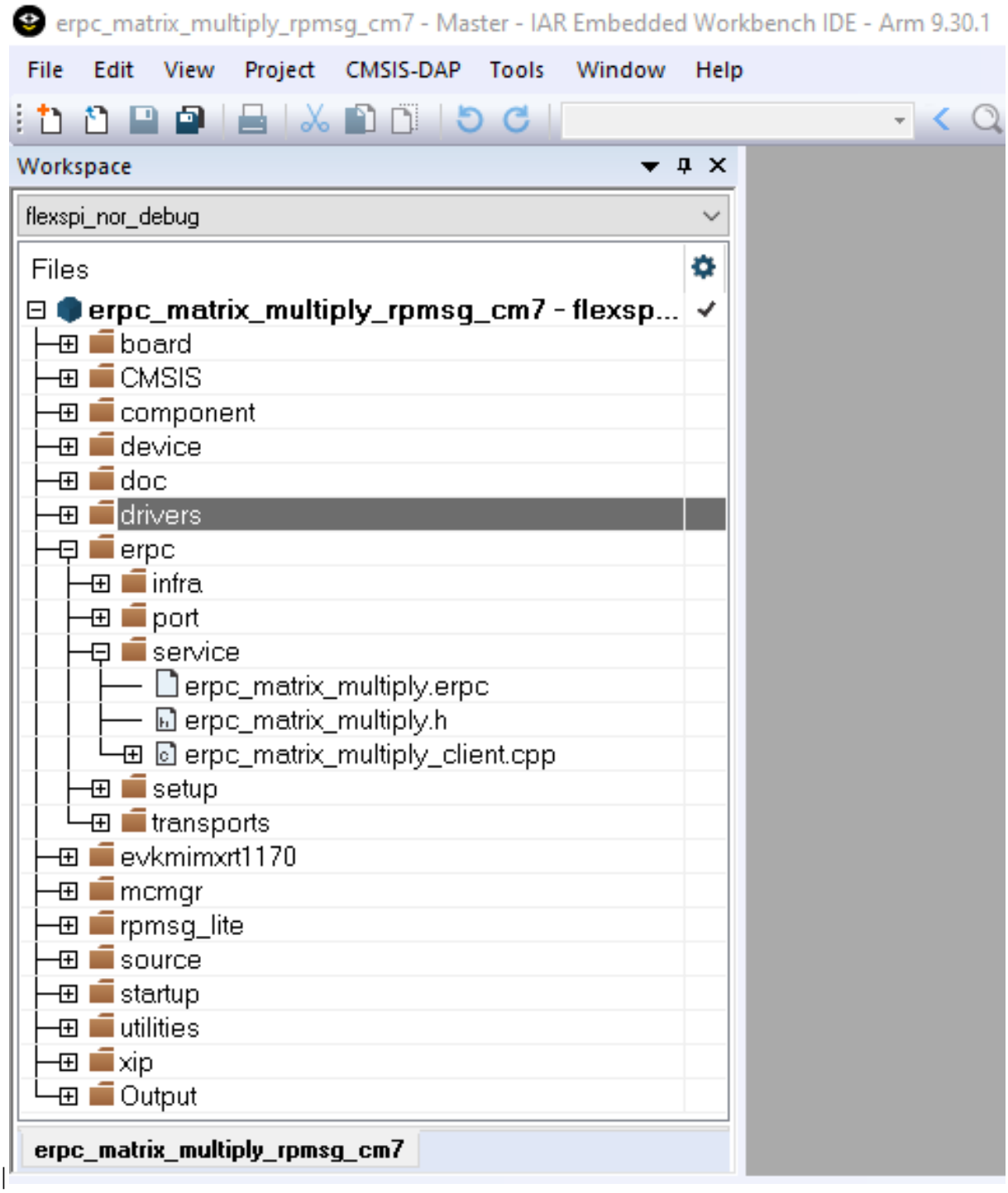
|

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.

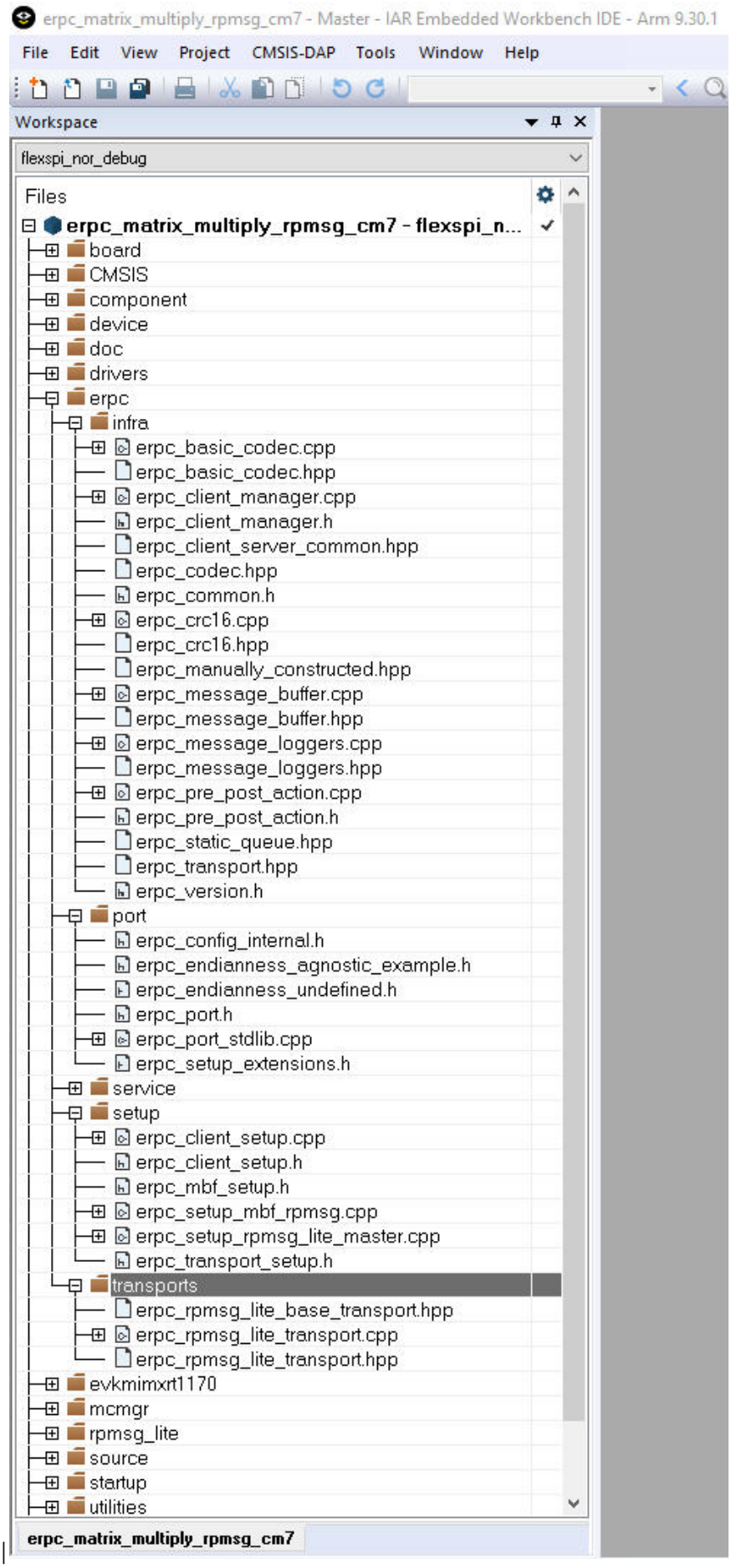
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

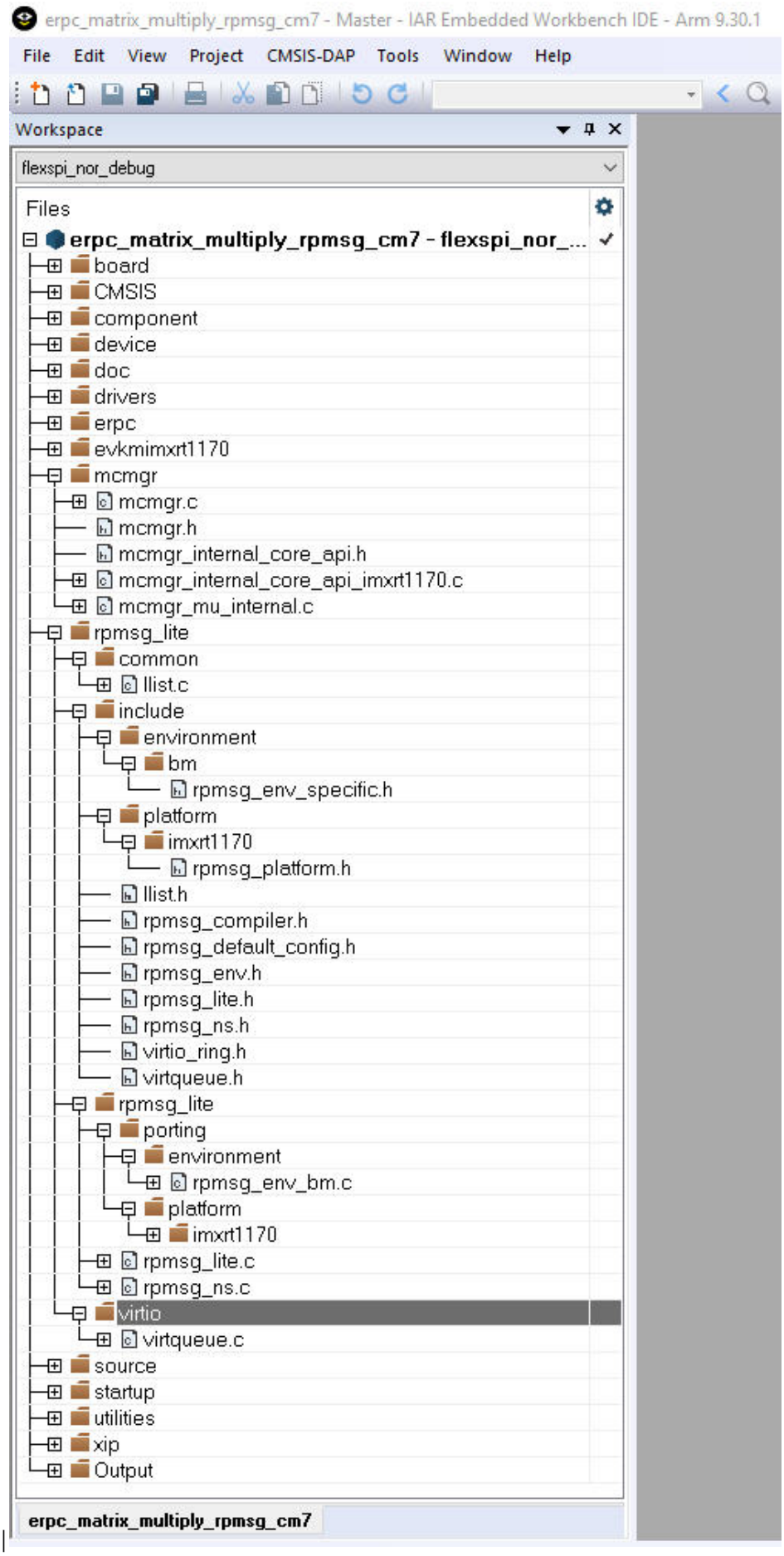
Parent topic: Multicore client application

Client 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/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

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.

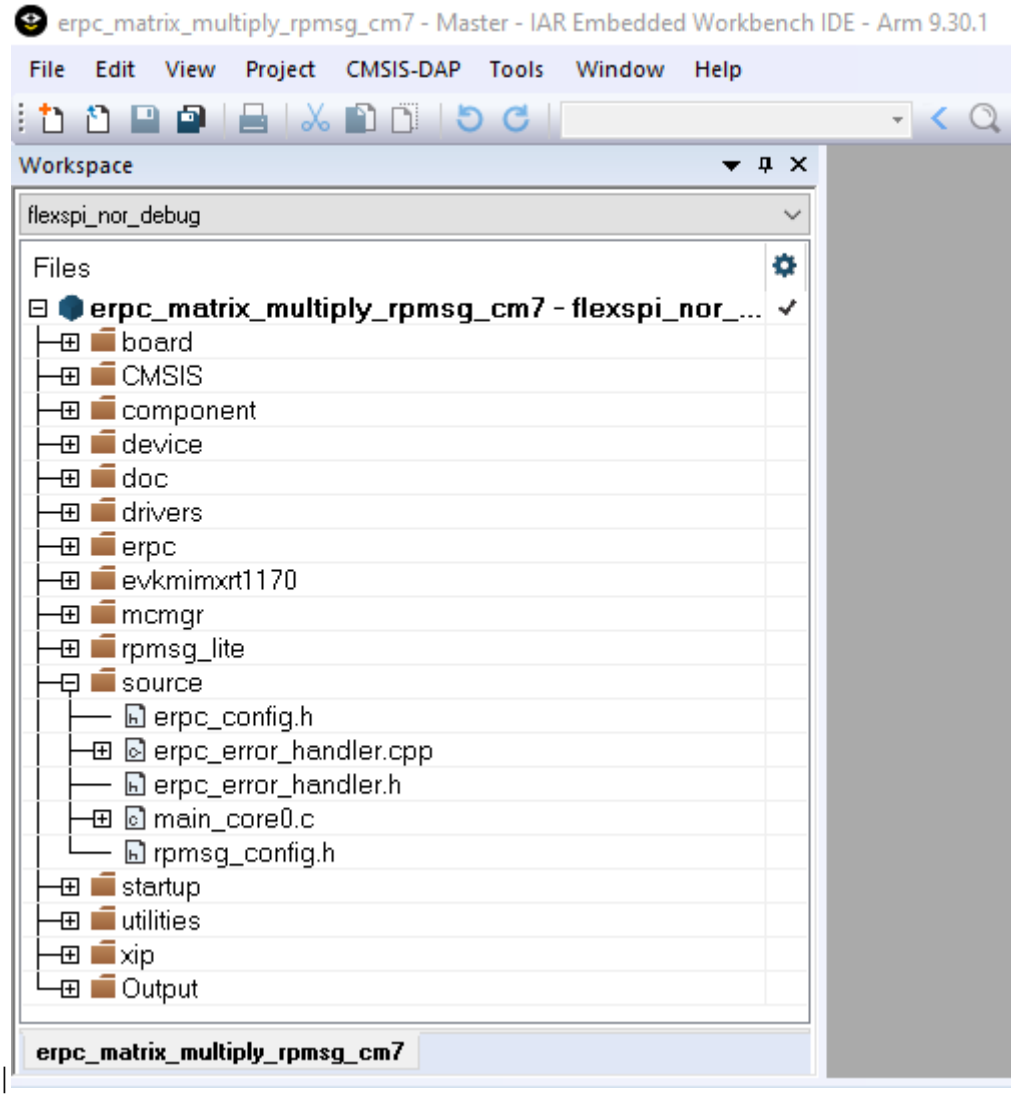
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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 occured 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 RPMsg-Lite). The RPMsg-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/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/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 RPMsg-Lite). The RPMsg-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:

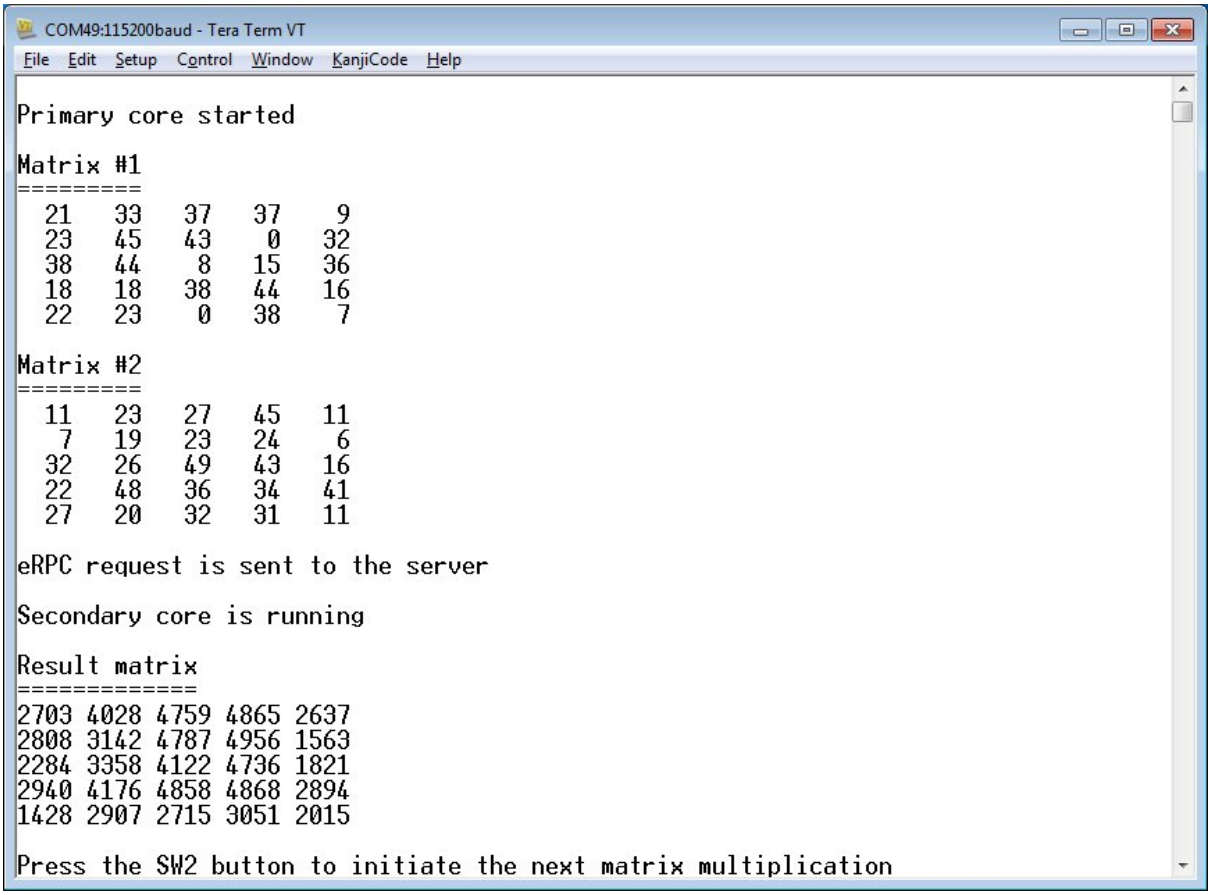
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&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 occured 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:



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 comunicates with the eRPC server that runs on anothe 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/*| |RPMsg-Lite source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/*|

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 is 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/s

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/erp
|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/serv* 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.

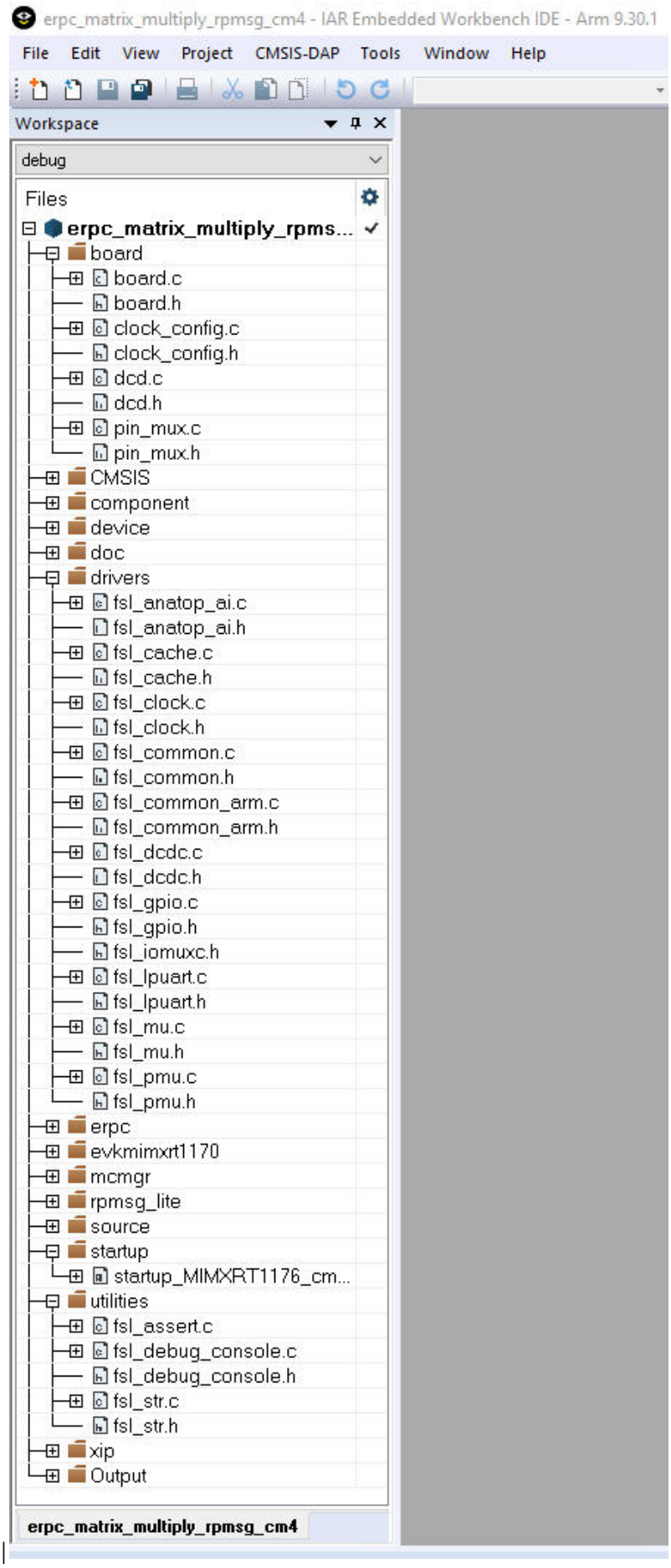
Multicore server application The “Matrix multiply” eRPC server project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/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>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

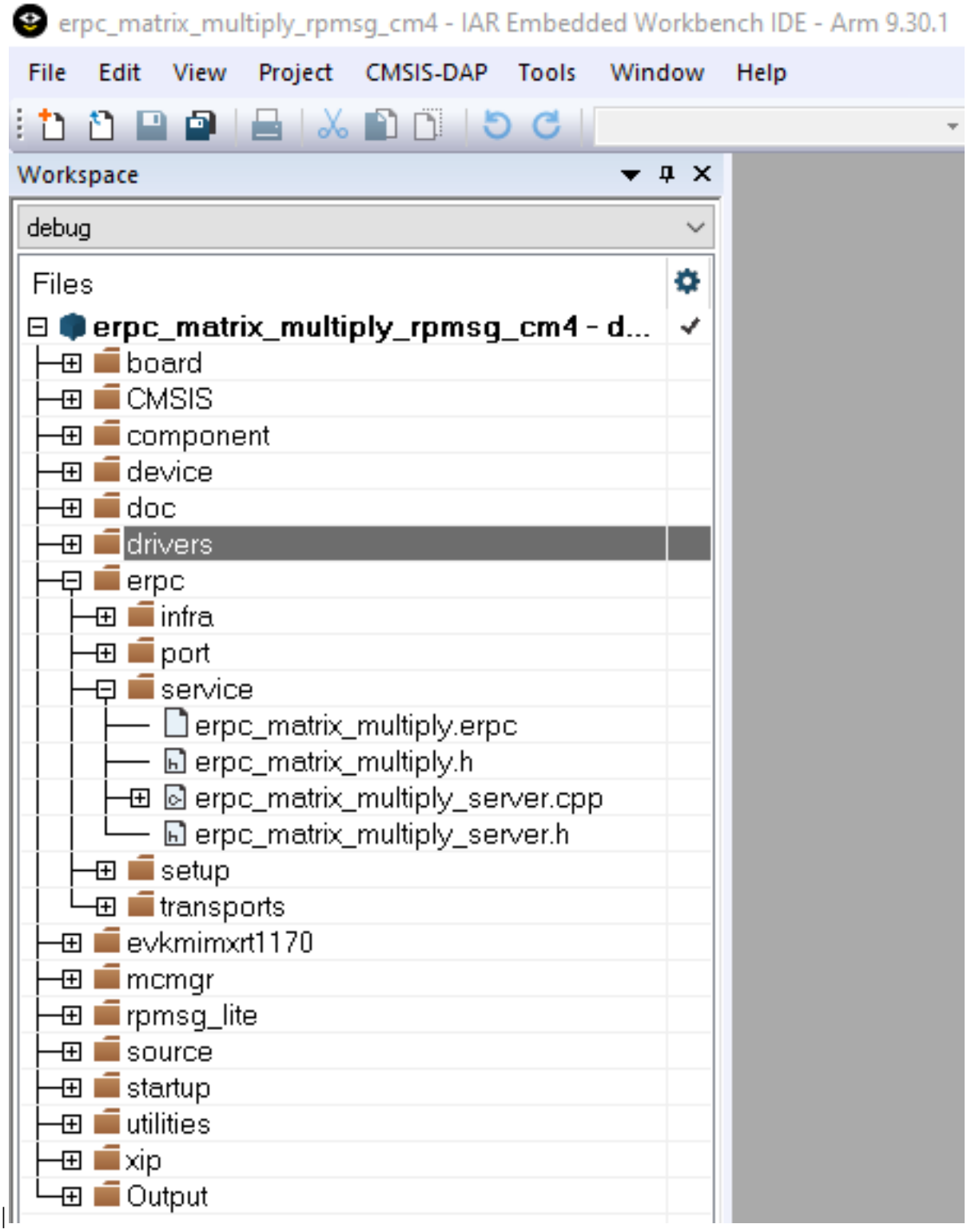
Parent topic: Multicore server application

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/s



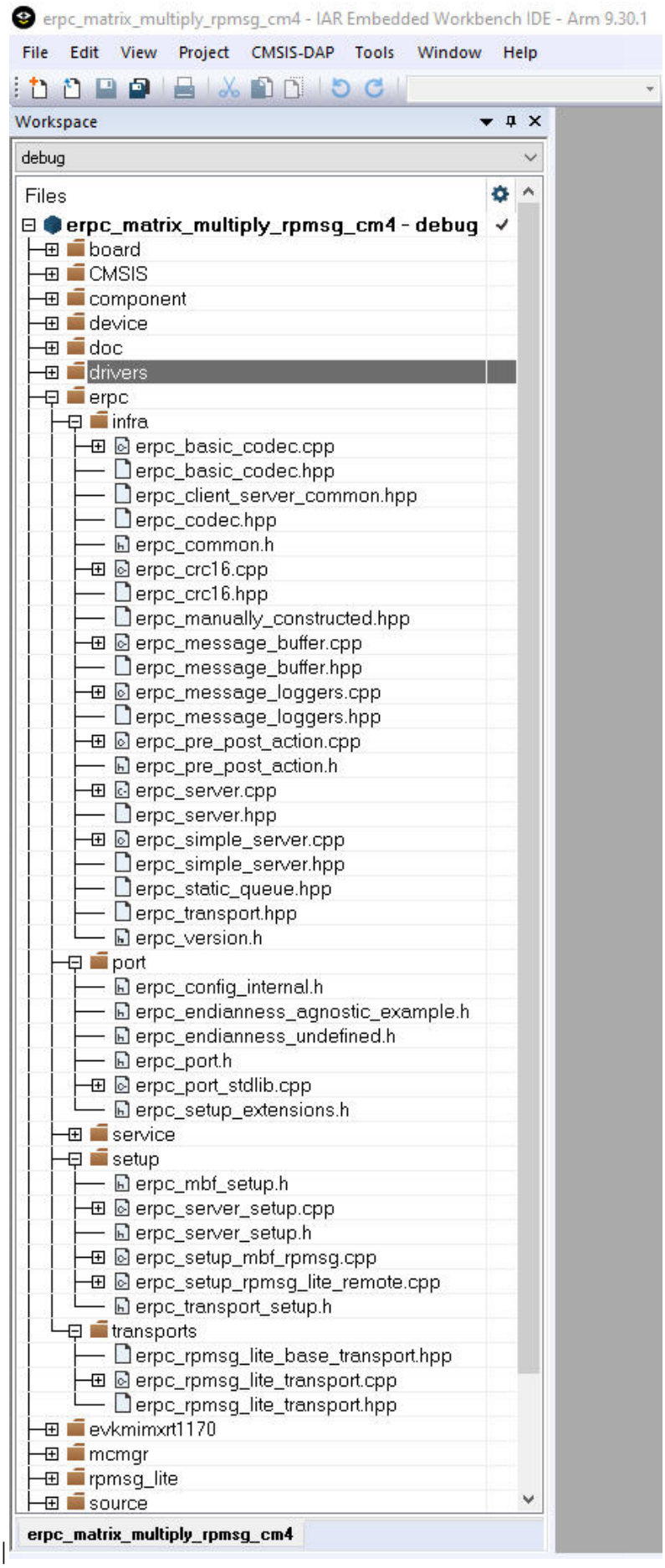
Parent topic: Multicore server application

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 needs 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 needs 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 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 need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

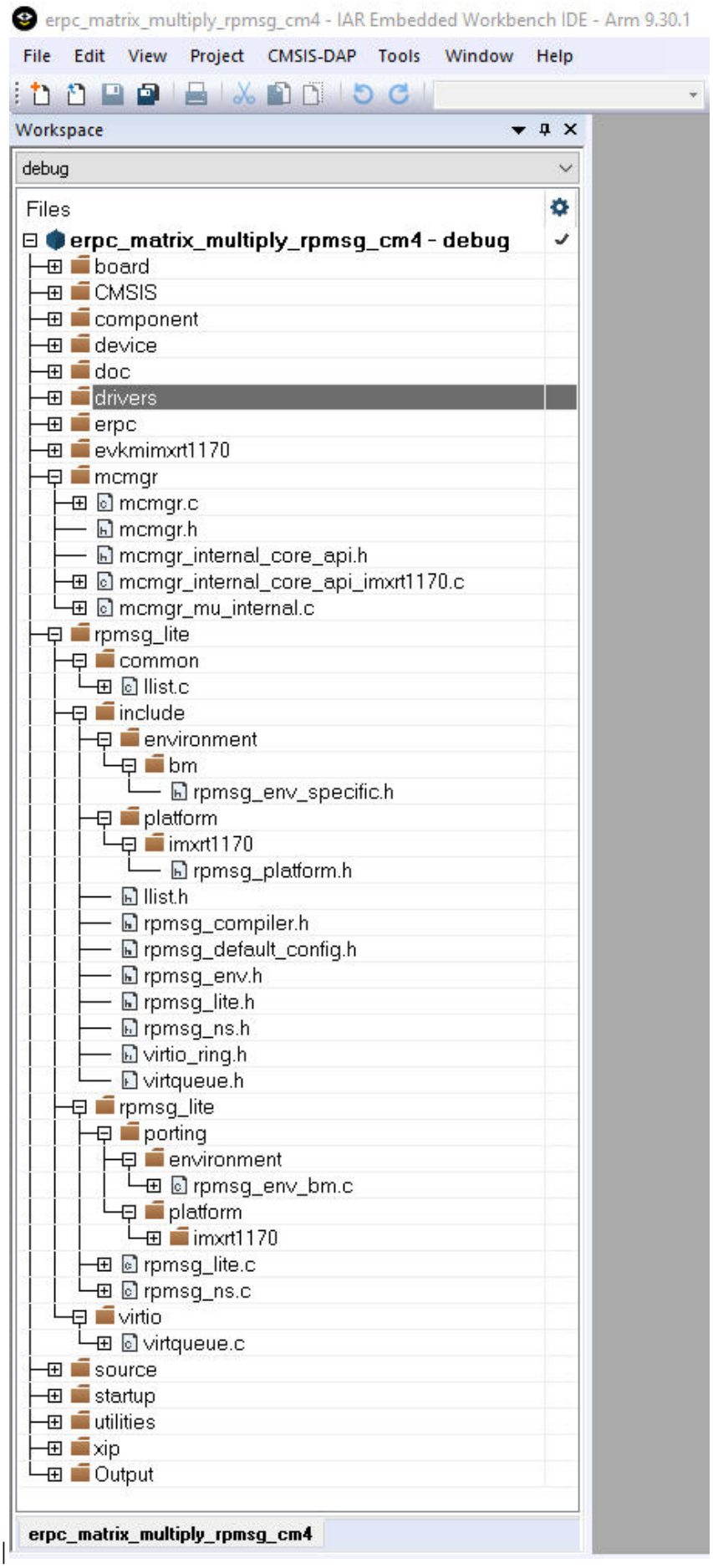
Parent topic: Multicore server application

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/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

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

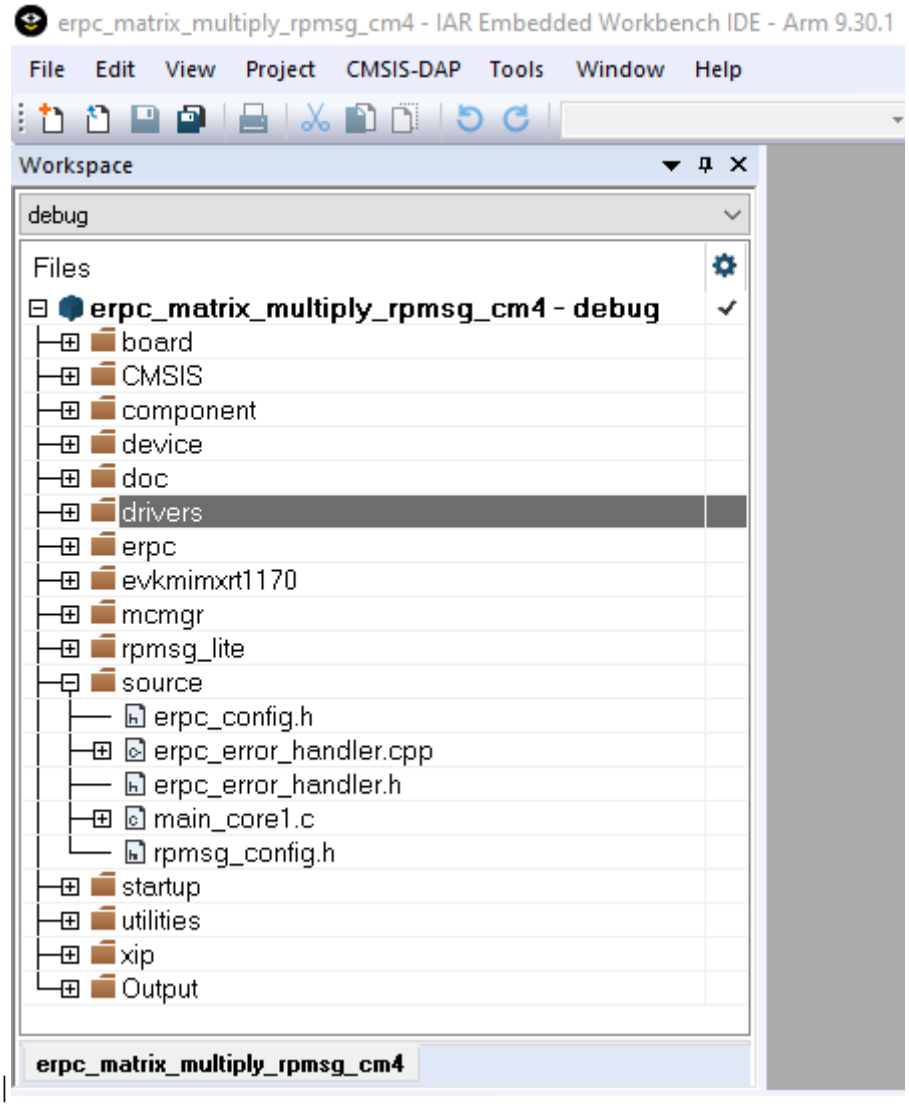
The main_core1.c file contains two functions:

- 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()
{
  ...
  /* RPMsg-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*

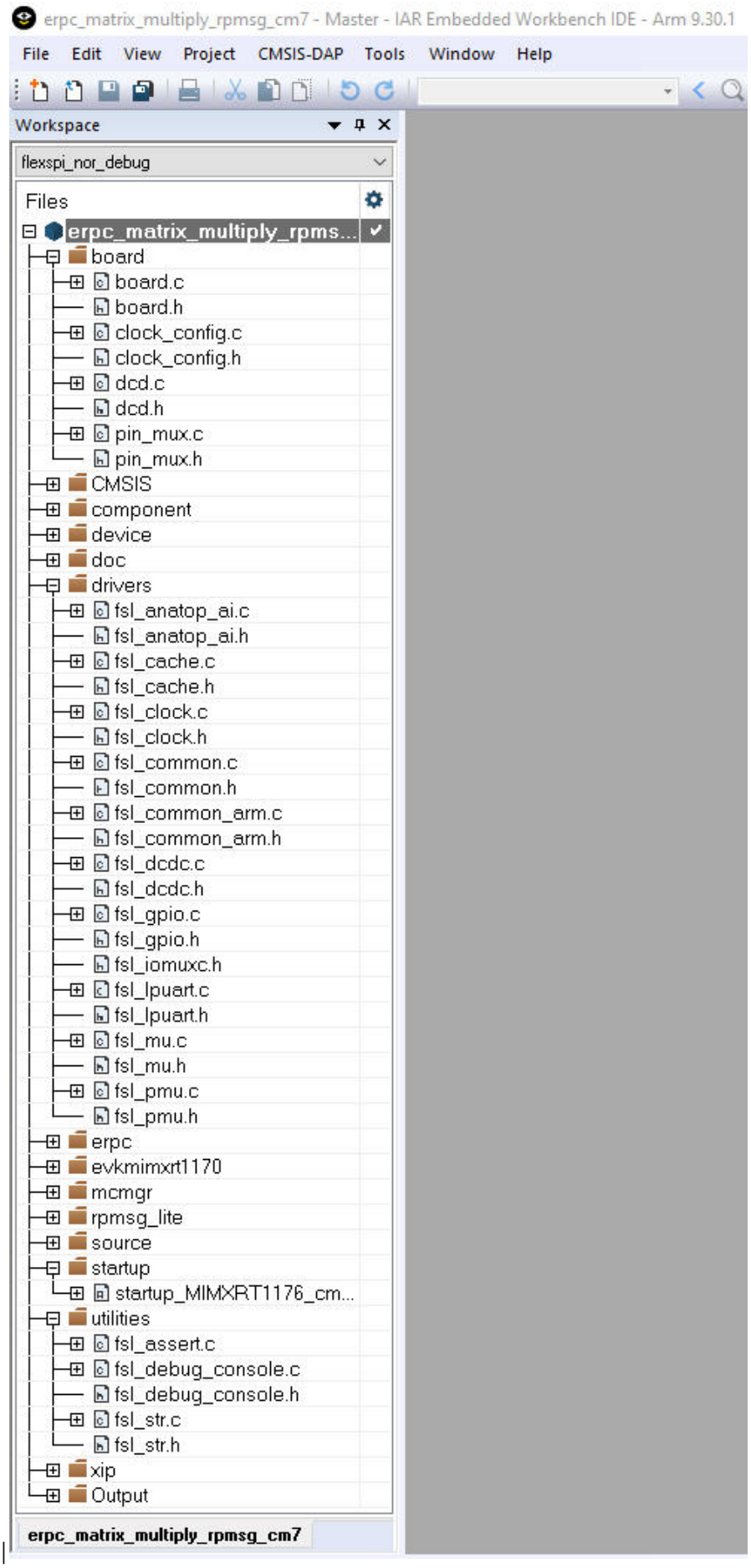
Multicore client application The "Matrix multiply" eRPC client project is located in the following folder:

<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>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

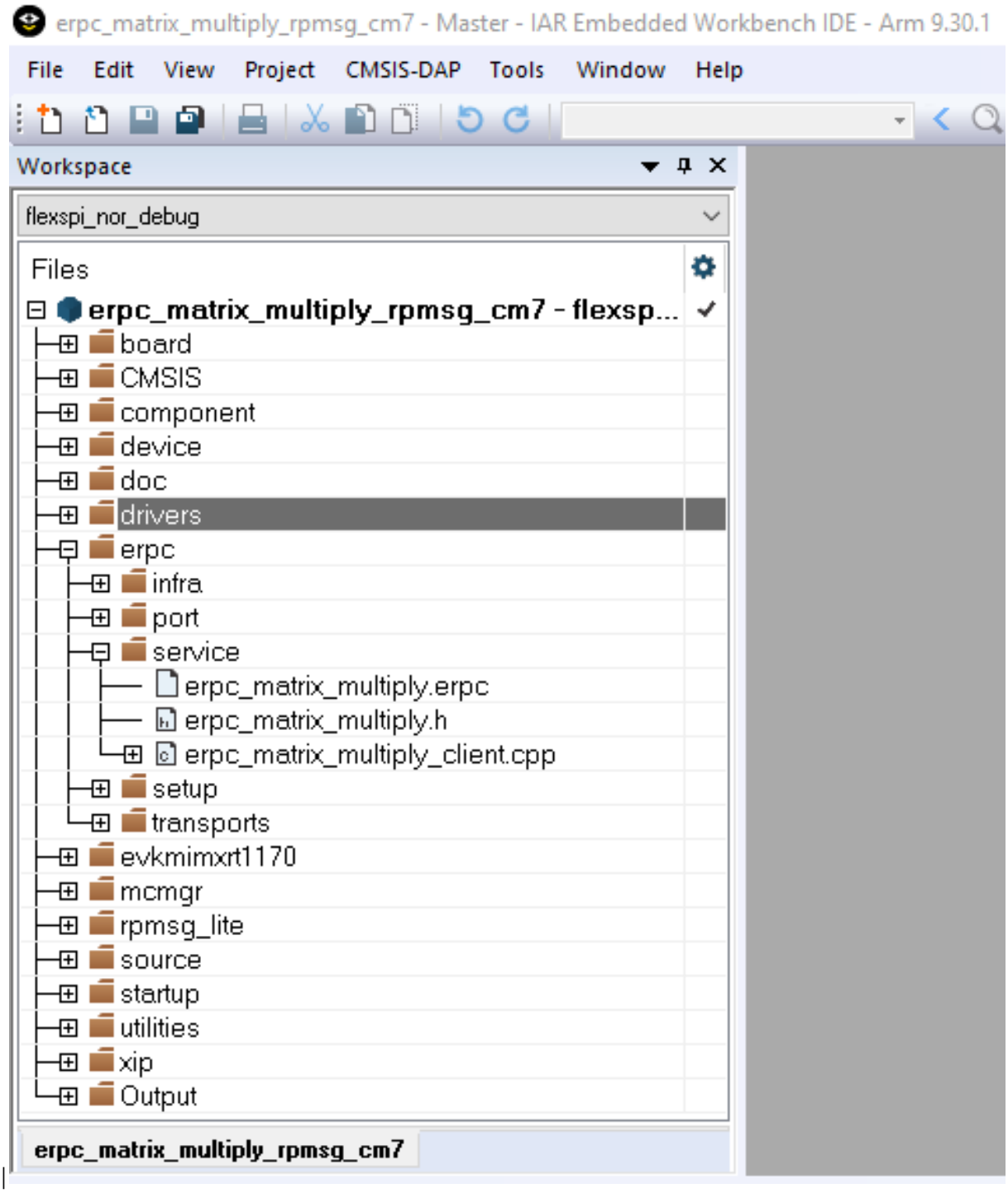
|

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.

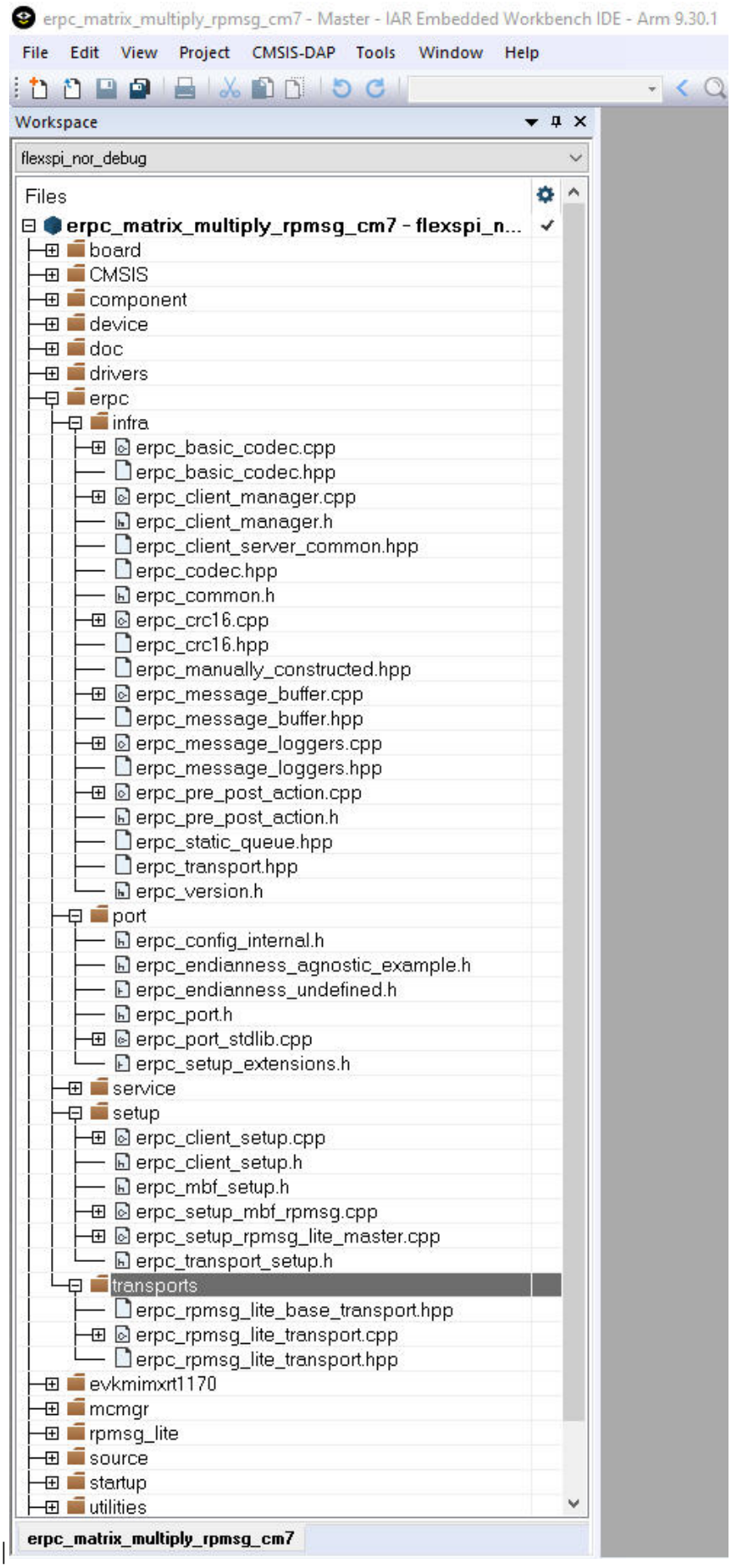
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

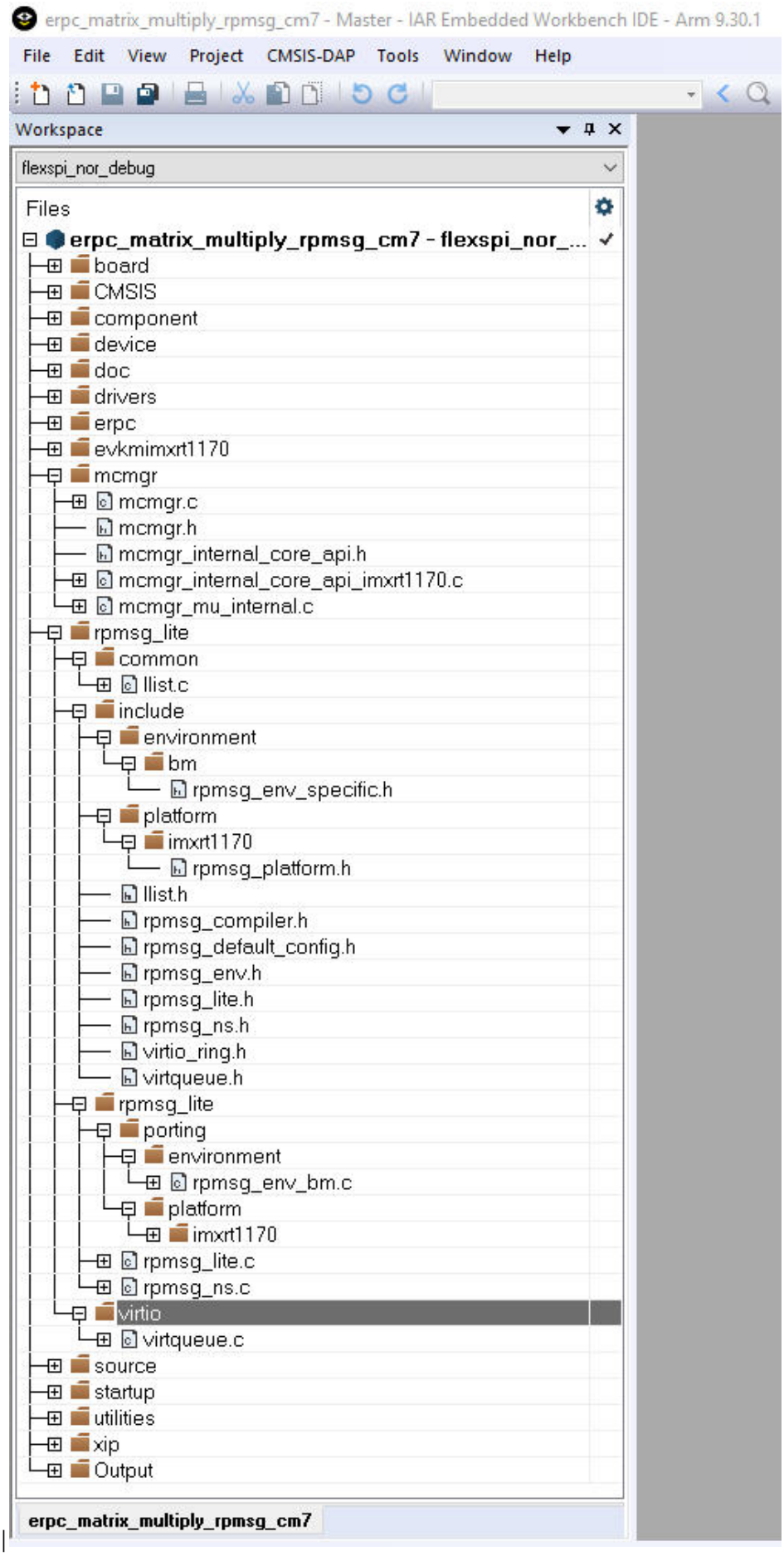
Parent topic: Multicore client application

Client 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/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

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.

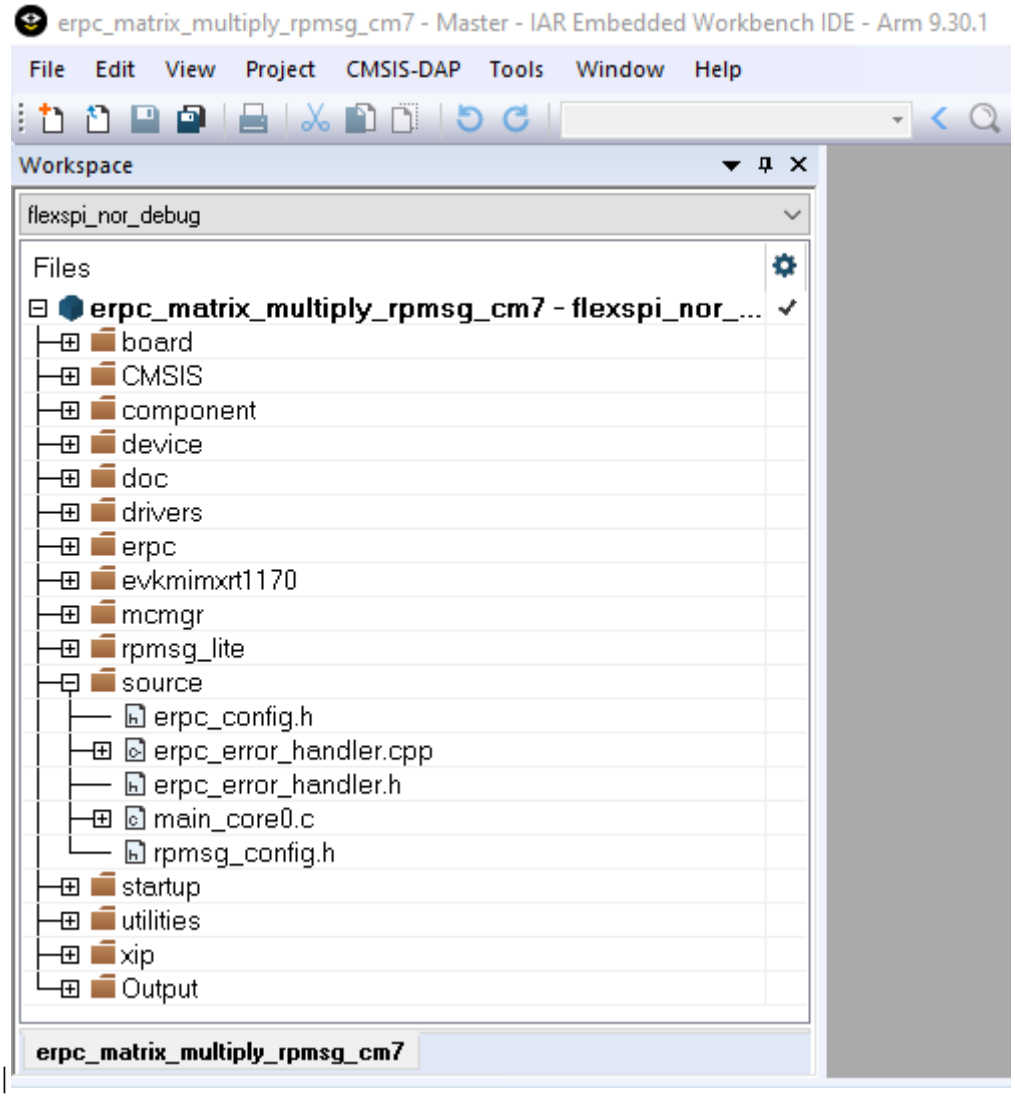
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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 occured 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 RPMsg-Lite). The RPMsg-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/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/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 RPMsg-Lite). The RPMsg-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:

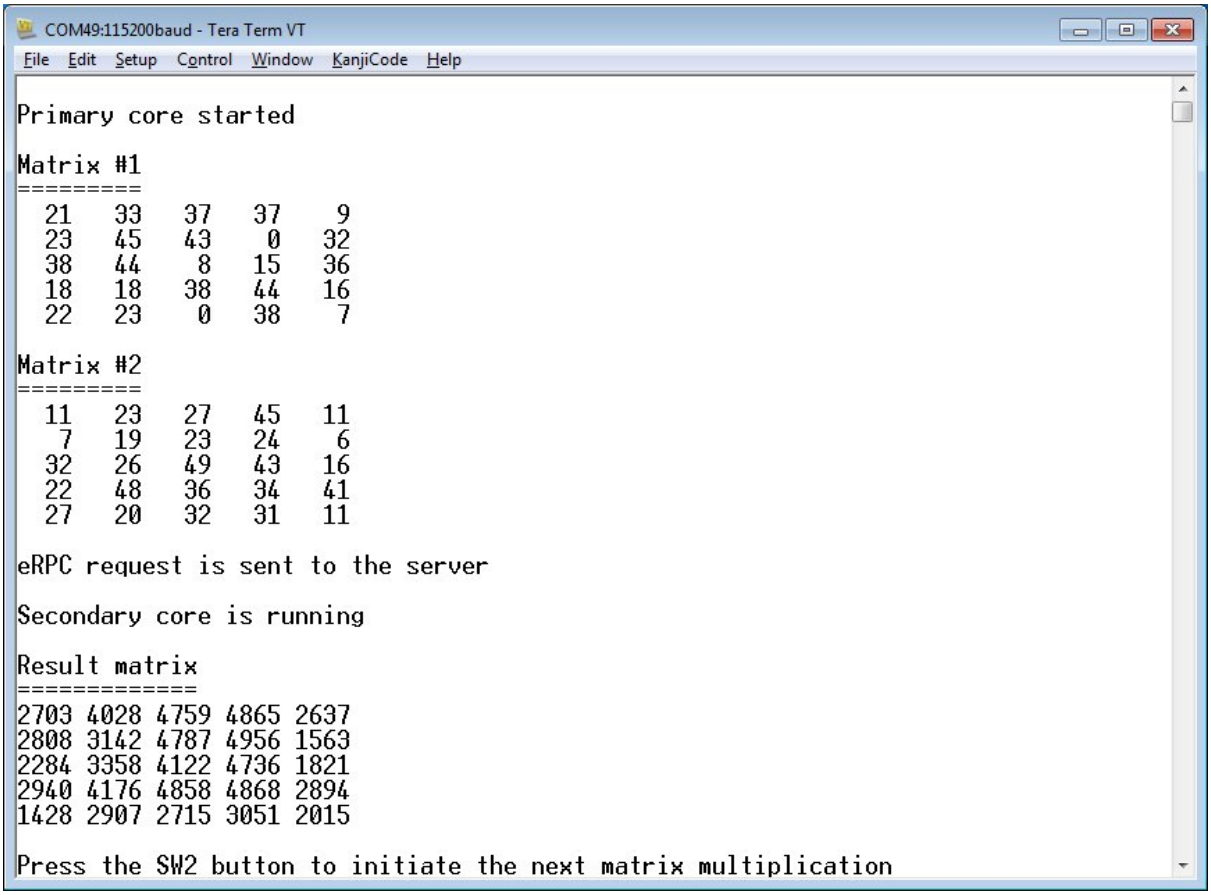
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&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 occured 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:



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.

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

Added

Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.

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 MessageBuffer, 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 LIBUSBSIO.GetNumPorts API use with libusbsio 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: MUTransport 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 distuingish which file can and cannot by 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 for 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 LIBUSBSIO 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 trasnport 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 ChangeLog

FreeRTOS kernel Readme

4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

4.1.5 corejson

JSON parser.

Readme

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

4.1.7 corepkcs11

PKCS #11 key management library.

Readme

4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme