MCUXpresso SDK Documentation

Release 26.06.00

NXP
Jun 24, 2026

Table of contents

1 EVK-MCIMX7ULP **3**
- 1.1 Overview . . . 3
- 1.2 Getting Started with MCUXpresso SDK Package . . . 3
  - 1.2.1 Getting Started with Package . . . 3
- 1.3 Getting Started with MCUXpresso SDK GitHub . . . 14
  - 1.3.1 Getting Started with MCUXpresso SDK Repository . . . 14
- 1.4 Release Notes . . . 39
  - 1.4.1 MCUXpresso SDK Release Notes . . . 39
- 1.5 ChangeLog . . . 44
  - 1.5.1 MCUXpresso SDK Changelog . . . 44
- 1.6 Driver API Reference Manual . . . 131
- 1.7 Middleware Documentation . . . 131
  - 1.7.1 Multicore . . . 131
  - 1.7.2 FreeMASTER . . . 131
  - 1.7.3 FreeRTOS . . . 131

2 MCIMX7U5 **133**
- 2.1 ACMP: Analog Comparator Driver . . . 133
- 2.2 CACHE: LMEM CACHE Memory Controller . . . 139
- 2.3 Clock . . . 143
- 2.4 CRC: Cyclic Redundancy Check Driver . . . 175
- 2.5 DAC12: 12-bit Digital-to-Analog Converter Driver . . . 178
- 2.6 DMAMUX: Direct Memory Access Multiplexer Driver . . . 184
- 2.7 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 185
- 2.8 EWM: External Watchdog Monitor Driver . . . 205
- 2.9 FGPIO Driver . . . 208
- 2.10 FlexIO: FlexIO Driver . . . 209
- 2.11 FlexIO Driver . . . 209
- 2.12 FlexIO eDMA I2S Driver . . . 227
- 2.13 FlexIO eDMA SPI Driver . . . 230
- 2.14 FlexIO eDMA UART Driver . . . 234
- 2.15 FlexIO I2C Master Driver . . . 237
- 2.16 FlexIO I2S Driver . . . 245
- 2.17 FlexIO SPI Driver . . . 256
- 2.18 FlexIO UART Driver . . . 269
- 2.19 GPIO: General-Purpose Input/Output Driver . . . 279
- 2.20 GPIO Driver . . . 281
- 2.21 Iomuxc_driver . . . 285
- 2.22 Common Driver . . . 314
- 2.23 LLWU: Low-Leakage Wakeup Unit Driver . . . 328
- 2.24 LPADC: 12-bit SAR Analog-to-Digital Converter Driver . . . 332
- 2.25 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 354
- 2.26 LPI2C Master Driver . . . 355
- 2.27 LPI2C Master DMA Driver . . . 369
- 2.28 LPI2C Slave Driver . . . 372
- 2.29 LPIT: Low-Power Interrupt Timer . . . 382

2.30 LPSPI: Low Power Serial Peripheral Interface . . . 389
2.31 LPSPI Peripheral driver . . . 389
2.32 LPSPI eDMA Driver . . . 410
2.33 LPTMR: Low-Power Timer . . . 417
2.34 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 423
2.35 LPUART Driver . . . 423
2.36 LPUART eDMA Driver . . . 442
2.37 LTC: LP Trusted Cryptography . . . 445
2.38 LTC AES driver . . . 447
2.39 LTC DES driver . . . 452
2.40 LTC HASH driver . . . 460
2.41 LTC PKHA driver . . . 462
2.42 LTC Blocking APIs . . . 471
2.43 Ltc_edma_driver . . . 471
2.44 LTC eDMA AES driver . . . 472
2.45 Ltc_edma_driver_des . . . 474
2.46 MCM: Miscellaneous Control Module . . . 485
2.47 MSMC: Multicore System Mode Controller . . . 490
2.48 MU: Messaging Unit . . . 499
2.49 PMC0: Power Management Controller . . . 507
2.50 PORT: Port Control and Interrupts . . . 521
2.51 QSPI: Quad Serial Peripheral Interface . . . 528
2.52 Quad Serial Peripheral Interface Driver . . . 528
2.53 Quad Serial Peripheral Interface EDMA Driver . . . 541
2.54 SAI: Serial Audio Interface . . . 544
2.55 SAI Driver . . . 544
2.56 SAI EDMA Driver . . . 570
2.57 SEMA42: Hardware Semaphores Driver . . . 577
2.58 SNVS: Secure Non-Volatile Storage . . . 580
2.59 Secure Non-Volatile Storage High-Power . . . 580
2.60 Secure Non-Volatile Storage Low-Power . . . 589
2.61 TPM: Timer PWM Module . . . 598
2.62 TRGMUX: Trigger Mux Driver . . . 615
2.63 TRNG: True Random Number Generator . . . 616
2.64 TSTMR: Timestamp Timer Driver . . . 621
2.65 WDOG32: 32-bit Watchdog Timer . . . 622
2.66 XRDC: Extended Resource Domain Controller . . . 628

3 Middleware **647**
3.1 Motor Control . . . 647
3.1.1 FreeMASTER . . . 647
3.2 Multicore SDK . . . 685
3.2.1 Quick links . . . 685
3.2.2 Components . . . 686

4 RTOS **777**
4.1 FreeRTOS . . . 777
4.1.1 FreeRTOS kernel . . . 777
4.1.2 FreeRTOS drivers . . . 779
4.1.3 backoffalgorithm . . . 779
4.1.4 corehttp . . . 779
4.1.5 corejson . . . 779
4.1.6 coremqtt . . . 780
4.1.7 corepkcs11 . . . 780
4.1.8 freertos-plus-tcp . . . 780

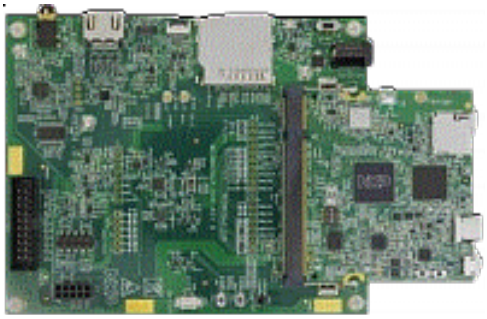
This documentation contains information specific to the evkmcimx7ulp board.

Chapter 1

EVK-MCIMX7ULP

1.1 Overview

The i.MX7ULP family of boards provides a powerful and flexible development system for NXP's Cortex-M4 MCUs.



MCU device and part on board is shown below:

- Device: MCIMX7U5
- PartNumber: MCIMX7U5DVP07

1.2 Getting Started with MCUXpresso SDK Package

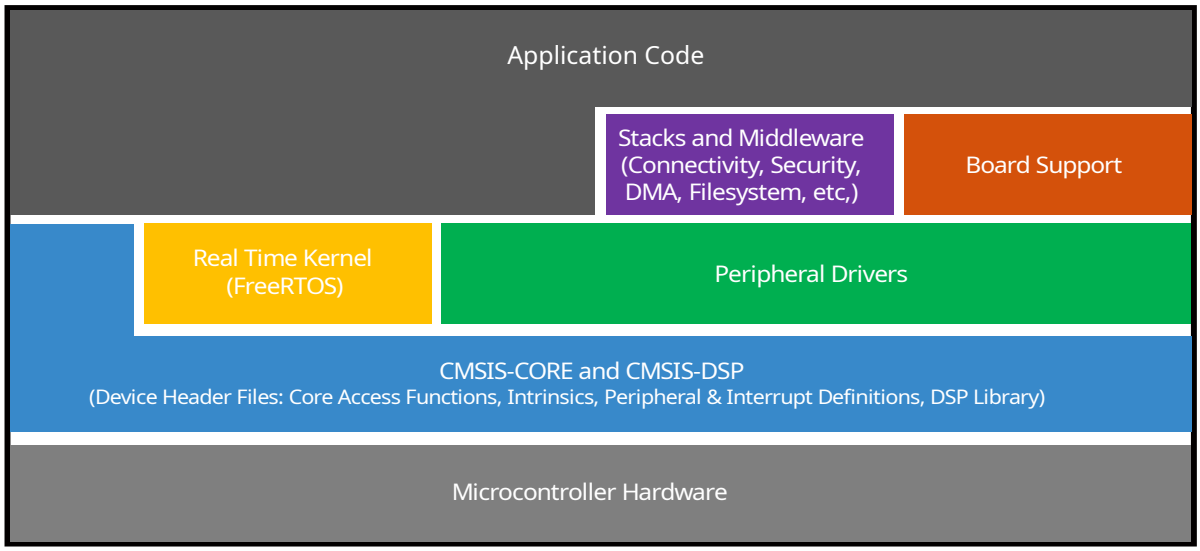
1.2.1 Getting Started with Package

Overview

The NXP MCUXpresso software and tools offer comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on general purpose, crossover, and Bluetooth-enabled MCUs from NXP. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications which can be used standalone or collaboratively with the A cores running another Operating System (such as Linux OS Kernel). 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, device stack, and various other middleware to support rapid development.

For supported toolchain versions, see the *MCUXpresso SDK Release Notes Supporting i.MX 7ULP Derivatives* (document MCUXSDKIMX7ULPRN)

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 provides example applications for development and evaluation boards. Board support packages are found inside of the top level <board_name> folder, and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each <board_name> folder there are various subfolders for each example that they contain. These include (but are not limited to):

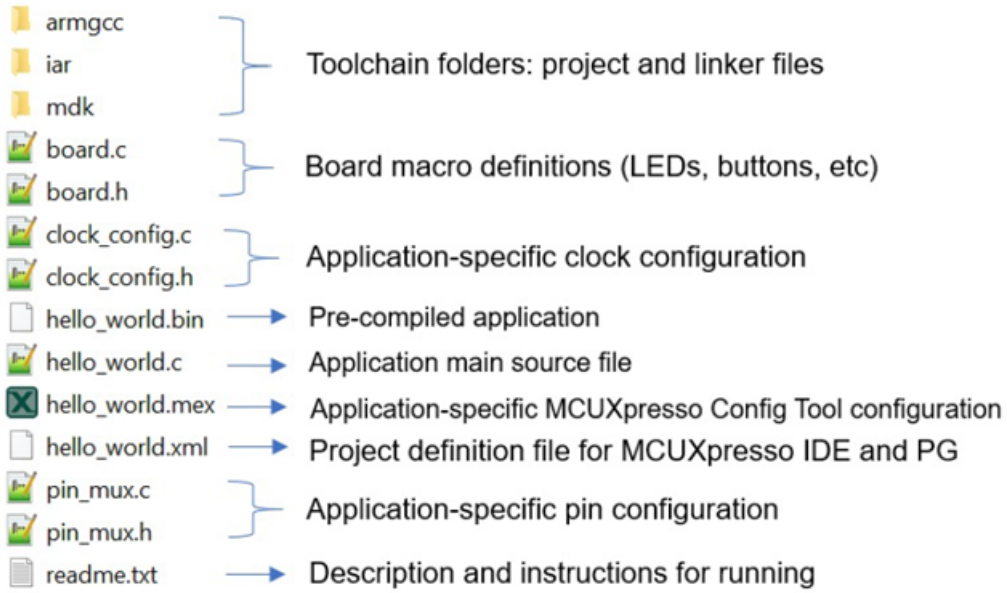
- demo_apps: Applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications intended to concisely illustrate how to use the MCUXpresso SDKs peripheral drivers for a single use case. These applications typically only use a single peripheral, but there are cases where multiple are used.
- rtos_examples: Basic FreeRTOS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK RTOS drivers
- cmsis_driver_examples: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- multicore_examples: Simple applications intended to concisely illustrate how to use middleware/multicore stack.
- mmcau_examples: Simple applications intended to concisely illustrate how to use middleware/mmcau 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 exam-

ple (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, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

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

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

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

Note: The RPMsg-Lite library is located in the *<install_dir>/middleware/multicore/rpmsg-lite* folder. For detailed information about the RPMsg-Lite, to see the RPMsg-Lite User's Guide, open the index.html located in the *<install_dir>/middleware/multicore/rpmsg_lite/doc* folder.

Parent topic:*MCUXpresso SDK Board Support Folders*

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 MCIMX7ULP-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 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
   ```

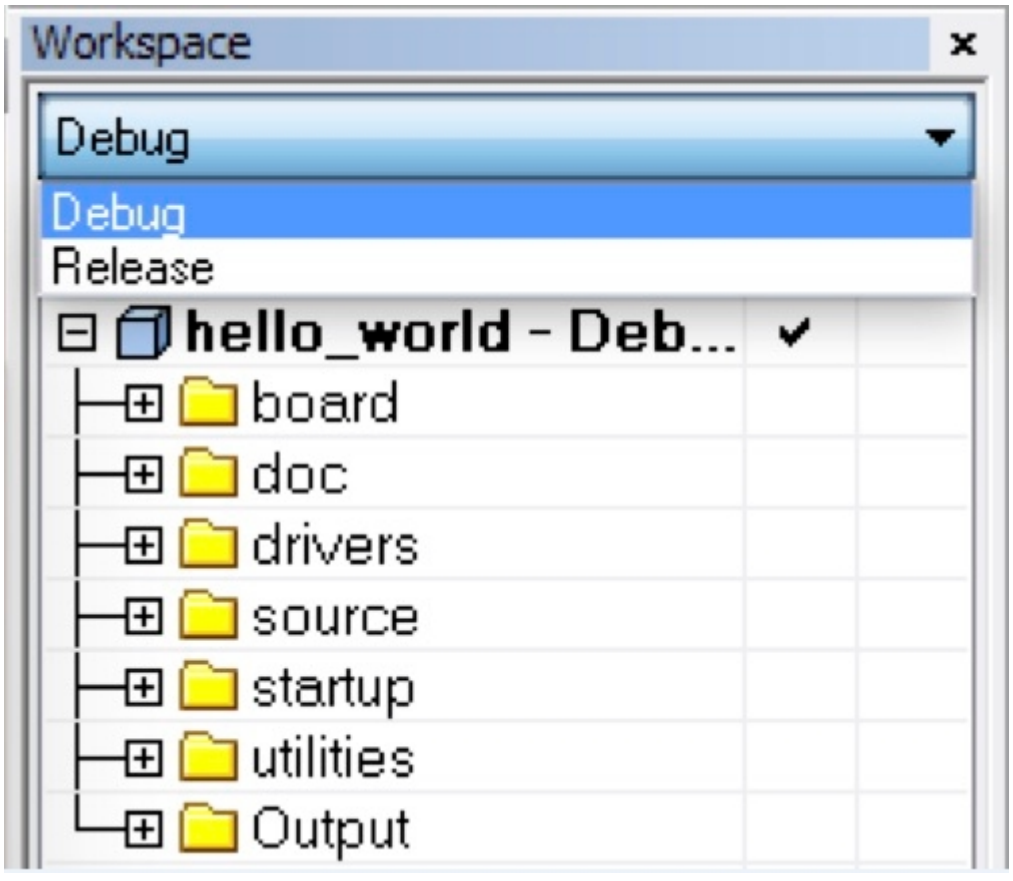
   Using the MCIMX7ULP-EVK hardware platform as an example, the hello_world workspace is located in;

   ```
   <install_dir>/boards/evkmcimx7ulp/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.

|

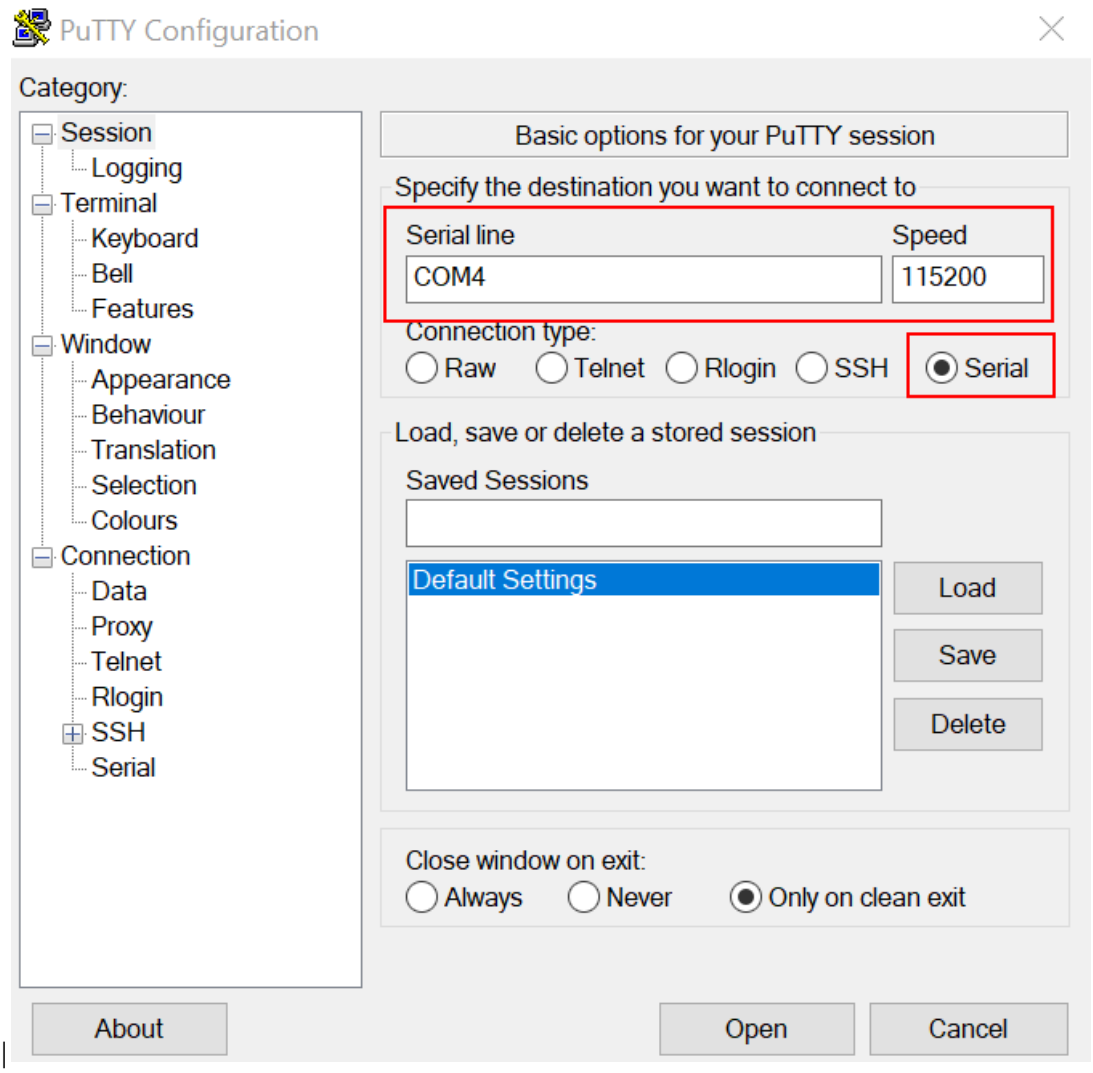
|

4. The build completes without errors.

Parent topic:*Run a demo application using IAR*

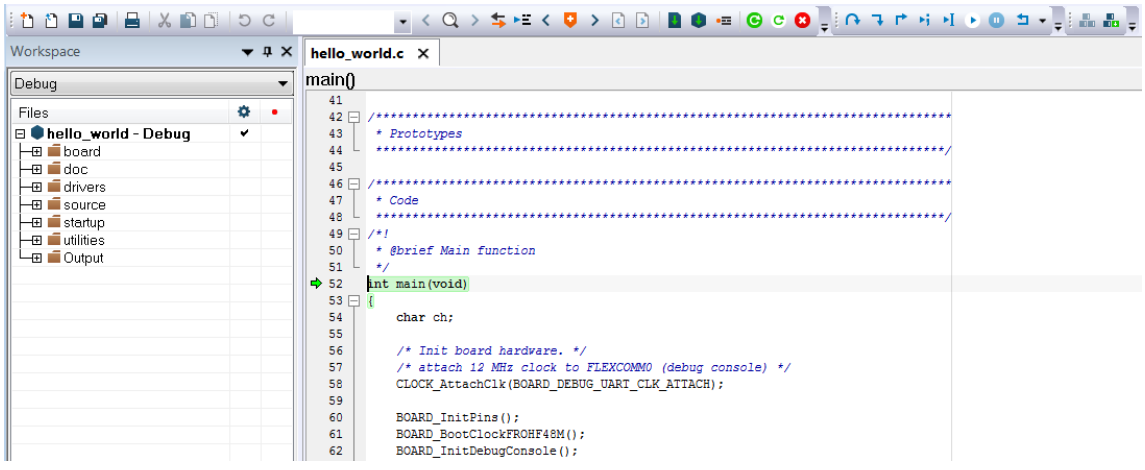
Run an example application To download and run the application, perform these steps:

1. This board supports the J-Link debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from www.segger.com.
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 5 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 Appendix A). Configure the terminal with these settings:
    1. 115200 baud rate
    2. No parity
    3. 8 data bits
    4. 1 stop bit



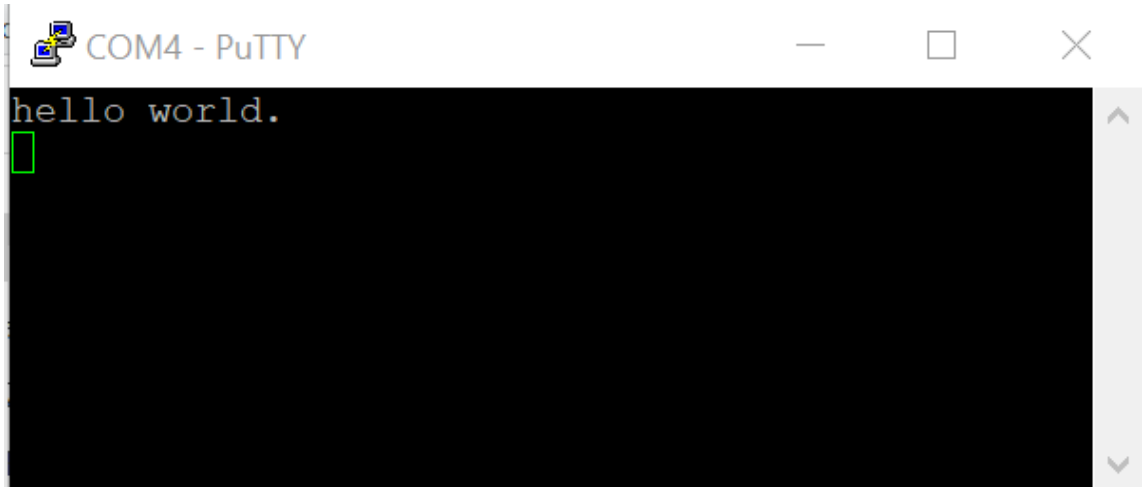
4. In IAR, click the “Download and Debug” button to download the application to the target.

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



6. Run the code by clicking the “Go” button to start the application.

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



Parent topic:*Run a demo application using IAR*

Debug QSPI FLASH XIP Application Most demo applications use the RAM linker file by default. If users want to use the flash linker file for QSPI XIP debugging, the linker file for QSPI FLASH must be changed from the project default RAM linker file to the FLASH linker file.

1. Open the hello_world project and select the hello_world top-level project as shown below. Once highlighted, one way to open the options is using ‘Alt-F7’. This opens the option window. Then, select “Linker”. The “Config” tab is shown by default. Enable Override default and enter the location: *C:|nxp|SDK_2.3_EVK_MCIMX7ULP|devices|MCIMX7U5|iar|MCIMX7U5xxx08_flash.icf*.

   |

```
Clean the project once the linker control has been configured. Use the key combo 'Alt-P' to clean. Then,␣
↪make project using the 'F7' key.
```

2. Select the “Use macro files” and the “Use flashloader “ as shown in the following pictures, and start debugging in IAR.

   |

|

```
|![](../images/use__flash__loaders__selection__imx7ulp.png ””Use flashloader
                                                                                ”
                                                                                selection”)
```

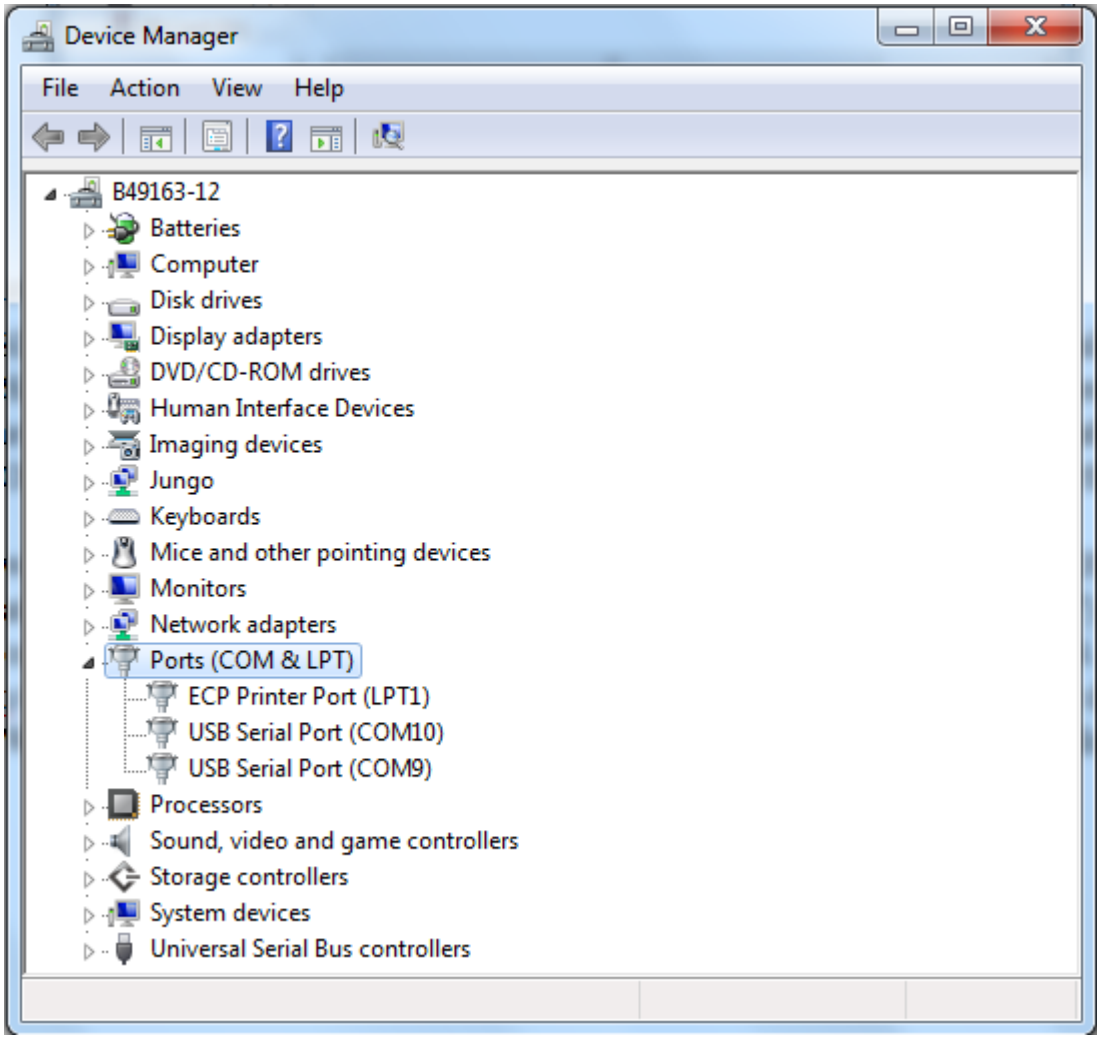
|

Parent topic:*Run a demo application using IAR*

Running an application from QSPI flash

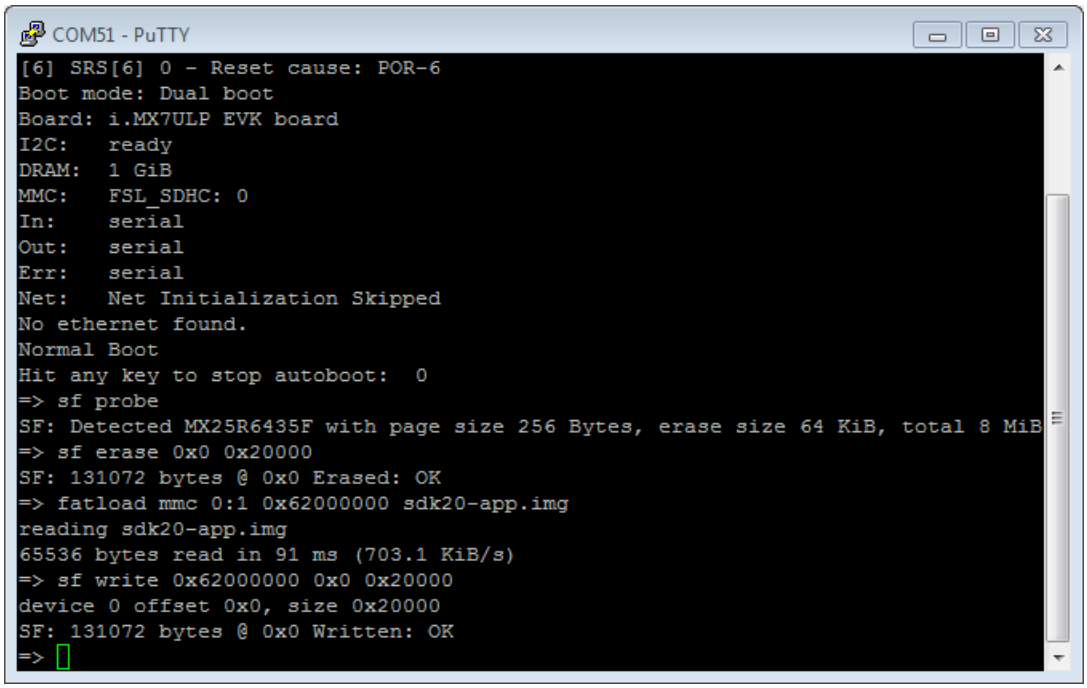
This section describes the steps to write a bootable SDK image to QSPI flash with the prebuilt U-Boot image for the i.MX processor. The following steps describe how to use the U-Boot:

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-A7 and the other is for the Cortex-M4. 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.

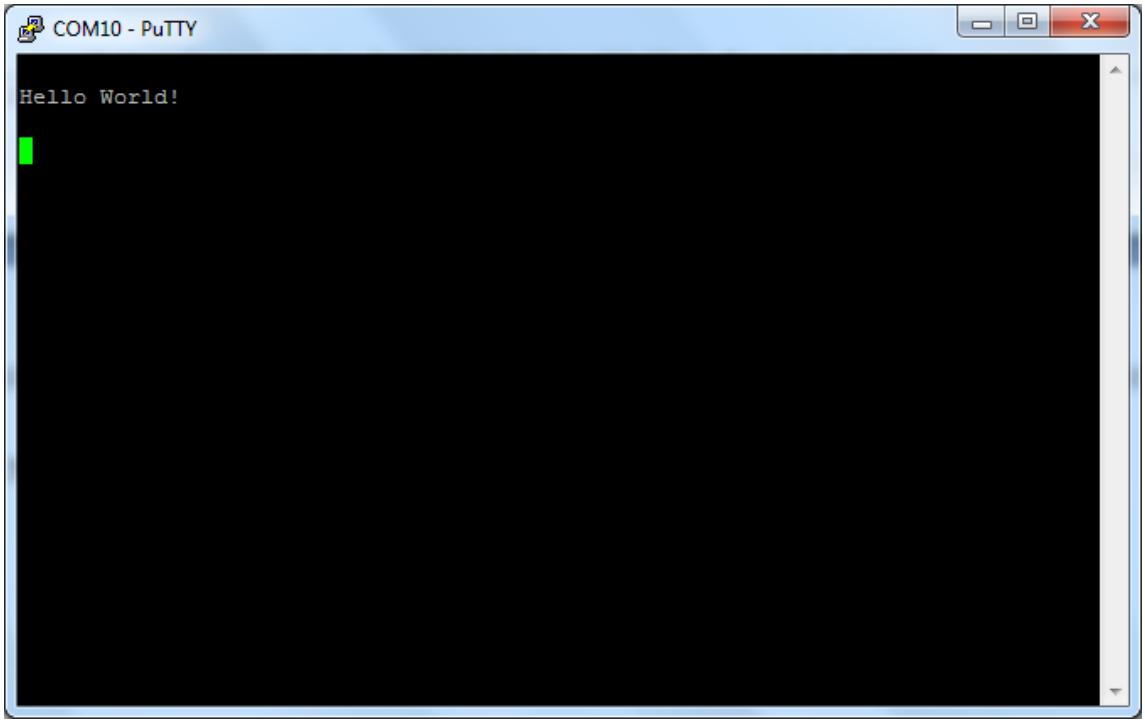


3. Build the application (for example, hello_world) and copy the built binary (sdk20-app.bin file) to the <*install_dir*>*/tools/imgutil/evkmcimx7ulp* folder.
4. In the <*install_dir*>*/tools/imgutil/evkmcimx7ulp*p folder, run mkimg.sh in mingw32 shell to get bootable image file sdk20- app.img.
   - If the image is built with RAM link file, use “mkimg.sh ram” to create the bootable image.
   - If the image is built with flash link file, use “mkimg.sh flash” to create the bootable XIP.
5. Prepare an SD card with the prebuilt U-Boot image and copy the sdk20-app.img generated into the SD card (as shown in Step 4). Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the DIP switch configuration.
6. Open your preferred serial terminals for the serial devices, setting the speed to 115200 bit/s, 8 data bits, 1 stop bit (115200, 8N1), no parity, then power on the board.
7. 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 QSPI Flash with the following commands (Assume that image size is less than 0x20000, otherwise the sf erase and write command size parameter must be enlarged. If the image size is bigger than 0x20000 (128 kB), change 0x20000 to a number larger or equal to the image size.):
   - sf probe.
   - sf erase 0x0 0x20000.

- fatload mmc 0:1 0x62000000 sdk20-app.img.
- sf write 0x62000000 0x0 0x20000.



8. Open another terminal application on the PC, such as PuTTY and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit
9. Power off and repower on the board.
10. 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.



How to determine COM port

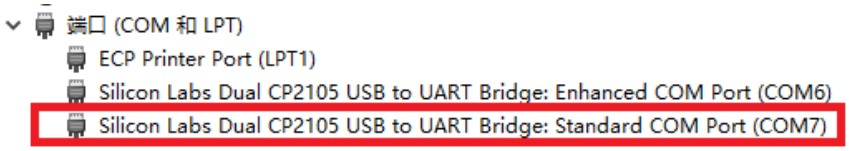
This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

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

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

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

2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.
3. In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names will be different for all the NXP boards.
   1. **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: MCUXpresso Installer
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
Code Warrior	DSC platforms	NXP CodeWarrior	CW_DIR
Xtensa	DSP core	Tensilica Tools	XCC_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
device	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 documentation for detailed installation and setup instructions.

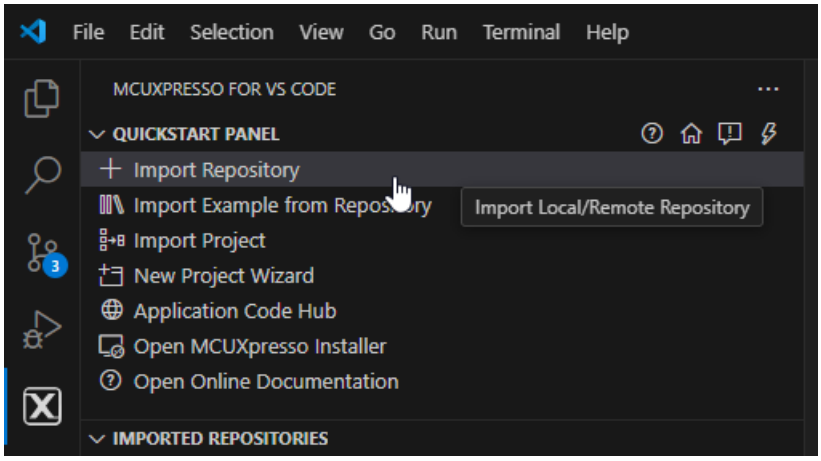
SDK Import and Setup

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.

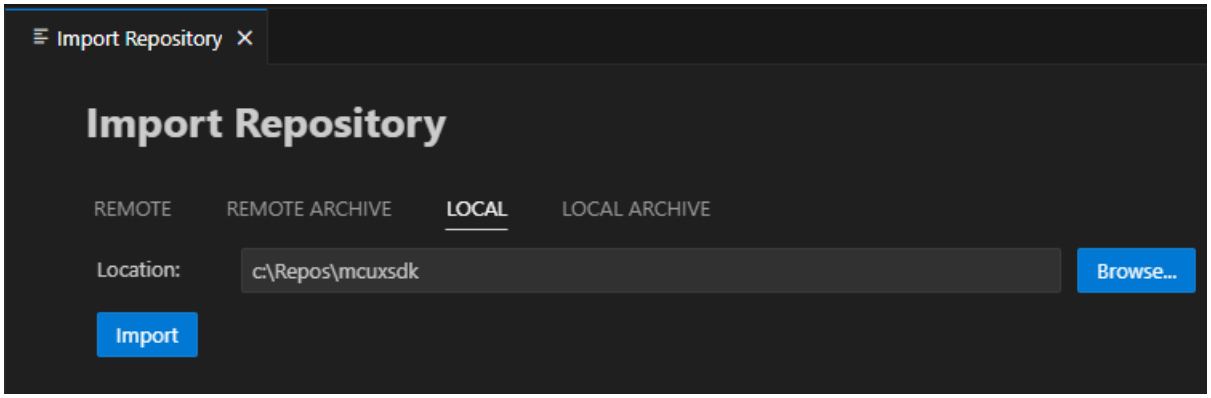
Import GitHub Repository SDK

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

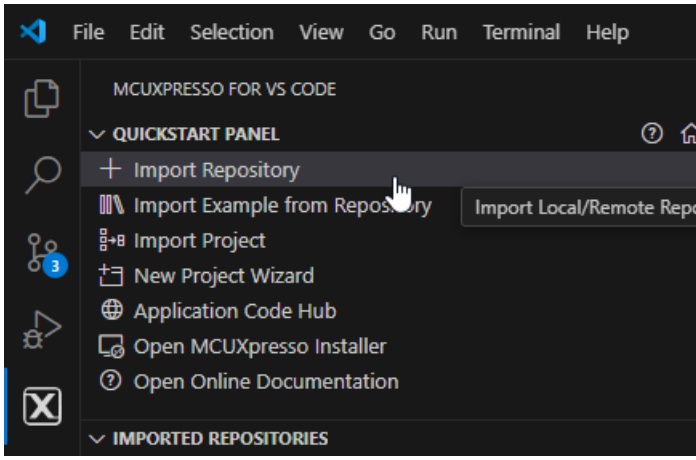


Note: You can import the SDK in several ways. Refer to the MCUXpresso for VS Code documentation for details.

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

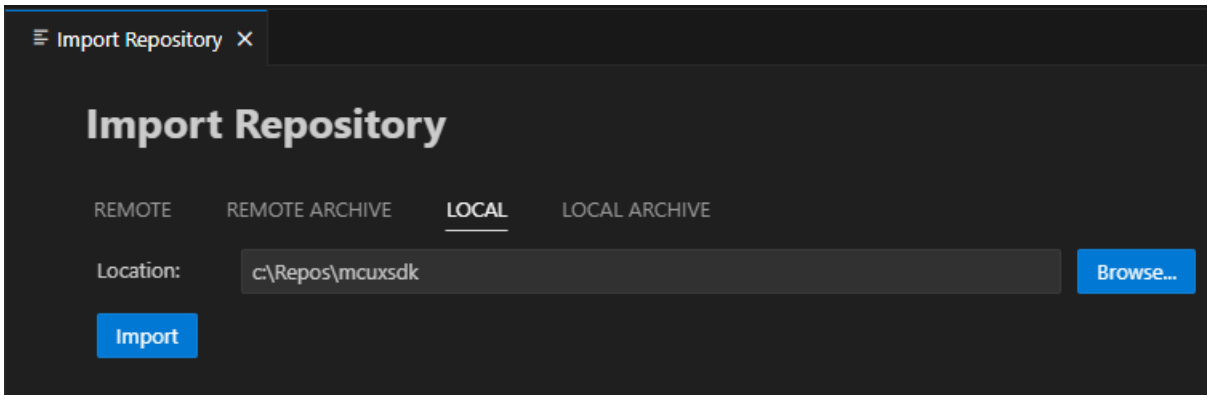


Import Repository-Layout SDK Package

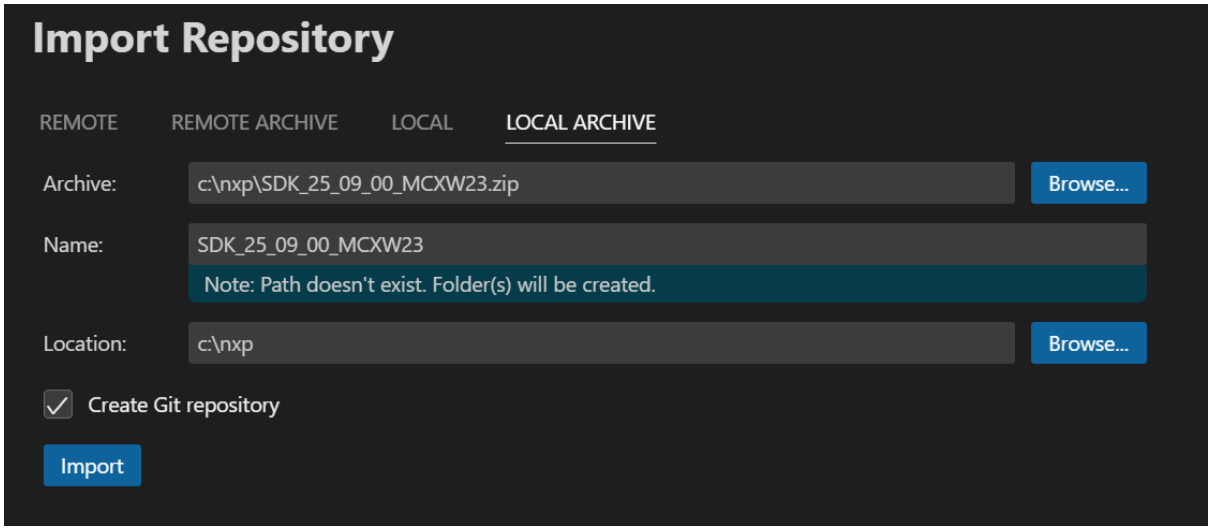


Click **Import Repository** from the **QUICKSTART PANEL**

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



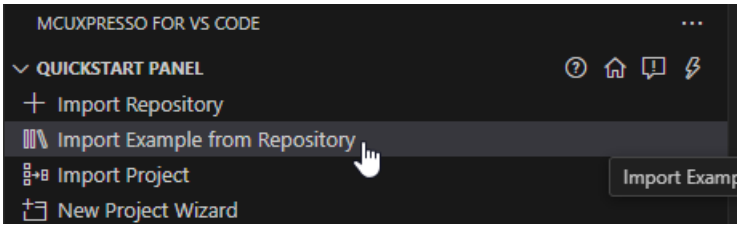
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



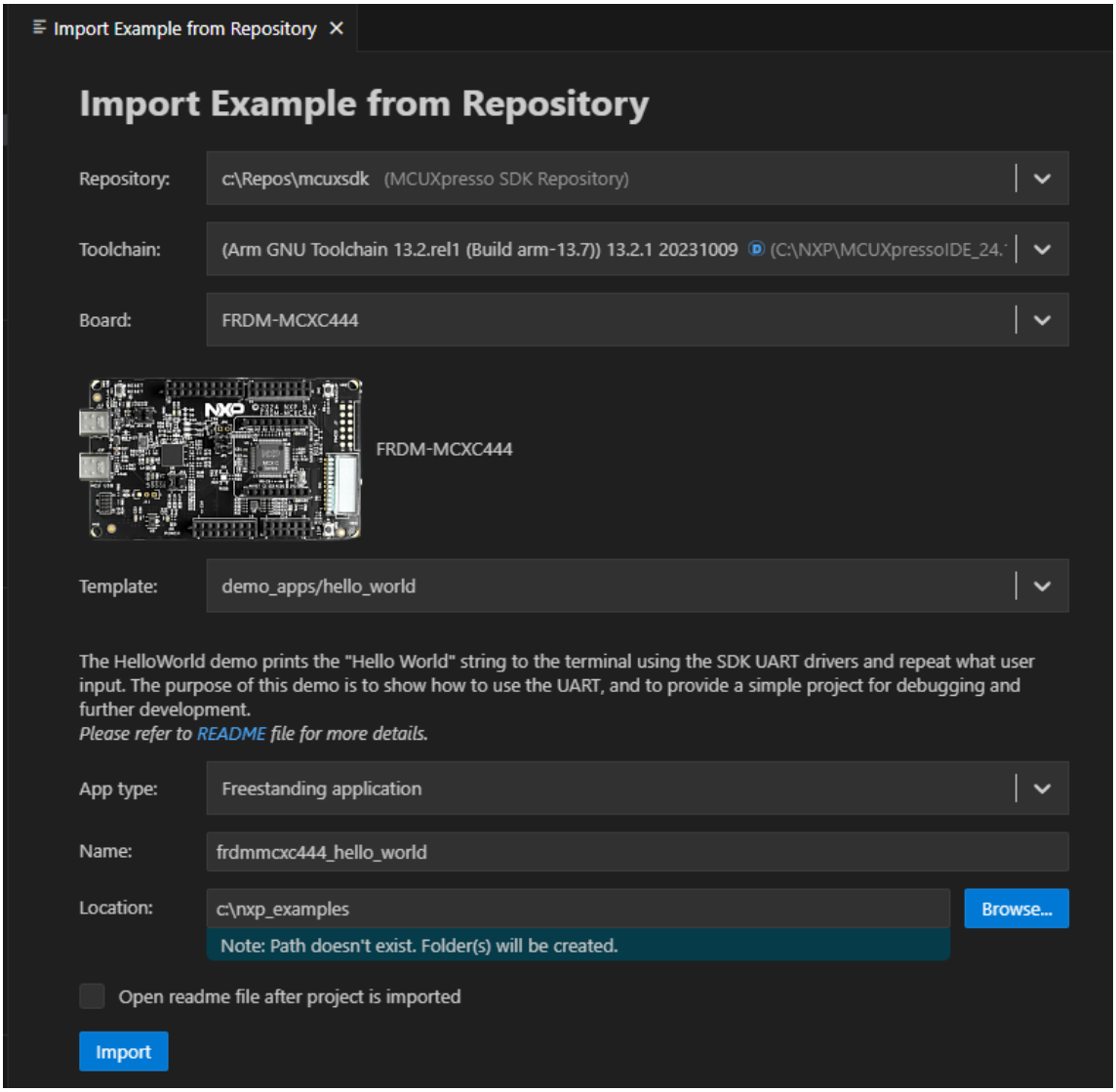
Building Example Applications

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

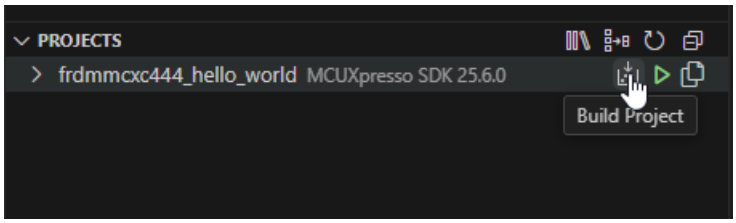
VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

Building Projects

Build Process

1. Navigate to **PROJECTS** view
2. Find your project

3. Click the **Build Project** icon

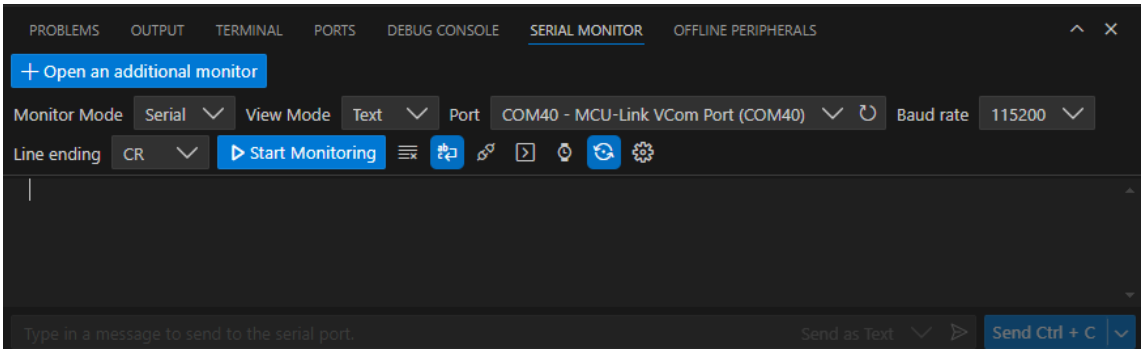


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

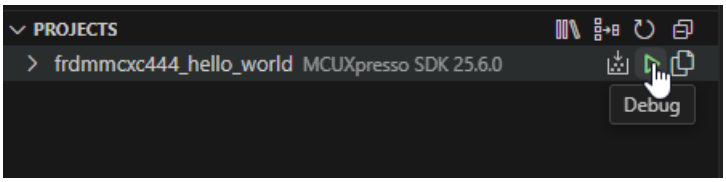
1. Open **Serial Monitor** from VS Code's integrated terminal



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



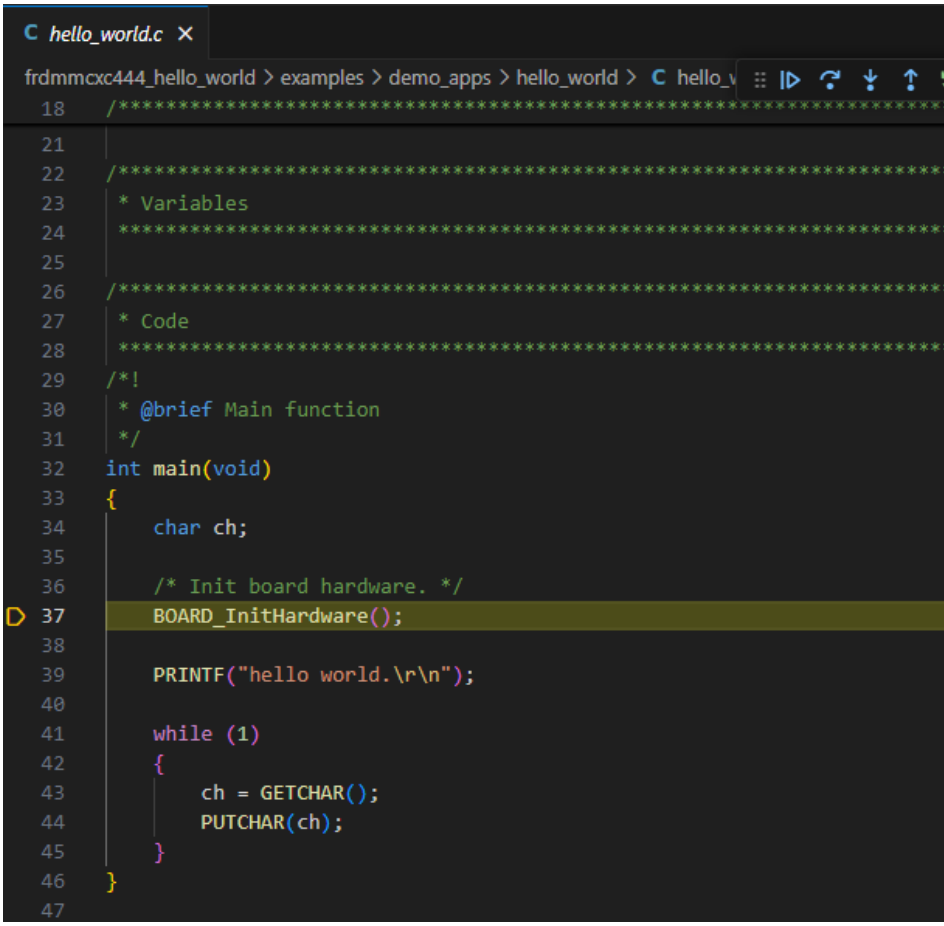
The debug session will begin with debug controls initially at the top of the interface.

Debug Controls

Use the debug controls to manage execution:

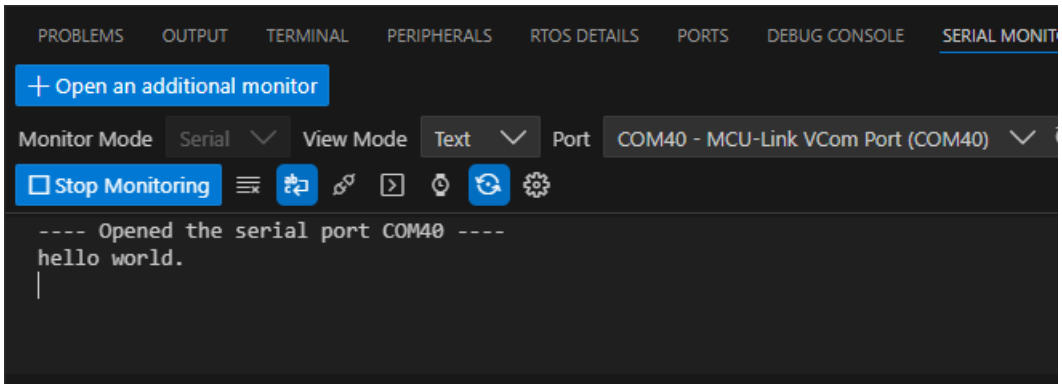
- **Continue**: Resume code execution
- **Step controls**: Navigate through code

- **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 Debug documentation.

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 to the MCUXpresso for VS Code documentation 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 9.70.4
- MCUXpresso for VS Code v26.06
- 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-MCIMX7ULP	MCIMX7U3CVP06, MCIMX7U3DVK07, MCIMX7U5CVP06, MCIMX7U5DVK07, **MCIMX7U5DVP07**

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.

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

Peripheral devices tested with USB Host stack

Table 1 provides a list of USB devices tested with the USB Host stack.

Device type	Device
USB HUB	BELKIN F5U233BELKIN F5U304BELKIN F5U307BELKIN F4U040UNITEK Y-2151Z-TEK ZK032AHYUNDAI HY-HB608
USB flash drive	ADATA C008 32 GBADATA S102 8 GADATA S102 16 GVerbatim STORE N GO USB Device 8 GKingston DataTraveler DT101 G2SanDisk Cruzer Blade 8 GBUnisplendour 1 GImation 2 GBV-mux 2 GBSanmina-SCI 128 MCorporate Express 1 GTOSHIBA THUHYBS-008G 8 GTranscend JF700 8 GNetac U903 16 GSSK SFD205 8 GBRex 4 GBSAMSUNG USB3.0 16GB
USB card reader/ USB Keyboard	SSK TF adapterKawau Multi Card ReaderKawau TF adapterKawau SDHC cardUSB Mouse DELL MS111-PDELL M066U0ADELL MUAVDEL8TARGUS AMU76APDELL MD56U0DELL MS111-TRAPOO M110 DELL SK8135DELL SK8115

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

MMCAU The NXP Memory-Mapped Cryptographic Acceleration Unit

mbedTLS mbedtls SSL/TLS library v2.x

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.

LPSPI b2b examples transfer fail on iar/armgcc flash target

Due to the latency of instruction retrieval (XiP), LPSPI-related flash target examples may fail. The failure happens because the data is not retrieved in time from FIFO on the receiving end.

To prevent the failure and boost the instruction fetch performance, place the fsl_lpspi.c file in the SRAM.

Apply the patch as below,

```
$ git diff  MCIMX7U5xxxxx_cm4_flash.ld
diff --git a/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_flash.ld b/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_
↪flash.ld
index b29b41a2b1..91dc0782e5 100644
--- a/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_flash.ld
+++ b/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_flash.ld
@@ -68,10 +68,22 @@ SECTIONS
   .text :
   {
     . = ALIGN(4);
-    *(.text)                    /* .text sections (code) */
-    *(.text*)                   /* .text* sections (code) */
-    *(.rodata)                  /* .rodata sections (constants, strings, etc.) */
-    *(.rodata*)                 /* .rodata* sections (constants, strings, etc.) */
+    *(EXCLUDE_FILE(
+        /* Exclude flash and frequently executed functions from XIP */
+        */fsl_lpspi.c.obj
+    ) .text)                    /* .text sections (code) */
+    *(EXCLUDE_FILE(
+        /* Exclude flash and frequently executed functions from XIP */
+        */fsl_lpspi.c.obj
+    ) .text*)                   /* .text* sections (code) */
+    *(EXCLUDE_FILE(
+        /* Exclude flash and frequently executed functions from XIP */
+        */fsl_lpspi.c.obj
+    ) .rodata)                  /* .rodata sections (constants, strings, etc.) */
+    *(EXCLUDE_FILE(
+        /* Exclude flash and frequently executed functions from XIP */
+        */fsl_lpspi.c.obj
+    ) .rodata*)                 /* .rodata* sections (constants, strings, etc.) */
     *(.glue_7)                  /* glue arm to thumb code */
     *(.glue_7t)                 /* glue thumb to arm code */
     *(.eh_frame)
@@ -173,6 +185,7 @@ SECTIONS
     __quickaccess_start__ = .;
     . = ALIGN(32);
     *(CodeQuickAccess)
+    /* Explicit placement of flash and frequently executed functions in SRAM  */
+    */fsl_lpspi.c.obj(.text .text* .rodata .rodata*)
     *(DataQuickAccess)
     . = ALIGN(128);
     __quickaccess_end__ = .;
```

LPSPI b2b examples transfer fail on iar/armgcc flash target.

Due to the latency of instruction retrieval (XiP), LPSPI-related flash target examples may fail. The failure happens because the data is not retrieved in time from FIFO on the receiving end.

.. code-block:: none

$ git diff MCIMX7U5xxxxx_cm4_flash.ld diff –git a/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_flash.ld b/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_flash.ld index b29b41a2b1..91dc0782e5 100644 — a/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_flash.ld +++ b/MCIMX7U5/gcc/MCIMX7U5xxxxx_cm4_flash.ld @@ -68,10 +68,22 @@ SECTIONS .text : { . = ALIGN(4); - *(.text)* / .text sections (code) / - (.text)* / .text* sections (code) / - (.rodata) / *.rodata sections (constants, strings, etc.)* / - *(.rodata)* / *.rodata* sections (constants, strings, etc.) */ + *(EXCLUDE_FILE(+ / Exclude flash and frequently executed functions from XIP */ + */fsl_lpspi.c.obj +) .text)* / .text sections (code) / + (EXCLUDE_FILE(+ / *Exclude flash and frequently executed functions from XIP* / + */fsl_lpspi.c.obj +) .text*) / *.text* sections (code) */ + *(EXCLUDE_FILE(+ / Exclude flash and frequently executed functions from XIP */ + */fsl_lpspi.c.obj +) .rodata)* / .rodata sections (constants, strings, etc.) / + (EXCLUDE_FILE(+ / *Exclude flash and frequently executed functions from XIP* / + */fsl_lpspi.c.obj +) .rodata*) / *.rodata* sections (constants, strings, etc.) */ *(.glue_7)* / glue arm to thumb code */ *(.glue_7t)* / glue thumb to arm code */ *(.eh_frame) @@ -173,6 +185,7 @@ SECTIONS **quickaccess_start** = .; . = ALIGN(32); *(CodeQuickAccess)* + / Explicit placement of flash and frequently executed functions in SRAM / + /fsl_lpspi.c.obj(.text .text *.rodata .rodata*) *(DataQuickAccess) . = ALIGN(128); **quickaccess_end** = .;

Workaround: To prevent the failure and boost the instruction fetch performance, place the fsl_lpspi.c file in the SRAM. Apply the patch as below,

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

ACMP

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.0]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

CACHE LMEM

[2.1.0]

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

[2.1.0]

- Improvements
  - Added new feature macro to support some device do not support PCCCR[ENWRBUF] bit field.

[2.0.6]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.5]

- Improvements
  - Updated the cache enable function, don’t enable again when it is already enabled.

[2.0.4]

- Bug Fixes
  - Updated full name for lmem driver.
  - Fixed doxygen issue.

[2.0.3]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.4 and 14.4.

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed the over-4KB-size maintenance issue in invalidate/clean/clean&invalidate by range APIs.

[2.0.0]

- Initial version.

COMMON

[2.6.5]

- Bug Fixes
  - Fixed the issue that the function SDK_DelayAtLeastUs may underflow and cause a long delay when the input parameter is small and the system clock is low.

[2.6.4]

- New Features
  - Add macros to tag deprecated features and experimental features.

[2.6.3]

- New Features
  - Added bit mask inversion macros to avoid type promotion.
  - Added register operation macros.
- Improvements
  - Make function $\mathrm{MSDK_EnableCpuCycleCounter}$ compatible with CMSIS-5 and CMSIS-6.
- 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.

CRC

[2.1.0]

- Bug fix:
  - Choosing CRC clocks from CRC clock array according to instance instead of hardcoded value.

[2.0.5]

- Bug fix:
  - Fix CERT-C issue with boolean-to-unsigned integer conversion.

[2.0.4]

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

[2.0.3]

- Bug fix:
  - Fix MISRA issues.

[2.0.2]

- Bug fix:
  - Fix MISRA issues.

[2.0.1]

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

[2.0.0]

- Initial version.

DAC12

[2.1.2]

- Bug Fixes
  - Fixed CERT INT31-C issue.

[2.1.1]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.

DMAMUX

[2.1.4]

- Bug Fixes
  - Fixed CERT INT31-C violation in DMAMUX_SetSource by adding non-negative assertion and explicit cast.

[2.1.3]

- Improvements
  - Wrap DMAMUX_GetInstance into FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL to avoid build issues.

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - Modify the type of parameter source from uint32_t to int32_t in the DMAMUX_SetSource.

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EDMA

[2.4.9]

- Bug Fixes
  - Fixed CERT INT31-C and INT30-C violations in edma driver.

[2.4.8]

- New Features
  - Added common IRQ handler entry EDMA_DriverIRQHandler.

[2.4.7]

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

[2.4.6]

- Bug Fixes
  - Fixed the EDMA header index retrieval error caused by done bit calculation mistake issue.

[2.4.5]

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

[2.4.4]

- Bug Fixes
  - Fixed comments by replacing STCD with TCD
  - Fixed the TCD overwrite issue when submit transfer request in the callback if there is a active TCD in hardware.
  - Fixed violations of MISRA C-2012 rule 10.8,5.6.

[2.4.3]

- Improvements
  - Added FSL_FEATURE_MEMORY_HAS_ADDRESS_OFFSET to convert the address between system mapped address and dma quick access address.
- Bug Fixes
  - Fixed the wrong tcd done count calculated in first TCD interrupt for the non scatter gather case.

[2.4.2]

- Bug Fixes
  - Fixed the wrong tcd done count calculated in first TCD interrupt by correct the initial value of the header.
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4.

[2.4.1]

- Bug Fixes
  - Added clear CITER and BITER registers in EDMA_AbortTransfer to make sure the TCD registers in a correct state for next calling of EDMA_SubmitTransfer.
  - Removed the clear DONE status for ESG not enabled case to aovid DONE bit cleared unexpectedly.

[2.4.0]

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

[2.3.2]

- Improvements
  - Fixed HIS ccm issue in function EDMA_PrepareTransferConfig.
  - Fixed violations of MISRA C-2012 rule 11.6, 10.7, 10.3, 18.1.
- Bug Fixes
  - Added ACTIVE & BITER & CITER bitfields to determine the channel status to fixed the issue of the transfer request cannot submit by function EDMA_SubmitTransfer when channel is idle.

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EWM

[2.0.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed the hard fault in EWM_Deinit.

[2.0.0]

- Initial version.

FLEXIO

[2.3.2]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.3.1]

- New Features
  - Added common IRQ handler entry FLEXIO_CommonDriverIRQHandler.

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
  - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

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

[2.1.0]

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

[2.0.4]

- Bug Fixes
  - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.2]

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

[2.0.1]

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

FLEXIO_I2C

[2.6.3]

- Bug Fixes
  - Fixed MISRA 10.4, 10.7 and 14.4 issues.

[2.6.2]

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

[2.6.1]

- Bug Fixes
  - Fixed coverity issues

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

- Bug Fixes
  - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
  - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
  - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
  - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
  - Fixed the Dozen mode configuration error in FLEXIO_I2C_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
  - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking API called FLEXIO_I2C_MasterTransferCreateHandle, which lead to the s_flexioHandle/s_flexioIsr/s_flexioType variable being written. Then, if calling FLEXIO_I2C_MasterTransferBlocking API multiple times, the s_flexioHandle/s_flexioIsr/s_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S_EDMA

[2.1.9]

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

[2.1.8]

- Improvements
  - Applied EDMA ERRATA 51327.

FLEXIO_SPI

[2.4.4]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.4.3]

- Improvements
  - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed coverity issues

[2.4.0]

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

[2.3.5]

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

[2.3.4]

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

[2.3.3]

- Bugfixes
  - Fixed cs-continuous mode.

[2.3.2]

- Improvements
  - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

  - Fixed the error register address return for 16-bit data write in FLEXIO_SPI_GetTxDataRegisterAddress.
  - Provided independent IRQHandler/transfer APIs for Master and slave to fix the baudrate limit issue.

FLEXIO_UART

[2.6.5]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.6.4]

- Improvements
  - Make UART_RETRY_TIMES configurable by CONFIG_UART_RETRY_TIMES.

[2.6.3]

- Bug Fixes
  - Fixed coverity issues

[2.6.2]

- Bug Fixes
  - Fixed coverity issues

[2.6.1]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

- Improvements
  - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_UART_EDMA

[2.3.1]

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

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0

GPIO

[2.8.6]

- Bug Fixes
  - Fixed CERT INT31-C violations in GPIO_FIT_REG macro for platforms with 8-bit register width.

[2.8.5]

- Improvements
  - Add AND logic in GPIO_GpioClearInterruptFlags to support clearing multiple interrupt flags simultaneously.

[2.8.4]

- Improvements
  - Make function GPIO_PortGetInterruptFlags and GPIO_PortClearInterruptFlags available for all variants, they are used for all pins flags in one GPIO port.

[2.8.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 5.7.

[2.8.2]

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

[2.8.1]

- Bug Fixes
  - Fixed CERT INT31-C issues.

[2.8.0]

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

[2.8.0]

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

[2.7.3]

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

[2.7.2]

- New Features
  - Support devices without PORT module.

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
  - Removed deprecated APIs.

[2.3.0]

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

[2.2.1]

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

[2.1.1]

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

[2.1.0]

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

LLWU

[2.0.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3.
  - Fixed the issue that function LLWU_SetExternalWakeupPinMode() does not work on 32-bit width platforms.

[2.0.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 10.6, 10.7, 11.3.
  - Fixed issue that LLWU_ClearExternalWakeupPinFlag may clear other filter flags by mistake on platforms with 32-bit LLWU registers.

[2.0.3]

- Bug Fixes
  - Fixed MISRA-2012 rules.
    * Rule 16.4.

[2.0.2]

- Improvements
  - Corrected driver function LLWU_SetResetPinMode parameter name.
- Bug Fixes
  - Fixed MISRA-2012 rules.
    * Rule 14.4, 10.8, 10.4, 10.3.

[2.0.1]

- Other Changes
  - Updates for KL8x.

[2.0.0]

- Initial version.

LPADC

[2.10.0]

- Improvements
  - Use $\#\text{if } !\text{defined}(\text{ADC_TCTRL_COUNT}) \,||\, (\text{ADC_TCTRL_COUNT} > \text{nU})$ to wrap some configurations because these configurations are invalid/illegal on some SoCs.

[2.9.5]

- Improvements
  - Fix doxygen issue, grouping command should be balanced.

[2.9.4]

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

[2.9.3]

- Improvements
  - Add timeout for while loop code.

[2.9.2]

- Improvements
  - Fixed CERT-C issues.

[2.9.1]

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

[2.9.0]

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

[2.8.4]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

- Improvements
  - Added the API LPADC_SetOffset12BitValue() to configure 12bit ADC conversion offset trim value manually.
  - Added the API LPADC_SetOffset16BitValue() to configure 16bit ADC conversion offset trim value manually.
  - Added API to set offset calibration mode.
  - Added configuration of alternate channel.
  - Updated auto calibration API and added calibration value conversion API.
- New feature
  - Added API LPADC_EnableHardwareTriggerCommandSelection() to enable trigger commands controlled by ADC_ETC.
  - Updated LPADC_DoAutoCalibration() to allow doing something else before the ADC initialization to be totally complete. Enhance initialization duration time of the ADC.
  - Added two new APIs to get/set calibration value.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Improvements
  - Added missing items to enable trigger interrupts.

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

- New Feature
  - Added API LPADC_MeasureTemperature() to get correct temperature from the internal sensor.
- Improvements
  - Separated lpadc_conversion_resolution_mode_t with related feature macro.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.3, 10.4, 10.6, 10.7 and 17.7.

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPI2C

[2.6.7]

- Bug Fixes
  - Fixed MISRA 10.3 and 10.8 issues.
  - Fixed LPI2C_MasterStop() to send I2C STOP in case of error.

[2.6.6]

- Bug Fixes
  - Set handle->chunkSize to 0 in LPI2C_MasterTransferNonBlocking().

[2.6.5]

- Bug Fixes
  - Removed write to STAR register in LPI2C_SlaveTransferHandleIRQ handler in case of kLPI2C_SlaveRepeatedStartDetectFlag or kLPI2C_SlaveStopDetectFlag.
  - Removed configuration of FILTSDA from LPI2C_MasterSetBaudRate().
  - Set minimal possible value for FILTSDA and FILTSCL to 0.
  - Updated configuration of glitch filters in LPI2C_SlaveInit().
  - Set minimal possible value for SCFGR2[CLKHOLD] to 1.
  - Added error check to while loop in LPI2C_TransferStateMachineSendCommand().
- Improvements
  - Removed receive data size limitation in LPI2C_MasterTransferBlocking() and LPI2C_MasterTransferNonBlocking().

[2.6.4]

- Bug Fixes
  - Limited value of filtSda in LPI2C_MasterSetBaudRate().
  - Updated LPI2C_MasterStop() and LPI2C_MasterTransferBlocking() to send I2C STOP also in case of error.
  - Removed unused flag kLPI2C_TransferRepeatedStartFlag.

[2.6.3]

- Bug Fixes
  - Fixed static analysis identified issues.

[2.6.2]

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

[2.6.1]

- Bug Fixes
  - Fixed coverity issues.

[2.6.0]

- New Feature
  - Added common IRQ handler entry LPI2C_DriverIRQHandler.

[2.5.7]

- Improvements
  - Added support for separated IRQ handlers.

[2.5.6]

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

[2.5.5]

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

[2.5.4]

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

[2.5.3]

- Improvements
  - Added handler for LPI2C7 and LPI2C8.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
  - Updated LPI2C_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes
  - Fixed bug in LPI2C_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
  - Fixed bug in LPI2C_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.
  - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
  - Fixed bug in the usage of LPI2C_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.
  - Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
  - Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

[2.3.0]

- New Features
  - Supported reading more than 256 bytes of data in one transfer as master.
  - Added API LPI2C_GetInstance.
- Bug Fixes
  - Fixed bug in LPI2C_MasterTransferAbortEDMA, LPI2C_MasterTransferAbort and LPI2C_MasterTransferHandleIRQ that before sending stop signal whether master is active and whether stop signal has been sent should be checked, to make sure no FIFO error or bus error will be caused.

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

[2.2.0]

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

[2.1.12]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.1.11]

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

[2.1.10]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rule 10.3, 14.4, 15.5.
  - Fixed unaligned access issue in LPI2C_RunTransferStateMachine.
  - Fixed uninitialized variable issue in LPI2C_MasterTransferHandleIRQ.
  - Used linked TCD to disable tx and enable rx in read operation to fix the issue that for platform sharing the same DMA request with tx and rx, during LPI2C read operation if interrupt with higher priority happened exactly after command was sent and before tx disabled, potentially both tx and rx could trigger dma and cause trouble.
  - Fixed MISRA issues.

        * Fixed rules 10.1, 10.3, 10.4, 11.6, 11.9, 14.4, 17.7.
    - Fixed the waitTimes variable not re-assignment issue for each byte read.
- New Features
    - Added the IRQHandler for LPI2C5 and LPI2C6 instances.
- Improvements
    - Updated the LPI2C_WAIT_TIMEOUT macro to unified name I2C_RETRY_TIMES.

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - API name changed:
    - LPI2C_MasterTransferCreateHandle -> LPI2C_MasterCreateHandle.

  * LPI2C_MasterTransferGetCount -> LPI2C_MasterGetTransferCount.
  * LPI2C_MasterTransferAbort -> LPI2C_MasterAbortTransfer.
  * LPI2C_MasterTransferHandleIRQ -> LPI2C_MasterHandleInterrupt.
  * LPI2C_SlaveTransferCreateHandle -> LPI2C_SlaveCreateHandle.
  * LPI2C_SlaveTransferGetCount -> LPI2C_SlaveGetTransferCount.
  * LPI2C_SlaveTransferAbort -> LPI2C_SlaveAbortTransfer.
  * LPI2C_SlaveTransferHandleIRQ -> LPI2C_SlaveHandleInterrupt.

[2.0.0]

- Initial version.

LPI2C_EDMA

[2.4.8]

- Bug Fixes
  - Fixed MISRA 10.6 and 14.3 issues.

[2.4.7]

- Bug Fixes
  - Fixed incorrect TX FIFO size (maxTxFifo) in LPI2C_MasterTransferEDMA() and LPI2C_MasterTransferEdmaHandleIRQ().

[2.4.6]

- Bug Fixes
  - Fixed static analysis identified issues.

[2.4.5]

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

[2.4.4]

- Improvements
  - Added support for 2KB data transfer

[2.4.3]

- Improvements
  - Added support for separated IRQ handlers.

[2.4.2]

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

[2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1

LPIT

[2.1.3]

- Bug Fixes
  - Fixed doxygen generation warnings.

[2.1.2]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPSPI

[2.7.6]

- Bug Fixes
  - Fixed calculation of SCK-to-SCK delay.

[2.7.5]

- Bug Fixes
  - Fixed MISRA 10.3, 12,2 and 10.7 issues.

[2.7.4]

- Bug Fixes
  - Clear WIDTH bits from the TCR register before writing a new value in LPSPI_MasterTransferBlocking().

[2.7.3]

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

[2.7.2]

- Bug Fixes
  - Fixed coverity issues.

[2.7.1]

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

[2.7.0]

- New Feature
  - Added common IRQ handler entry LPSPI_DriverIRQHandler.

[2.6.10]

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

[2.6.9]

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

[2.6.8]

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

[2.6.7]

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

[2.6.6]

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

[2.6.5]

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

[2.6.4]

- Bug Fixes
  - Added LPSPI6_DriverIRQHandler for LPSPI6 instance.

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPSPI_EDMA

[2.4.10]

- Bug Fixes
  - Fixed MISRA 10.6 issues.

[2.4.9]

- Improvements
  - Removed unused code from LPSPI_SeparateEdmaReadData().

[2.4.8]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

- Improvements
  - Add the TXMSK wait after TCR setting.

[2.4.0]

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

LPTMR

[2.2.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPUART

[2.11.0]

- New Feature
  - Added support to enable transmitter RTS and configure transmitter RTS polarity.
  - Added support to enable Transmit Data Inversion.
- Improvements
  - Updated to work on DSC platform.

[2.10.0]

- New Feature
  - Added support to configure RTS watermark.

[2.9.4]

- Improvements
  - Merged duplicate code.

[2.9.3]

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

[2.9.2]

- Bug Fixes
  - Fixed coverity issues.

[2.9.1]

- Bug Fixes
  - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.7]

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

[2.7.6]

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

[2.7.5]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.1]

- Bug Fixes
  - Fixed MISRA advisory 17.7 issues.

[2.4.0]

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

[2.3.1]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.3.0]

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

[2.2.8]

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

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - Update transactional APIs.

LPUART_EDMA

[2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0

LTC

[2.0.18]

- Bug fix:
- Fix CERT INT30-C and INT31-C compliance.

[2.0.17]

- Bug fix:
  - Fix CMAC for payloads over one block, and if BRIC is present on the device, remove XCBC and “decrypt key” functionality

[2.0.16]

- Bug fix:
  - Fix unitialized GCC warning in LTC_AES_GenerateDecryptKey().

[2.0.15]

- Bug fix:
  - Fix MISRA-2012 issues.

[2.0.14]

- Improvements:
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Deinit function.

[2.0.13]

- Improvements:
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_init function.

[2.0.12]

- Bug fix:
  - Fix AES Decrypt in CBC modes fail when used kLTC_DecryptKey.

[2.0.11]

- Bug fix:
  - Fix MISRA-2012 issues.

[2.0.10]

- Bug fix:
  - Fix MISRA-2012 issues.

[2.0.9]

- Bug fix:
  - Fix sign-compare warning in ltc_set_context and in ltc_get_context.

[2.0.8]

- Bug fix:
  - Fixed coverity issues introduced in 2.0.7

[2.0.7]

- Bug fix:
  - Fixed MISRA-2012 issues
    - Fixed rule 15.7. 10.1. 8.6. 10.7. 10.3, 11.6, 11.3, 10.8, 16.1.

[2.0.6]

- Bug fix:
  - Fixed [KPSDK-23603][LTC] AES Decrypt in ECB and CBC modes fail when ciphertext size > 0xff0 bytes.

[2.0.5]

- Improvements:
  - Fixed MISRA issues
    - Fixed rule 15.7, rule 14.4, rule 10.4, rule 18.4, rule 17.7, 5.7, 12.2.

[2.0.4]

- Improvements:
  - Constant LTC_PKHA_CompareBigNum() processing time.

[2.0.3]

- Bug fix:
  - Fixed LTC_PKHA_CompareBigNum() in case an integer argument is an array of all zeros.

[2.0.2]

- Bug fix:
  - Fixed [KPSDK-10932][LTC][SHA] LTC_HASH() blocks indefinitely when message size exceeds 4080 Bytes.

[2.0.1]

- Bug fix:
  - Fixed c++ build warning in block_to_ififo() functions.

[2.0.0]

- Initial version.

MCM

[2.2.1]

- Improvements
  - Add feature to support platforms that do not have the MCM CPCR (Core Platform Control Register).

[2.2.0]

- Improvements
  - Support platforms with less features.

[2.1.0]

- Others
  - Remove byteID from mcm_lmem_fault_attribute_t for document update.

[2.0.0]

- Initial version.

MSMC

[2.1.2]

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

[2.1.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 3.1, 10.1, 10.4, 10.8, 14.4.

[2.1.0]

- Added new APIs with FEATURE macros support: SMC_GetStopEntryStatus() SMC_ClearStopEntryStatus() SMC_SetForceBootOptionConfig() SMC_SRAMEnableLowPowerMode() SMC_SRAMEnableDeepSleepMode()
- Updated APIs with FEATURE macros support: SMC_SetPowerModeStop() SMC_SetPowerModeVlpr() SMC_SetPowerModeLls() SMC_SetPowerModeVlls() SMC_ConfigureResetPinFilter()

[2.0.0]

- Initial version.

MU

[2.3.3]

- Improvements
  - Fix build failure when driver clock control disabled.

[2.3.2]

- Improvements
  - Fix MISRA C-2012 Rule 20.8 violations.

[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.

PMC0

[2.1.1]

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

[2.1.0]

- New features
  - Added temperature sensor control APIs.

[2.0.0]

- Initial version.

PORT

[2.5.1]

- Bug Fixes
  - Fix CERT INT31-C issues.
  - Fixed the violations of MISRA C-2012 rules: 10.1.

[2.5.0]

- Bug Fixes
  - Correct the kPORT_MuxAsGpio for some platforms.

[2.4.1]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

- New Features
  - Added new api PORT_EnablePinDoubleDriveStrength.

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

QSPI

[2.3.2]

- Improvements
  - Fixed Coverity MSG issues.

[2.3.1]

- Improvements
  - Fixed Coverity MSG issues.

[2.3.0]

- New Features
  - Applied the QSPI IP update with register field changes.
  - Added Soc specific driver to integrate Soc configuration.
- Changed
  - Updated the QSPI LUT update function to be compatible with different sequence unit.
  - Added new feature macro FSL_FEATURE_QSPI_HAS_SOC_SPECIFIC_CONFIG which represents there’re Soc specific QSPI configurations. Soc specific driver should cover these configurations. Previous Soc specific code in the common driver should be masked.

[2.2.5]

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

[2.2.4]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3.

[2.2.3]

- Bug Fixes
  - Cleared buffer generic configuration when do software reset.

[2.2.2]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1 and 11.9.

[2.2.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.6, 10.8, 11.3, 11.6, 11.8, 11.9, 14.4, 16.1, 16.4, 17.7.

[2.2.0]

- New Features
  - Added new API QSPI_ClearCache to clear cache for new IP feature FSL_FEATURE_QSPI_SOCCR_HAS_CLR_LPCAC.
- Bug Fixes
  - Fixed the QSPI_WriteBlocking API programming issue for low watermark, caused by previous improvement change of using TX watermark signal to fill the TX FIFO. Reverted change to previous implementation to use TX FIFO full flag for filling the FIFO. Improved previous API by accessing TX data register directly.
  - Fixed the issue that QSPI_SetIPCommandSize incorrectly triggered a transaction.
  - Fixed clock divider accurate issue when using internal QSPI internal divider.
  - Fixed build fail issue for some devices' not supporting API QSPI_SetDqsConfig for DQS configuration.

[2.1.0]

- New Features
  - Added new API QSPI_SetDqsConfig for DQS configuration.
- Improvements
  - Updated the QSPI_WriteBlocking API to fill the TX FIFO once there are bytes of TX watermark room in the FIFO. This will improve the performance of filling TX FIFO when watermark is high.

[2.0.2]

- Improvements
  - New Macro function:
    * Added QSPI_LUT_SEQ() function for users to set LUT table easily.
    * Added LUT command macros for users to easy use.
  - Comment update:
    * Added the comments for the limitation of QSPI_ReadBlocking and QSPI_TransferReceiveBlocking.

[2.0.1]

- Improvements
  - New API:
    * QSPI_SetReadArea to set the read area.
- Bug Fixes
  - Fixed the issue that QSPI_UpdateLUT function only updated first LUT.
  - Fixed issue that some function that hardcode QSPI0 as base.

[2.0.0]

- Initial version.

QSPI_EDMA

[2.2.5]

- Improvements
  - Fixed Coverity MSG issues.

[2.2.4]

- Changed
  - Compatible with new EDMA driver.

[2.2.3]

- Changed
  - Used instance array number to count the QSPI instead of feature macro.

[2.0.0]

- Initial version.

SAI

[2.4.11]

- Bug Fixes
  - Fixed MSG findings for platforms with multiple FIFO size support.

[2.4.10]

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

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

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

[2.4.7]

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

[2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

[2.4.5]

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

[2.4.4]

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

[2.4.3]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
  - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

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

[2.3.6]

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

[2.3.5]

- Improvements
  - Make driver to be aarch64 compatible.

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
  - Added new APIs for parameters collection and simplified user interfaces:
    - SAI_Init
    - SAI_SetMasterClockConfig
    - SAI_TxSetBitClockRate
    - SAI_TxSetSerialDataConfig
    - SAI_TxSetFrameSyncConfig
    - SAI_TxSetFifoConfig
    - SAI_TxSetBitclockConfig
    - SAI_TxSetConfig
    - SAI_TxSetTransferConfig
    - SAI_RxSetBitClockRate
    - SAI_RxSetSerialDataConfig
    - SAI_RxSetFrameSyncConfig
    - SAI_RxSetFifoConfig
    - SAI_RxSetBitclockConfig
    - SAI_RXSetConfig
    - SAI_RxSetTransferConfig
    - SAI_GetClassicI2SConfig
    - SAI_GetLeftJustifiedConfig
    - SAI_GetRightJustifiedConfig
    - SAI_GetTDMConfig

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - API name changes:
    - SAI_GetSendRemainingBytes -> SAI_GetSentCount.
    - SAI_GetReceiveRemainingBytes -> SAI_GetReceivedCount.
    - All names of transactional APIs were added with “Transfer” prefix.
    - All transactional APIs use base and handle as input parameter.
    - Unified the parameter names.
- Bug Fixes
  - Fixed WLC bug while reading TCSR/RCSR registers.
  - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI_TxInit/SAI_RxInit.

[2.0.0]

- Initial version.

SAI_EDMA

[2.7.4]

- Bug Fixes
  - Fixed MSG finding for platforms with multiple FIFO size support.

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
  - Removed deprecated APIs.

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2

SEMA42

[2.1.2]

- Improvements
  - Fix MISRA C-2012 Rule 20.8 violations.

[2.1.1]

- Improvements
  - Updated SEMA42_TryLock function to avoid unsigned integer operations wrap issue.

[2.1.0]

- New Features
  - Added SEMA42_BUSY_POLL_COUNT parameter to prevent infinite polling loops in SEMA42 operations.
  - Added timeout mechanism to all polling loops in SEMA42 driver code.
- Improvements
  - Updated SEMA42_Lock function to return status_t instead of void for better error handling.
  - Enhanced documentation to clarify timeout behavior and return values.

[2.0.4]

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

[2.0.3]

- Improvements
  - Changed to implement SEMA42_Lock base on SEMA42_TryLock.

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

SNVS_HP

[2.3.2]

- Make SNVS_HP_RTC_Init()/SNVS_HP_RTC_Deinit more transparent. Use function SNVS_HP_Init()/SNVS_HP_Deinit() instead of copy of this code in SNVS_HP_RTC_XXX() function.

[2.3.1]

- Fixed problem in SNVS_HP_RTC_Init(), which is clearing bits that should stay intact.

[2.3.0]

- Re-map Security Violation for RT11xx specific violations.

[2.2.0]

- Fixed doxygen issues.
- Add SNVS HP Set locks.

[2.1.4]

- Fix MISRA issues.

[2.1.3]

- Fixed IAR Pa082 warnings.

[2.1.2]

- Fixed problem with initialization of the periodic interrupt frequency.
- Fixed problem with SNVS entering into fail state when HAB enters closed mode.

[2.1.1]

- Added APIs for HP security violation status flags.

[2.1.0]

- Added APIs for High Assurance Counter (HAC), Zeroizable Master Key (ZMK) and Software Security Violation.

[2.0.0]

- Initial version.

SNVS_LP

[2.4.6]

- Fix a bug in SNVS_LP_EnableRxActiveTamper() where assignments to base->LPATRC2R were done wrongly to LPATRC1R.

[2.4.5]

- Fix a bug in SNVS_LP_EnableRxActiveTamper() where assignments to base->LPATRC1R would overwrite previously set bits.

[2.4.4]

- Make SNVS_LP_SRTC_Init()/SNVS_LP_SRTC_Deinit more transparent. Use function SNVS_LP_Init()/SNVS_LP_Deinit() instead of copy of this code in SNVS_LP_SRTC_XXX() function.

[2.4.3]

- Fixed problem in SNVS_LP_SRTC_Init(), which is clearing bits that should stay intact.

[2.4.2]

- Updated driver to match with new device header files.

[2.4.1]

- Fixed MISRA issues.

[2.4.0]

- Fix backward compatibility with version 2.2.x.

[2.3.0]

- Add active pin, clock, voltage and temperature tamper features.

[2.2.0]

- Fixed doxygen issues.
- Add Transition SNVS SSM state to Trusted/Non-secure from Check state.

[2.1.2]

- Fix MISRA issues.

[2.1.1]

- Fix IAR Pa082 warning.

[2.1.0]

- Added APIs for Zeroizable Master Key (ZMK) and Monotonic Counter (MC).

[2.0.0]

- Initial version.

TPM

[2.5.0]

- Improvements
  - Added TPM_GetChannelControlBits API to replace the misspelled TPM_GetChannelContorlBits, which is now marked as deprecated.

[2.4.6]

- Improvements
  - Added feature macro FSL_FEATURE_TPM_HAS_NO_SC_CPWMS to support SoCs whose TPM SC register does not have the CPWMS field.

[2.4.5]

- Improvements
  - Added branch coverage justification comments and GCOVR_EXCL_BR_LINE markers for structurally unreachable branches in TPM_UpdatePwmDutycycle.

[2.4.4]

- Bug Fixes
  - Fixed CERT INT31-C: removed unsafe (uint8_t) cast on FSL_FEATURE_TPM_HAS_32BIT_COUNTERn() (which returns int with possible value -1) in TPM_MAX_COUNTER_VALUE macro and TPM_SetTimerPeriod; now uses direct int comparison to avoid truncation.

[2.4.3]

- Bug Fixes
  - Fixed CERT INT31-C Coverity issue: Casting from int to unsigned char without checking its value may result in lost or misinterpreted data.

[2.4.2]

- Bug Fixes
  - Fixed CERT INT30-C INT31-C issue for:
    - Unsigned integer operation may wrap.
    - Casting from “int” to “unsigned char” may result in lost or misinterpreted data.

[2.4.1]

- Improvements
  - Add Coverage Justification for uncovered code.

[2.4.0]

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

[2.3.6]

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

[2.3.5]

- New Feature
  - Added IRQ handler entry for TPM2.

[2.3.4]

- New Feature
  - Added common IRQ handler entry TPM_DriverIRQHandler.

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.8]

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

[2.0.7]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

TRGMUX

[2.0.1]

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

[2.0.0]

- Initial version.

TRNG

[2.0.22]

- New features:
  - Added support for KW43 and MCXW70 devices.

[2.0.21]

- New features:
  - Added support for MCXC devices.

[2.0.20]

- New features:
  - Added support for MCXA devices.

[2.0.19]

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

[2.0.18]

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

[2.0.17]

- New features:
  - Add support for RT700.

[2.0.16]

- Improvements:
  - Added support for Dual oscillator mode.

[2.0.15]

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

[2.0.14]

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

[2.0.13]

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

[2.0.12]

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

[2.0.11]

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

[2.0.10]

- Bug Fix:
  - Fixed doxygen issues.

[2.0.9]

- Bug Fix:
  - Fix HIS_CCM metrics issues.

[2.0.8]

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

[2.0.7]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

- Improvements:
  - fix MISRA issues
    * Rule 14.4.

[2.0.1]

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

[2.0.0]

- Initial version.

TSTMR

[2.1.0]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

WDOG32

[2.2.1]

- Bug Fixes
  - Fix CERT INT31-C that the bool value shall be converted to unsigned int 0 or 1 then passed to registers.
  - Fix MISRA 2012 20.3 vilation.

[2.2.0]

- Improvements
  - Added while loop timeout config value for WDOG32 reconfiguration and unlock sequence.
  - Change the return type of WDOG32_Init, WDOG32_Deinit and WDOG32_Unlock from void to status_t.

[2.1.0]

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

[2.0.4]

- Improvements
  - To ensure that the reconfiguration is inside 128 bus clocks unlock window, put all reconfiguration APIs in quick access code section.

[2.0.3]

- Bug Fixes
  - Fixed the noncompliance issue of the reference document.
    - Waited until for new configuration to take effect by checking the RCS bit field.
    - Waited until for registers to be unlocked by checking the ULK bit field.
- Improvements
  - Added 128 bus clocks delay ensures a smooth transition before restarting the counter with the new configuration when there is no RCS status bit.

[2.0.2]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    - WDOG32_Refresh

[2.0.1]

- Bug Fixes
  - WDOG must be configured within its configuration time period.
    - Added WDOG32_Init API to quick access section.
    - Defined register variable in WDOG32_Init API.

[2.0.0]

- Initial version.

XRDC

[2.0.7]

- Improvements
  - Handle errata ERR050593.

[2.0.6]

- Improvements
  - Supported platforms which don't have XRDC clock gate control.

[2.0.5]

- Bug Fixes
  - Fixed XRDC_GetAndClearFirstSpecificDomainError potential array over index issue.

[2.0.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 3.1, 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 11.3, 12.2, 14.4, 17.7, 20.7.

[2.0.3]

- Improvements
  - Added necessary driver supports for K32H844P.
  - Added new APIs concerning new features of Exclusive Access Lock and programmable domain access flags configurations.

[2.0.2]

- Bug Fixes
  - Fixed wrong assert of assignIndex input check in the xRDC driver.
- Improvements
  - Added master input CPU/non-CPU check in XRDC_SetNonProcessorDomainAssignment and XRDC_SetProcessorDomainAssignment API.
  - Added necessary assert checks for several config inputs.

[2.0.1]

- Improvements
  - Changed reserved bit fields in the structs into unnamed-identifier bit fields.

[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.

MCIMX7U5

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

MCIMX7U5

2.1 ACMP: Analog Comparator Driver

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

Initializes the ACMP.

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

Parameters

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

void ACMP_Deinit(CMP_Type *base)

Deinitializes the ACMP.

Parameters

- base – ACMP peripheral base address.

void ACMP_GetDefaultConfig(*acmp_config_t* *config)

Gets the default configuration for ACMP.

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

Example:

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

Parameters

- config – Pointer to ACMP configuration structure.

void ACMP_Enable(CMP_Type *base, bool enable)

Enables or disables the ACMP.

Parameters

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

void ACMP_EnableLinkToDAC(CMP_Type *base, bool enable)

Enables the link from CMP to DAC enable.

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

Parameters

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

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

Sets the channel configuration.

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

Example:

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

Parameters

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

void ACMP_EnableDMA(CMP_Type *base, bool enable)

Enables or disables DMA.

Parameters

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

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

Configures the filter.

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

Example:

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

Parameters

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

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

Configures the internal DAC.

Example:

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

Parameters

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

void ACMP_EnableInterrupts(CMP_Type *base, uint32_t mask)

Enables interrupts.

Parameters

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

void ACMP_DisableInterrupts(CMP_Type *base, uint32_t mask)

Disables interrupts.

Parameters

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

uint32_t ACMP_GetStatusFlags(CMP_Type *base)

Gets status flags.

Parameters

- base – ACMP peripheral base address.

Returns

Status flags asserted mask. See "_acmp_status_flags".

void ACMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)

Clears status flags.

Parameters

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

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

Configure the discrete mode.

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

Parameters

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

void ACMP_GetDefaultDiscreteModeConfig(*acmp_discrete_mode_config_t* *config)

Get the default configuration for discrete mode setting.

Parameters

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

FSL_ACMP_DRIVER_VERSION

ACMP driver version 2.5.0.

enum _acmp_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kACMP_OutputRisingInterruptEnable

Enable the interrupt when comparator outputs rising.

enumerator kACMP_OutputFallingInterruptEnable

Enable the interrupt when comparator outputs falling.

enum _acmp_status_flags

Status flag mask.

Values:

enumerator kACMP_OutputRisingEventFlag

Rising-edge on compare output has occurred.

enumerator kACMP_OutputFallingEventFlag

Falling-edge on compare output has occurred.

enumerator kACMP_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum _acmp_offset_mode

Comparator hard block offset control.

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

Values:

enumerator kACMP_OffsetLevel0

The comparator hard block output has level 0 offset internally.

enumerator kACMP_OffsetLevel1

The comparator hard block output has level 1 offset internally.

enum _acmp_hysteresis_mode

Comparator hard block hysteresis control.

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

Values:

enumerator kACMP_HysteresisLevel0

Offset is level 0 and Hysteresis is level 0.

enumerator kACMP_HysteresisLevel1

Offset is level 0 and Hysteresis is level 1.

enumerator kACMP_HysteresisLevel2

Offset is level 0 and Hysteresis is level 2.

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

enum _acmp_reference_voltage_source
CMP Voltage Reference source.

Values:

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

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

enum _acmp_port_input
Port input source.

Values:

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

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

enum _acmp_dac_work_mode
Internal DAC's work mode.

Values:

enumerator kACMP_DACWorkLowSpeedMode
DAC is selected to work in low speed and low power mode.

enumerator kACMP_DACWorkHighSpeedMode
DAC is selected to work in high speed high power mode.

typedef enum *_acmp_offset_mode* acmp_offset_mode_t
Comparator hard block offset control.

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

typedef enum *_acmp_hysteresis_mode* acmp_hysteresis_mode_t
Comparator hard block hysteresis control.

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

typedef enum *_acmp_reference_voltage_source* acmp_reference_voltage_source_t
CMP Voltage Reference source.

typedef enum *_acmp_port_input* acmp_port_input_t
Port input source.

typedef enum *_acmp_dac_work_mode* acmp_dac_work_mode_t
Internal DAC's work mode.

typedef struct *_acmp_config* acmp_config_t
Configuration for ACMP.

typedef struct *_acmp_channel_config* acmp_channel_config_t

Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

typedef struct *_acmp_filter_config* acmp_filter_config_t

Configuration for filter.

typedef struct *_acmp_dac_config* acmp_dac_config_t

Configuration for DAC.

typedef struct *_acmp_discrete_mode_config* acmp_discrete_mode_config_t

Configuration for discrete mode.

CMP_C0_CFx_MASK

The mask of status flags cleared by writing 1.

struct _acmp_config

#include <fsl_acmp.h> Configuration for ACMP.

Public Members

acmp_offset_mode_t offsetMode

Offset mode.

acmp_hysteresis_mode_t hysteresisMode

Hysteresis mode.

bool enableHighSpeed

Enable High Speed (HS) comparison mode.

bool enableInvertOutput

Enable inverted comparator output.

bool useUnfilteredOutput

Set compare output(COUT) to equal COUTA(true) or COUT(false).

bool enablePinOut

The comparator output is available on the associated pin.

struct _acmp_channel_config

#include <fsl_acmp.h> Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

Public Members

acmp_port_input_t positivePortInput

Input source of the comparator's positive port.

uint32_t plusMuxInput

Plus mux input channel(0~7).

acmp_port_input_t negativePortInput

Input source of the comparator's negative port.

uint32_t minusMuxInput

Minus mux input channel(0~7).

struct _acmp_filter_config

#include <fsl_acmp.h> Configuration for filter.

Public Members

uint32_t filterCount

Filter Sample Count. Available range is 1-7, 0 would cause the filter disabled.

uint32_t filterPeriod

Filter Sample Period. The divider to bus clock. Available range is 0-255.

struct _acmp_dac_config

#include <fsl_acmp.h> Configuration for DAC.

Public Members

acmp_reference_voltage_source_t referenceVoltageSource

Supply voltage reference source.

uint32_t DACValue

Value for DAC Output Voltage. Available range is 0-255.

bool enableOutput

Enable the DAC output.

struct _acmp_discrete_mode_config

#include <fsl_acmp.h> Configuration for discrete mode.

Public Members

bool enablePositiveChannelDiscreteMode

Positive Channel Continuous Mode Enable. By default, the continuous mode is used.

bool enableNegativeChannelDiscreteMode

Negative Channel Continuous Mode Enable. By default, the continuous mode is used.

2.2 CACHE: LMEM CACHE Memory Controller

void L1CACHE_EnableCodeCache(void)

Enables the processor code bus cache.

void L1CACHE_DisableCodeCache(void)

Disables the processor code bus cache.

void L1CACHE_InvalidateCodeCache(void)

Invalidates the processor code bus cache.

void L1CACHE_InvalidateCodeCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates processor code bus cache by range.

Note: Address and size should be aligned to “L1CODCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to L1CODEBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

void L1CACHE_CleanCodeCache(void)

Cleans the processor code bus cache.

void L1CACHE_CleanCodeCacheByRange(uint32_t address, uint32_t size_byte)

Cleans processor code bus cache by range.

Note: Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

void L1CACHE_CleanInvalidateCodeCache(void)

Cleans and invalidates the processor code bus cache.

void L1CACHE_CleanInvalidateCodeCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate processor code bus cache by range.

Note: Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be Cleaned and Invalidated.

static inline void L1CACHE_EnableCodeCacheWriteBuffer(bool enable)

Enables/disables the processor code bus write buffer.

Parameters

- enable – The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.

void L1CACHE_EnableSystemCache(void)

Enables the processor system bus cache.

void L1CACHE_DisableSystemCache(void)

Disables the processor system bus cache.

void L1CACHE_InvalidateSystemCache(void)

Invalidates the processor system bus cache.

void L1CACHE_InvalidateSystemCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates processor system bus cache by range.

Note: Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1SYSTEMBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

void L1CACHE_CleanSystemCache(void)

Cleans the processor system bus cache.

void L1CACHE_CleanSystemCacheByRange(uint32_t address, uint32_t size_byte)

Cleans processor system bus cache by range.

Note: Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1SYSTEMBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

void L1CACHE_CleanInvalidateSystemCache(void)

Cleans and invalidates the processor system bus cache.

void L1CACHE_CleanInvalidateSystemCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates processor system bus cache by range.

Note: Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1SYSTEMBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be Clean and Invalidated.

static inline void L1CACHE_EnableSystemCacheWriteBuffer(bool enable)

Enables/disables the processor system bus write buffer.

Parameters

- enable – The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.

void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cortex-m4 L1 instrument cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned.

Parameters

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

static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cortex-m4 L1 data cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

Parameters

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

void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cortex-m4 L1 data cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

Parameters

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

void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates cortex-m4 L1 data cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

Parameters

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

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

Note: Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1ICACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

Note: Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

Note: Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

Note: Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated.

FSL_CACHE_DRIVER_VERSION

cache driver version.

L1CODEBUSCACHE_LINESIZE_BYTE

code bus cache line size is equal to system bus line size, so the unified I/D cache line size equals too.

The code bus CACHE line size is 16B = 128b.

L1SYSTEMBUSCACHE_LINESIZE_BYTE

The system bus CACHE line size is 16B = 128b.

2.3 Clock

enum _clock_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_CoreSysClk
Core/system clock

enumerator kCLOCK_PlatClk
Platform clock

enumerator kCLOCK_ExtClk
External clock

enumerator kCLOCK_BusClk
Bus clock

enumerator kCLOCK_SlowClk
Slow clock

enumerator kCLOCK_ScgSysOscClk
SCG system OSC clock. (SYSOSC)

enumerator kCLOCK_ScgSircClk
SCG SIRC clock.

enumerator kCLOCK_ScgFircClk
SCG FIRC clock.

enumerator kCLOCK_ScgRtcOscClk
SCG RTC OSC clock. (ROSC)

enumerator kCLOCK_ScgAuxPllClk
SCG auxiliary PLL clock. (AUXPLL)

enumerator kCLOCK_ScgSysPllClk
SCG system PLL clock. (SYSPLL)

enumerator kCLOCK_ScgSysOscAsyncDiv1Clk
SOSCDIV1_CLK.

enumerator kCLOCK_ScgSysOscAsyncDiv2Clk
SOSCDIV2_CLK.

enumerator kCLOCK_ScgSysOscAsyncDiv3Clk
SOSCDIV3_CLK.

enumerator kCLOCK_ScgSircAsyncDiv1Clk
SIRCDIV1_CLK.

enumerator kCLOCK_ScgSircAsyncDiv2Clk
SIRCDIV2_CLK.

enumerator kCLOCK_ScgSircAsyncDiv3Clk
SIRCDIV3_CLK.

enumerator kCLOCK_ScgFircAsyncDiv1Clk
FIRCDIV1_CLK.

enumerator kCLOCK_ScgFircAsyncDiv2Clk
FIRCDIV2_CLK.

enumerator kCLOCK_ScgFircAsyncDiv3Clk
FIRCDIV3_CLK.

enumerator kCLOCK_ScgSysPllPfd0Clk
spll pfd0.

enumerator kCLOCK_ScgSysPllPfd1Clk
spll pfd1.

enumerator kCLOCK_ScgSysPllPfd2Clk
spll pfd2.

enumerator kCLOCK_ScgSysPllPfd3Clk
spll pfd3.

enumerator kCLOCK_ScgAuxPllPfd0Clk
apll pfd0.

enumerator kCLOCK_ScgAuxPllPfd1Clk
apll pfd1.

enumerator kCLOCK_ScgAuxPllPfd2Clk
apll pfd2.

enumerator kCLOCK_ScgAuxPllPfd3Clk
apll pfd3.

enumerator kCLOCK_ScgSysPllAsyncDiv1Clk
SPLLDIV1_CLK.

enumerator kCLOCK_ScgSysPllAsyncDiv2Clk
SPLLDIV2_CLK.

enumerator kCLOCK_ScgSysPllAsyncDiv3Clk
SPLLDIV3_CLK.

enumerator kCLOCK_ScgAuxPllAsyncDiv1Clk
APLLDIV1_CLK.

enumerator kCLOCK_ScgAuxPllAsyncDiv2Clk
APLLDIV2_CLK.

enumerator kCLOCK_ScgAuxPllAsyncDiv3Clk
APLLDIV3_CLK.

enumerator kCLOCK_LpoClk
LPO clock

enumerator kCLOCK_Osc32kClk
External OSC 32K clock (OSC32KCLK)

enumerator kCLOCK_ErClk
ERCLK. The external reference clock from SCG.

enumerator kCLOCK_LvdsClk
LVDS pad input clock frequency.

enum _clock_ip_src
Clock source for peripherals that support various clock selections.

Values:

enumerator kCLOCK_IpSrcNone
Clock is off.

enumerator kCLOCK_IpSrcSysOscAsync
SYSOSC platform or bus clock, depending on clock IP.

enumerator kCLOCK_IpSrcSircAsync
SIRC platform or bus clock, depending on clock IP.

enumerator kCLOCK_IpSrcFircAsync
FIRC platform or bus clock, depending on clock IP.

enumerator kCLOCK_IpSrcRtcAuxPllAsync
RTC OSC clock or AUXPLL main clock, depending on clock IP.

enumerator kCLOCK_IpSrcSystem
System platform or bus clock, depending on clock IP.

enumerator kCLOCK_IpSrcSysPllAsync
SYSPLL platform or bus clock, depending on clock IP.

enumerator kCLOCK_IpSrcPllPfdAsync
SYSPLL PFD3 or AUXPLL PFD0 clock, depending on clock IP.

enum _clock_lptmr_src
Clock source for LPTMR.

Values:

enumerator kCLOCK_LptmrSrcSircAsync
SIRC clock.

enumerator kCLOCK_LptmrSrcLPO1K
LPO 1KHz clock.

enumerator kCLOCK_LptmrSrcXTAL32K
RTC XTAL clock.

enumerator kCLOCK_LptmrSrcExternal
External clock.

enum _clock_ip_name
Peripheral clock name difinition used for clock gate, clock source and clock divider setting. It is defined as the corresponding register address.

Values:

enumerator kCLOCK_IpInvalid

enumerator kCLOCK_Dma0

enumerator kCLOCK_Rgpio2p0

enumerator kCLOCK_Xrdc0

enumerator kCLOCK_Sema420

enumerator kCLOCK_Dmamux0

enumerator kCLOCK_MuA

enumerator kCLOCK_Wdog0

enumerator kCLOCK_Crc0

enumerator kCLOCK_Ltc0

enumerator kCLOCK_Trng0

enumerator kCLOCK_Lpit0

enumerator kCLOCK_Lptmr0

enumerator kCLOCK_Lptmr1

enumerator kCLOCK_Tpm0

enumerator kCLOCK_Tpm1

enumerator kCLOCK_Flexio0

enumerator kCLOCK_Lpi2c0

enumerator kCLOCK_Lpi2c1

enumerator kCLOCK_Lpi2c2

enumerator kCLOCK_Lpi2c3

enumerator kCLOCK_Sai0

enumerator kCLOCK_Lpspi0

enumerator kCLOCK_Lpspi1

enumerator kCLOCK_Lpuart0

enumerator kCLOCK_Lpuart1

enumerator kCLOCK_PctlA

enumerator kCLOCK_PctlB

enumerator kCLOCK_Adc0

enumerator kCLOCK_Cmp0

enumerator kCLOCK_Cmp1

enumerator kCLOCK_Dac0

enumerator kCLOCK_Dac1

enumerator kCLOCK_Snvs

enumerator kCLOCK_Tpiu

enumerator kCLOCK_Qspi

enumerator kCLOCK_Tpm2

enumerator kCLOCK_Tpm3

enumerator kCLOCK_Sai1

enumerator kCLOCK_Lpuart2

enumerator kCLOCK_Lpuart3

enumerator kCLOCK_Adc1

enumerator kCLOCK_Dma1

enumerator kCLOCK_Rgpio2p1

enumerator kCLOCK_Flexbus

enumerator kCLOCK_Sema421

enumerator kCLOCK_Dmamux1

enumerator kCLOCK_Caam

enumerator kCLOCK_Tpm4

enumerator kCLOCK_Tpm5

enumerator kCLOCK_Lpit1

enumerator kCLOCK_Lpspi2

enumerator kCLOCK_Lpspi3

enumerator kCLOCK_Lpi2c4

enumerator kCLOCK_Lpi2c5

enumerator kCLOCK_Lpuart4

enumerator kCLOCK_Lpuart5

enumerator kCLOCK_Flexio1

enumerator kCLOCK_Usb0

enumerator kCLOCK_Usb1

enumerator kCLOCK_UsbPhy

enumerator kCLOCK_UsbPl301

enumerator kCLOCK_Usdhc0

enumerator kCLOCK_Usdhc1

enumerator kCLOCK_Wdog1

enumerator kCLOCK_Wdog2

enumerator kCLOCK_Tpm6

enumerator kCLOCK_Tpm7

enumerator kCLOCK_Lpi2c6

enumerator kCLOCK_Lpi2c7

enumerator kCLOCK_Lpuart6

enumerator kCLOCK_Lpuart7

enumerator kCLOCK_Viu

enumerator kCLOCK_Dsi

enumerator kCLOCK_Lcdif

enumerator kCLOCK_Mmdc

enumerator kCLOCK_PctlC

enumerator kCLOCK_PctlD

enumerator kCLOCK_PctlE

enumerator kCLOCK_PctlF

enumerator kCLOCK_Gpu3D

enumerator kCLOCK_Gpu2D

SCG status return codes.

Values:

enumerator kStatus_SCG_Busy
Clock is busy.

enumerator kStatus_SCG_InvalidSrc
Invalid source.

enum _scg_sys_clk
SCG system clock type.

Values:

enumerator kSCG_SysClkSlow
System slow clock.

enumerator kSCG_SysClkBus
Bus clock.

enumerator kSCG_SysClkExt
External clock.

enumerator kSCG_SysClkPlat
Platform clock.

enumerator kSCG_SysClkCore
Core clock.

enum _scg_sys_clk_src
SCG system clock source.

Values:

enumerator kSCG_SysClkSrcSysOsc
System OSC.

enumerator kSCG_SysClkSrcSirc
Slow IRC.

enumerator kSCG_SysClkSrcFirc
Fast IRC.

enumerator kSCG_SysClkSrcRosc
RTC OSC.

enumerator kSCG_SysClkSrcAuxPll
Auxiliary PLL.

enumerator kSCG_SysClkSrcSysPll
System PLL.

enum _scg_sys_clk_div
SCG system clock divider value.

Values:

enumerator kSCG_SysClkDivBy1
Divided by 1.

enumerator kSCG_SysClkDivBy2
Divided by 2.

enumerator kSCG_SysClkDivBy3
Divided by 3.

enumerator kSCG_SysClkDivBy4
Divided by 4.

enumerator kSCG_SysClkDivBy5
Divided by 5.

enumerator kSCG_SysClkDivBy6
Divided by 6.

enumerator kSCG_SysClkDivBy7
Divided by 7.

enumerator kSCG_SysClkDivBy8
Divided by 8.

enumerator kSCG_SysClkDivBy9
Divided by 9.

enumerator kSCG_SysClkDivBy10
Divided by 10.

enumerator kSCG_SysClkDivBy11
Divided by 11.

enumerator kSCG_SysClkDivBy12
Divided by 12.

enumerator kSCG_SysClkDivBy13
Divided by 13.

enumerator kSCG_SysClkDivBy14
Divided by 14.

enumerator kSCG_SysClkDivBy15
Divided by 15.

enumerator kSCG_SysClkDivBy16
Divided by 16.

enum _clock_clkout_src
SCG clock out configuration (CLKOUTSEL).

Values:

enumerator kClockClkoutSelScgExt
SCG external clock.

enumerator kClockClkoutSelSysOsc
System OSC.

enumerator kClockClkoutSelSirc
Slow IRC.

enumerator kClockClkoutSelFirc
Fast IRC.

enumerator kClockClkoutSelScgRtcOsc
SCG RTC OSC clock.

enumerator kClockClkoutSelScgAuxPll
SCG Auxiliary PLL clock.

enumerator kClockClkoutSelSysPll
System PLL.

enum _scg_async_clk
SCG asynchronous clock type.

Values:

enumerator kSCG_AsyncDiv1Clk
The async clock by DIV1, e.g. SOSCDIV1_CLK, SIRCDIV1_CLK.

enumerator kSCG_AsyncDiv2Clk
The async clock by DIV2, e.g. SOSCDIV2_CLK, SIRCDIV2_CLK.

enumerator kSCG_AsyncDiv3Clk
The async clock by DIV3, e.g. SOSCDIV3_CLK, SIRCDIV3_CLK.

enum scg_async_clk_div
SCG asynchronous clock divider value.

Values:

enumerator kSCG_AsyncClkDisable
Clock output is disabled.

enumerator kSCG_AsyncClkDivBy1
Divided by 1.

enumerator kSCG_AsyncClkDivBy2
Divided by 2.

enumerator kSCG_AsyncClkDivBy4
Divided by 4.

enumerator kSCG_AsyncClkDivBy8
Divided by 8.

enumerator kSCG_AsyncClkDivBy16
Divided by 16.

enumerator kSCG_AsyncClkDivBy32
Divided by 32.

enumerator kSCG_AsyncClkDivBy64
Divided by 64.

enum _scg_sosc_monitor_mode
SCG system OSC monitor mode.

Values:

enumerator kSCG_SysOscMonitorDisable
Monitor disabled.

enumerator kSCG_SysOscMonitorInt
Interrupt when the system OSC error is detected.

enumerator kSCG_SysOscMonitorReset
Reset when the system OSC error is detected.

enum _scg_sosc_mode
OSC work mode.

Values:

enumerator kSCG_SysOscModeExt
Use external clock.

enumerator kSCG_SysOscModeOscLowPower
Oscillator low power.

enumerator kSCG_SysOscModeOscHighGain
Oscillator high gain.

enum _scg_sosc_enable_mode
OSC enable mode.

Values:

enumerator kSCG_SysOscEnable
Enable OSC clock.

enumerator kSCG_SysOscEnableInStop
Enable OSC in stop mode.

enumerator kSCG_SysOscEnableInLowPower
Enable OSC in low power mode.

enum _scg_sirc_range
SCG slow IRC clock frequency range.

Values:

enumerator kSCG_SircRangeLow
Slow IRC low range clock (2 MHz, 4 MHz for i.MX 7ULP).

enumerator kSCG_SircRangeHigh
Slow IRC high range clock (8 MHz, 16 MHz for i.MX 7ULP).

enum _scg_sirc_enable_mode
SIRC enable mode.

Values:

enumerator kSCG_SircEnable
Enable SIRC clock.

enumerator kSCG_SircEnableInStop
Enable SIRC in stop mode.

enumerator kSCG_SircEnableInLowPower
Enable SIRC in low power mode.

enum _scg_firc_trim_mode

SCG fast IRC trim mode.

Values:

enumerator kSCG_FircTrimNonUpdate

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

enumerator kSCG_FircTrimUpdate

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

enum _scg_firc_trim_div

SCG fast IRC trim predivided value for system OSC.

Values:

enumerator kSCG_FircTrimDivBy1

Divided by 1.

enumerator kSCG_FircTrimDivBy128

Divided by 128.

enumerator kSCG_FircTrimDivBy256

Divided by 256.

enumerator kSCG_FircTrimDivBy512

Divided by 512.

enumerator kSCG_FircTrimDivBy1024

Divided by 1024.

enumerator kSCG_FircTrimDivBy2048

Divided by 2048.

enum _scg_firc_trim_src

SCG fast IRC trim source.

Values:

enumerator kSCG_FircTrimSrcUsb0

USB0 start of frame (1kHz).

enumerator kSCG_FircTrimSrcUsb1

USB1 start of frame (1kHz).

enumerator kSCG_FircTrimSrcSysOsc

System OSC.

enumerator kSCG_FircTrimSrcRtcOsc

RTC OSC (32.768 kHz).

enum _scg_firc_range

SCG fast IRC clock frequency range.

Values:

enumerator kSCG_FircRange48M

Fast IRC is trimmed to 48 MHz.

enumerator kSCG_FircRange52M

Fast IRC is trimmed to 52 MHz.

enumerator kSCG_FircRange56M
Fast IRC is trimmed to 56 MHz.

enumerator kSCG_FircRange60M
Fast IRC is trimmed to 60 MHz.

enum _scg_firc_enable_mode
FIRC enable mode.

Values:

enumerator kSCG_FircEnable
Enable FIRC clock.

enumerator kSCG_FircEnableInStop
Enable FIRC in stop mode.

enum _scg_spll_src
SCG system PLL clock source.

Values:

enumerator kSCG_SysPllSrcSysOsc
System PLL clock source is system OSC.

enumerator kSCG_SysPllSrcFirc
System PLL clock source is fast IRC.

enum _scg_spll_enable_mode
SPLL enable mode.

Values:

enumerator kSCG_SysPllEnable
Enable SPLL clock.

enumerator kSCG_SysPllEnableInStop
Enable SPLL in stop mode.

enum _scg_spll_pfd_clkout
SCG system PLL PFD clouk out select.

Values:

enumerator kSCG_SysPllPfd0Clk
PFD0 output clock selected.

enumerator kSCG_SysPllPfd1Clk
PFD1 output clock selected.

enumerator kSCG_SysPllPfd2Clk
PFD2 output clock selected.

enumerator kSCG_SysPllPfd3Clk
PFD3 output clock selected.

enum _scg_rosc_monitor_mode
SCG RTC OSC monitor mode.

Values:

enumerator kSCG_rtcOscMonitorDisable
Monitor disable.

enumerator kSCG_rtcOscMonitorInt
Interrupt when the RTC OSC error is detected.

enumerator kSCG_rtcOscMonitorReset
Reset when the RTC OSC error is detected.

enum _scg_apll_src
SCG auxiliary PLL clock source.

Values:

enumerator kSCG_AuxPllSrcSysOsc
Auxiliary PLL clock source is the system OSC.

enumerator kSCG_AuxPllSrcFirc
Auxiliary PLL clock source is the fast IRC.

enum _scg_apll_enable_mode
APLL enable mode.

Values:

enumerator kSCG_AuxPllEnable
Enable APLL clock.

enumerator kSCG_AuxPllEnableInStop
Enable APLL in stop mode.

enum _scg_apll_pfd_clkout
SCG auxiliary PLL PFD clouk out select.

Values:

enumerator kSCG_AuxPllPfd0Clk
PFD0 output clock selected.

enumerator kSCG_AuxPllPfd1Clk
PFD1 output clock selected.

enumerator kSCG_AuxPllPfd2Clk
PFD2 output clock selected.

enumerator kSCG_AuxPllPfd3Clk
PFD3 output clock selected.

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

typedef enum *_clock_ip_src* clock_ip_src_t
Clock source for peripherals that support various clock selections.

typedef enum *_clock_lptmr_src* clock_lptmr_src_t
Clock source for LPTMR.

typedef enum *_clock_ip_name* clock_ip_name_t
Peripheral clock name difinition used for clock gate, clock source and clock divider setting. It is defined as the corresponding register address.

typedef enum *_scg_sys_clk* scg_sys_clk_t
SCG system clock type.

typedef enum *_scg_sys_clk_src* scg_sys_clk_src_t
SCG system clock source.

typedef enum *_scg_sys_clk_div* scg_sys_clk_div_t
SCG system clock divider value.

typedef struct *_scg_sys_clk_config* scg_sys_clk_config_t
SCG system clock configuration.

typedef enum *_clock_clkout_src* clock_clkout_src_t
SCG clock out configuration (CLKOUTSEL).

typedef enum *_scg_async_clk* scg_async_clk_t
SCG asynchronous clock type.

typedef enum *scg_async_clk_div* scg_async_clk_div_t
SCG asynchronous clock divider value.

typedef enum *_scg_sosc_monitor_mode* scg_sosc_monitor_mode_t
SCG system OSC monitor mode.

typedef enum *_scg_sosc_mode* scg_sosc_mode_t
OSC work mode.

typedef struct *_scg_sosc_config* scg_sosc_config_t
SCG system OSC configuration.

typedef enum *_scg_sirc_range* scg_sirc_range_t
SCG slow IRC clock frequency range.

typedef struct *_scg_sirc_config* scg_sirc_config_t
SCG slow IRC clock configuration.

typedef enum *_scg_firc_trim_mode* scg_firc_trim_mode_t
SCG fast IRC trim mode.

typedef enum *_scg_firc_trim_div* scg_firc_trim_div_t
SCG fast IRC trim predivided value for system OSC.

typedef enum *_scg_firc_trim_src* scg_firc_trim_src_t
SCG fast IRC trim source.

typedef struct *_scg_firc_trim_config* scg_firc_trim_config_t
SCG fast IRC clock trim configuration.

typedef enum *_scg_firc_range* scg_firc_range_t
SCG fast IRC clock frequency range.

typedef struct *_scg_firc_config_t* scg_firc_config_t
SCG fast IRC clock configuration.

typedef enum *_scg_spll_src* scg_spll_src_t
SCG system PLL clock source.

typedef enum *_scg_spll_pfd_clkout* scg_spll_pfd_clkout_t
SCG system PLL PFD clouk out select.

typedef struct *_scg_spll_config* scg_spll_config_t
SCG system PLL configuration.

typedef enum *_scg_rosc_monitor_mode* scg_rosc_monitor_mode_t
SCG RTC OSC monitor mode.

typedef struct *_scg_rosc_config* scg_rosc_config_t
SCG RTC OSC configuration.

typedef enum *_scg_apll_src* scg_apll_src_t

SCG auxiliary PLL clock source.

typedef enum *_scg_apll_pfd_clkout* scg_apll_pfd_clkout_t

SCG auxiliary PLL PFD clouk out select.

typedef struct *_scg_apll_config* scg_apll_config_t

SCG auxiliary PLL configuration.

volatile uint32_t g_xtal0Freq

External XTAL0 (OSC0/SYSOSC) clock frequency.

The XTAL0/EXTAL0 (OSC0/SYSOSC) clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal0Freq to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
CLOCK_InitSysOsc(...);
CLOCK_SetXtal0Freq(80000000);
```

This is important for the multicore platforms where only one core needs to set up the OSC0/SYSOSC using CLOCK_InitSysOsc. All other cores need to call the CLOCK_SetXtal0Freq to get a valid clock frequency.

volatile uint32_t g_xtal32Freq

External XTAL32/EXTAL32 clock frequency.

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

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

volatile uint32_t g_lvdsFreq

External LVDS pad clock frequency.

The LVDS pad clock frequency in Hz. When the clock is set up, use the function CLOCK_SetLvdsFreq to set the value in the clock driver.

static inline void CLOCK_EnableClock(*clock_ip_name_t* name)

Enable the clock for specific IP.

Parameters

- name – Which clock to enable, see clock_ip_name_t.

static inline void CLOCK_DisableClock(*clock_ip_name_t* name)

Disable the clock for specific IP.

Parameters

- name – Which clock to disable, see clock_ip_name_t.

static inline bool CLOCK_IsEnabledByOtherCore(*clock_ip_name_t* name)

Check whether the clock is already enabled and configured by any other core.

Parameters

- name – Which peripheral to check, see clock_ip_name_t.

Returns

True if clock is already enabled, otherwise false.

static inline void CLOCK_SetIpSrc(*clock_ip_name_t* name, *clock_ip_src_t* src)

Set the clock source for specific IP module.

Set the clock source for specific IP, not all modules need to set the clock source, should only use this function for the modules need source setting.

Parameters

- name – Which peripheral to check, see clock_ip_name_t.
- src – Clock source to set.

static inline void CLOCK_SetIpSrcDiv(*clock_ip_name_t* name, *clock_ip_src_t* src, uint8_t divValue, uint8_t fracValue)

Set the clock source and divider for specific IP module.

Set the clock source and divider for specific IP, not all modules need to set the clock source and divider, should only use this function for the modules need source and divider setting.

Divider output clock = Divider input clock x [(fracValue+1)/(divValue+1)].

Parameters

- name – Which peripheral to check, see clock_ip_name_t.
- src – Clock source to set.
- divValue – The divider value.
- fracValue – The fraction multiply value.

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_GetCoreSysClkFreq(void)

Get the core clock or system clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetPlatClkFreq(void)

Get the platform clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetExtClkFreq(void)

Get the external clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetBusClkFreq(void)

Get the bus clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetSlowClkFreq(void)

Get the slow clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetOsc32kClkFreq(void)

Get the OSC 32K clock frequency (OSC32KCLK).

Returns
Clock frequency in Hz.

uint32_t CLOCK_GetErClkFreq(void)

Get the external reference clock frequency (ERCLK).

Returns
Clock frequency in Hz.

uint32_t CLOCK_GetLvdsClkFreq(void)

Get the external LVDS pad clock frequency (LVDS).

Returns
Clock frequency in Hz.

uint32_t CLOCK_GetIpFreq(*clock_ip_name_t* name)

Gets the clock frequency for a specific IP module.

This function gets the IP module clock frequency based on PCC registers. It is only used for the IP modules which could select clock source by PCC[PCS].

Parameters

- name – Which peripheral to get, see clock_ip_name_t.

Returns
Clock frequency value in hertz

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.3.1.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

SCG

Re-map the SCG peripheral base address for i.MX 7ULP. This driver is for SCG 0 on Core 0 of i.MX 7ULP only.

SCG_PLLPFD_PFD_VAL(pfdClkout, fracValue)

SCG (A/S)PLLPFD[PFDx] value.

SCG_PLLPFD_PFD_MASK(pfdClkout)

SCG (A/S)PLLPFD[PFD] mask.

SCG_PLLPFD_PFD_VALID_MASK(pfdClkout)

SCG (A/S)PLLPFD[PFDx_VALID] mask.

SCG_PLLPFD_PFD_CLKGATE_MASK(pfdClkout)

SCG (A/S)PLLPFD[PFDx_CLKGATE] mask.

PCC_CLKCFG_PCD_MASK

Re-define PCC register masks and bitfield operations to unify the different namings in the soc header file.

PCC_CLKCFG_PCD_SHIFT

PCC_CLKCFG_PCD(x)

PCC_CLKCFG_FRAC_MASK

PCC_CLKCFG_FRAC_SHIFT

PCC_CLKCFG_FRAC(x)

PCC_CLKCFG_PCS_MASK

PCC_CLKCFG_PCS_SHIFT

PCC_CLKCFG_PCS(x)

PCC_CLKCFG_INUSE_MASK

PCC_CLKCFG_CGC_MASK

PCC_CLKCFG_PR_MASK

DMAMUX_CLOCKS
: Clock ip name array for DMAMUX.

RGPIO2P_CLOCKS
: Clock ip name array for RGPIO2P.

SAI_CLOCKS
: Clock ip name array for SAI.

PCTL_CLOCKS
: Clock ip name array for PCTL.

LPI2C_CLOCKS
: Clock ip name array for LPI2C.

FLEXIO_CLOCKS
: Clock ip name array for FLEXIO.

EDMA_CLOCKS
: Clock ip name array for EDMA.

LPUART_CLOCKS
: Clock ip name array for LPUART.

DAC_CLOCKS
: Clock ip name array for DAC.

SNVS_HP_CLOCKS
: Clock ip name array for DAC.

SNVS_LP_CLOCKS

LPTMR_CLOCKS
: Clock ip name array for LPTMR.

LPADC_CLOCKS
: Clock ip name array for LPADC.

TRNG_CLOCKS
: Clock ip name array for TRNG.

LPSPI_CLOCKS
: Clock ip name array for LPSPI.

TPM_CLOCKS
: Clock ip name array for TPM.

LPIT_CLOCKS
: Clock ip name array for LPIT.

CRC_CLOCKS
: Clock ip name array for CRC.

CMP_CLOCKS
: Clock ip name array for CMP.

XRDC_CLOCKS
: Clock ip name array for XRDC.

MU_CLOCKS
: Clock ip name array for MU.

WDOG_CLOCKS
: Clock ip name array for WDOG.

LTC_CLOCKS
: Clock ip name array for LTC.

DPM_CLOCKS
: Clock ip name array for DPM.

SEMA42_CLOCKS
: Clock ip name array for SEMA42.

TPIU_CLOCKS
: Clock ip name array for TPIU.

QSPI_CLOCKS
: Clock ip name array for QSPI.

LPO_CLK_FREQ
: LPO clock frequency.

TPIU_CLK_FREQ
: TPIU clock frequency.

kCLOCK_Osc0ErClk
: For compatible with other MCG platforms.

kCLOCK_Er32kClk
: For compatible with other MCG platforms.

CLOCK_GetOsc0ErClkFreq
: For compatible with other MCG platforms.

CLOCK_GetEr32kClkFreq
: For compatible with other MCG platforms.

uint32_t CLOCK_GetSysClkFreq(*scg_sys_clk_t* type)
: Gets the SCG system clock frequency.

  This function gets the SCG system clock frequency. These clocks are used for core, platform, external, and bus clock domains.

  Parameters

  - type – Which type of clock to get, core clock or slow clock.

  Returns
  Clock frequency.

static inline void CLOCK_SetVlprModeSysClkConfig(const *scg_sys_clk_config_t* *config)
: Sets the system clock configuration for VLPR mode.

  This function sets the system clock configuration for VLPR mode.

  Parameters

  - config – Pointer to the configuration.

static inline void CLOCK_SetRunModeSysClkConfig(const *scg_sys_clk_config_t* *config)

Sets the system clock configuration for RUN mode.

This function sets the system clock configuration for RUN mode.

Parameters

- config – Pointer to the configuration.

static inline void CLOCK_SetHsrunModeSysClkConfig(const *scg_sys_clk_config_t* *config)

Sets the system clock configuration for HSRUN mode.

This function sets the system clock configuration for HSRUN mode.

Parameters

- config – Pointer to the configuration.

static inline void CLOCK_GetCurSysClkConfig(*scg_sys_clk_config_t* *config)

Gets the system clock configuration in the current power mode.

This function gets the system configuration in the current power mode.

Parameters

- config – Pointer to the configuration.

static inline void CLOCK_SetClkOutSel(*clock_clkout_src_t* setting)

Sets the clock out selection.

This function sets the clock out selection (CLKOUTSEL).

Parameters

- setting – The selection to set.

Returns

The current clock out selection.

status_t CLOCK_InitSysOsc(const *scg_sosc_config_t* *config)

Initializes the SCG system OSC.

This function enables the SCG system OSC clock according to the configuration.

Note: This function can't detect whether the system OSC has been enabled and used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – System OSC is initialized.
- kStatus_SCG_Busy – System OSC has been enabled and is used by the system clock.
- kStatus_ReadOnly – System OSC control register is locked.

status_t CLOCK_DeinitSysOsc(void)

De-initializes the SCG system OSC.

This function disables the SCG system OSC clock.

Note: This function can't detect whether the system OSC is used by an IP.

Return values

- kStatus_Success – System OSC is deinitialized.
- kStatus_SCG_Busy – System OSC is used by the system clock.
- kStatus_ReadOnly – System OSC control register is locked.

static inline void CLOCK_SetSysOscAsyncClkDiv(*scg_async_clk_t* asyncClk, *scg_async_clk_div_t* divider)

Set the asynchronous clock divider.

Note: There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

Parameters

- asyncClk – Which asynchronous clock to configure.
- divider – The divider value to set.

uint32_t CLOCK_GetSysOscFreq(void)

Gets the SCG system OSC clock frequency (SYSOSC).

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

uint32_t CLOCK_GetSysOscAsyncFreq(*scg_async_clk_t* type)

Gets the SCG asynchronous clock frequency from the system OSC.

Parameters

- type – The asynchronous clock type.

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

static inline bool CLOCK_IsSysOscErr(void)

Checks whether the system OSC clock error occurs.

Returns
True if the error occurs, false if not.

static inline void CLOCK_ClearSysOscErr(void)

Clears the system OSC clock error.

static inline void CLOCK_SetSysOscMonitorMode(*scg_sosc_monitor_mode_t* mode)

Sets the system OSC monitor mode.

This function sets the system OSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

static inline bool CLOCK_IsSysOscValid(void)

Checks whether the system OSC clock is valid.

Returns
True if clock is valid, false if not.

status_t CLOCK_InitSirc(const *scg_sirc_config_t* *config)

Initializes the SCG slow IRC clock.

This function enables the SCG slow IRC clock according to the configuration.

Note: This function can't detect whether the system OSC has been enabled and used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – SIRC is initialized.
- kStatus_SCG_Busy – SIRC has been enabled and is used by system clock.
- kStatus_ReadOnly – SIRC control register is locked.

status_t CLOCK_DeinitSirc(void)

De-initializes the SCG slow IRC.

This function disables the SCG slow IRC.

Note: This function can't detect whether the SIRC is used by an IP.

Return values

- kStatus_Success – SIRC is deinitialized.
- kStatus_SCG_Busy – SIRC is used by system clock.
- kStatus_ReadOnly – SIRC control register is locked.

static inline void CLOCK_SetSircAsyncClkDiv(*scg_async_clk_t* asyncClk, *scg_async_clk_div_t* divider)

Set the asynchronous clock divider.

Note: There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

Parameters

- asyncClk – Which asynchronous clock to configure.
- divider – The divider value to set.

static inline void CLOCK_EnableLpoPowerOption(bool enable)

Enables/disables the SCG slow IRC 1khz LPO clock in LLS/VLLSx modes.

This function enables/disables the SCG slow IRC 1khz LPO clock in LLS/VLLSx modes.

Parameters

- enable – Switcher of LPO Power Option which controls whether the 1 kHz LPO clock is enabled in LLS/VLLSx modes. "true" means to enable, "false" means not enabled.

uint32_t CLOCK_GetSircFreq(void)

Gets the SCG SIRC clock frequency.

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

uint32_t CLOCK_GetSircAsyncFreq(*scg_async_clk_t* type)

Gets the SCG asynchronous clock frequency from the SIRC.

Parameters

- type – The asynchronous clock type.

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

static inline bool CLOCK_IsSircValid(void)

Checks whether the SIRC clock is valid.

Returns
True if clock is valid, false if not.

status_t CLOCK_InitFirc(const *scg_firc_config_t* *config)

Initializes the SCG fast IRC clock.

This function enables the SCG fast IRC clock according to the configuration.

Note: This function can't detect whether the FIRC has been enabled and used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – FIRC is initialized.
- kStatus_SCG_Busy – FIRC has been enabled and is used by the system clock.
- kStatus_ReadOnly – FIRC control register is locked.

status_t CLOCK_DeinitFirc(void)

De-initializes the SCG fast IRC.

This function disables the SCG fast IRC.

Note: This function can't detect whether the FIRC is used by an IP.

Return values

- kStatus_Success – FIRC is deinitialized.
- kStatus_SCG_Busy – FIRC is used by the system clock.
- kStatus_ReadOnly – FIRC control register is locked.

static inline void CLOCK_SetFircAsyncClkDiv(*scg_async_clk_t* asyncClk, *scg_async_clk_div_t* divider)

Set the asynchronous clock divider.

Note: There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

Parameters

- asyncClk – Which asynchronous clock to configure.
- divider – The divider value to set.

uint32_t CLOCK_GetFircFreq(void)

Gets the SCG FIRC clock frequency.

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

uint32_t CLOCK_GetFircAsyncFreq(*scg_async_clk_t* type)

Gets the SCG asynchronous clock frequency from the FIRC.

Parameters

- type – The asynchronous clock type.

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

static inline bool CLOCK_IsFircErr(void)

Checks whether the FIRC clock error occurs.

Returns
True if the error occurs, false if not.

static inline void CLOCK_ClearFircErr(void)

Clears the FIRC clock error.

static inline bool CLOCK_IsFircValid(void)

Checks whether the FIRC clock is valid.

Returns
True if clock is valid, false if not.

uint32_t CLOCK_GetRtcOscFreq(void)

Gets the SCG RTC OSC clock frequency.

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

static inline bool CLOCK_IsRtcOscErr(void)

Checks whether the RTC OSC clock error occurs.

Returns
True if error occurs, false if not.

static inline void CLOCK_ClearRtcOscErr(void)

Clears the RTC OSC clock error.

static inline void CLOCK_SetRtcOscMonitorMode(*scg_rosc_monitor_mode_t* mode)

Sets the RTC OSC monitor mode.

This function sets the RTC OSC monitor mode. The mode can be disabled. It can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

static inline bool CLOCK_IsRtcOscValid(void)

Checks whether the RTC OSC clock is valid.

Returns

True if the clock is valid, false if not.

status_t CLOCK_InitAuxPll(const *scg_apll_config_t* *config)

Initializes the SCG auxiliary PLL.

This function enables the SCG auxiliary PLL clock according to the configuration. The auxiliary PLL can use the system OSC or FIRC as the clock source. Ensure that the source clock is valid before calling this function.

Example code for initializing APLL clock output:

```
const scg_apll_config_t g_scgAuxPllConfig = {.enableMode = kSCG_AuxPllEnable,
                                             .div1 = kSCG_AsyncClkDisable,
                                             .div2 = kSCG_AsyncClkDisable,
                                             .div3 = kSCG_AsyncClkDisable,
                                             .src = kSCG_SysPllSrcFirc,
                                             .isPfdSelected = false,
                                             .prediv = 5U,
                                             .pfdClkout = kSCG_AuxPllPfd0Clk,
                                             .mult = 20U,
                                             .pllPostdiv1 = kSCG_SysClkDivBy3,
                                             .pllPostdiv2 = kSCG_SysClkDivBy4,
                                             .num = 578,
                                             .denom = 1000};
CLOCK_InitAuxPll(&g_scgAuxPllConfig);
```

Note: This function can't detect whether the auxiliary PLL has been enabled and is used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – auxiliary PLL is initialized.
- kStatus_SCG_Busy – auxiliary PLL has been enabled and is used by the system clock.
- kStatus_ReadOnly – auxiliary PLL control register is locked.

status_t CLOCK_DeinitAuxPll(void)

De-initializes the SCG auxiliary PLL.

This function disables the SCG auxiliary PLL.

Note: This function can't detect whether the auxiliary PLL is used by an IP.

Return values

- kStatus_Success – auxiliary PLL is deinitialized.
- kStatus_SCG_Busy – auxiliary PLL is used by the system clock.
- kStatus_ReadOnly – auxiliary PLL control register is locked.

static inline void CLOCK_SetAuxPllAsyncClkDiv(*scg_async_clk_t* asyncClk, *scg_async_clk_div_t* divider)

Set the asynchronous clock divider.

Note: There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

Parameters

- asyncClk – Which asynchronous clock to configure.
- divider – The divider value to set.

uint32_t CLOCK_GetAuxPllFreq(void)

Gets the SCG auxiliary PLL clock frequency.

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

uint32_t CLOCK_GetAuxPllAsyncFreq(*scg_async_clk_t* type)

Gets the SCG asynchronous clock frequency from the auxiliary PLL.

Parameters

- type – The asynchronous clock type.

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

uint32_t CLOCK_GetAuxPllPfdFreq(*scg_apll_pfd_clkout_t* pfdClkout)

Gets the SCG auxiliary PLL PFD clock frequency.

Parameters

- pfdClkout – The selected PFD clocks out. See “scg_apll_pfd_clkout_t”.

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

void CLOCK_EnableAuxPllPfdClkout(*scg_apll_pfd_clkout_t* pfdClkout, uint8_t fracValue)

Enables the SCG auxiliary PLL Fractional Divide (PFD) clock out with configurations.

APLL Frequency = Fref * (MULT + NUM/DENOM) PFD Clock Frequency = PLL output frequency * 18/frac value

Example code for configuring APLL as APLL PFD clock output:

```
const scg_apll_config_t g_scgAuxPllConfig = {.enableMode = kSCG_AuxPllEnable,
                                             .div1 = kSCG_AsyncClkDisable,
                                             .div2 = kSCG_AsyncClkDisable,
                                             .div3 = kSCG_AsyncClkDisable,
                                             .src = kSCG_SysPllSrcFirc,
                                             .isPfdSelected = true,
                                             .prediv = 5U,
                                             .pfdClkout = kSCG_AuxPllPfd3Clk,
clock
                                             .mult = 20U,
                                             .pllPostdiv1 = kSCG_SysClkDivBy1,
                                             .pllPostdiv2 = kSCG_SysClkDivBy1,
                                             .num = 578,
                                             .denom = 1000};
CLOCK_InitAuxPll(&g_scgAuxPllConfig);
CLOCK_EnableAuxPllPfdClkout(g_scgAuxPllConfig.pfdClkout, 15U);
```

Parameters

- pfdClkout – APLL PFD clock out select.
- fracValue – Fractional Divider value. Recommended to be kept between 12-35 for all PFDs.

static inline void CLOCK_DisableAuxPllPfdClkout(*scg_apll_pfd_clkout_t* pfdClkout)

Disables the SCG auxiliary PLL Fractional Divide (PFD) clock out.

static inline void CLOCK_EnableAuxPllSpectrumModulation(uint16_t step, uint16_t stop)

Enables the SCG auxiliary PLL spread spectrum modulation feature with configurations.

This function sets the SCG auxiliary PLL spread spectrum modulation configurations. STOP and STEP together control the modulation depth (maximum frequency change) and modulation frequency.

Modulation Depth = (STOP/MFD)*Fref where MFD is the DENOM field value in DENOM register. Modulation Frequency = (STEP/(2*STOP))*Fref.

Parameters

- step – APLL Spread Spectrum STEP.
- stop – APLL Spread Spectrum STOP.

static inline void CLOCK_DisableAuxPllSpectrumModulation(void)

Disables the SCG auxiliary PLL spread spectrum modulation.

static inline void CLOCK_SetAuxPllLockTime(uint16_t lockTime)

Sets the SCG auxiliary PLL lock time.

Parameters

- lockTime – Reference clocks to count before APLL is considered locked and valid.

static inline bool CLOCK_IsAuxPllValid(void)

Checks whether the auxiliary PLL clock is valid.

Returns

True if the clock is valid, false if not.

status_t CLOCK_InitSysPll(const *scg_spll_config_t* *config)

Initializes the SCG system PLL.

This function enables the SCG system PLL clock according to the configuration. The system PLL can use the system OSC or FIRC as the clock source. Ensure that the source clock is valid before calling this function.

Example code for initializing SPLL clock output:

```
const scg_spll_config_t g_scgSysPllConfig = {.enableMode = kSCG_SysPllEnable,
                                             .div1 = kSCG_AsyncClkDivBy1,
                                             .div2 = kSCG_AsyncClkDisable,
                                             .div3 = kSCG_AsyncClkDivBy2,
                                             .src = kSCG_SysPllSrcFirc,
                                             .isPfdSelected = false,
                                             .prediv = 0U,
                                             .pfdClkout = kSCG_SysPllPfd0Clk,
                                             .mult = 3U};
CLOCK_InitSysPll(&g_scgSysPllConfig);
```

Note: This function can't detect whether the system PLL has been enabled and used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – System PLL is initialized.
- kStatus_SCG_Busy – System PLL has been enabled and is used by the system clock.
- kStatus_ReadOnly – System PLL control register is locked.

status_t CLOCK_DeinitSysPll(void)

De-initializes the SCG system PLL.

This function disables the SCG system PLL.

Note: This function can't detect whether the system PLL is used by an IP.

Return values

- kStatus_Success – system PLL is deinitialized.
- kStatus_SCG_Busy – system PLL is used by the system clock.
- kStatus_ReadOnly – System PLL control register is locked.

static inline void CLOCK_SetSysPllAsyncClkDiv(*scg_async_clk_t* asyncClk, *scg_async_clk_div_t* divider)

Set the asynchronous clock divider.

Note: There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

Parameters

- asyncClk – Which asynchronous clock to configure.
- divider – The divider value to set.

uint32_t CLOCK_GetSysPllFreq(void)

Gets the SCG system PLL clock frequency.

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

uint32_t CLOCK_GetSysPllAsyncFreq(*scg_async_clk_t* type)

Gets the SCG asynchronous clock frequency from the system PLL.

Parameters

- type – The asynchronous clock type.

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

uint32_t CLOCK_GetSysPllPfdFreq(*scg_spll_pfd_clkout_t* pfdClkout)

Gets the SCG system PLL PFD clock frequency.

Parameters

- pfdClkout – The selected PFD clock out. See "scg_spll_pfd_clkout_t".

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

void CLOCK_EnableSysPllPfdClkout(*scg_spll_pfd_clkout_t* pfdClkout, uint8_t fracValue)

Enables the SCG system PLL Fractional Divide (PFD) clock out with configurations.

SPLL Frequency = Fref * (MULT + NUM/DENOM) PFD Clock Frequency = PLL output frequency * 18/frac value

```
Example code for configuring SPLL as SPLL PFD clock output:
const scg_spll_config_t g_scgSysPllConfig = {.enableMode = kSCG_SysPllEnable,
                                             .div1 = kSCG_AsyncClkDisable,
                                             .div2 = kSCG_AsyncClkDisable,
                                             .div3 = kSCG_AsyncClkDisable,
                                             .src = kSCG_SysPllSrcFirc,
                                             .isPfdSelected = true,
                                             .prediv = 5U,
                                             .pfdClkout = kSCG_AuxPllPfd3Clk,
clock
                                             .mult = 20U};
CLOCK_InitSysPll(&g_scgSysPllConfig);
CLOCK_EnableSysPllPfdClkout(g_scgSysPllConfig.pfdClkout, 15U);
```

Parameters

- pfdClkout – SPLL PFD clock out select.
- fracValue – Fractional Divider value. Recommended to be kept between 12-35 for all PFDs.

static inline void CLOCK_DisableSysPllPfdClkout(*scg_spll_pfd_clkout_t* pfdClkout)

Disables the SCG system PLL Fractional Divide (PFD) clock out.

static inline bool CLOCK_IsSysPllErr(void)

Checks whether the system PLL clock error occurs.

Returns
True if an error occurs, false if not.

static inline void CLOCK_ClearSysPllErr(void)

Clears the system PLL clock error.

static inline bool CLOCK_IsSysPllValid(void)

Checks whether the system PLL clock is valid.

Returns
True if the clock is valid, false if not.

static inline void CLOCK_SetXtal0Freq(uint32_t freq)

Sets the XTAL0 frequency based on board settings.

Parameters

- freq – The XTAL0/EXTAL0 input clock frequency in Hz.

static inline void CLOCK_SetXtal32Freq(uint32_t freq)

Sets the XTAL32 frequency based on board settings.

Parameters

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

static inline void CLOCK_SetLvdsFreq(uint32_t freq)

Sets the LVDS pad frequency based on board settings.

Parameters

- freq – The LVDS pad input clock frequency in Hz.

uint32_t divSlow
: Slow clock divider, see scg_sys_clk_div_t.

uint32_t divBus
: Bus clock divider, see scg_sys_clk_div_t.

uint32_t ___pad0___
: Reserved.

uint32_t divPlat
: Platform clock divider, which can only be divided by 1. See kSCG_SysClkDivBy1.

uint32_t divCore
: Core clock divider, see scg_sys_clk_div_t.

uint32_t ___pad1___
: Reserved.

uint32_t src
: System clock source, see scg_sys_clk_src_t.

uint32_t ___pad2___
: reserved.

uint32_t freq
: System OSC frequency.

scg_sosc_monitor_mode_t monitorMode
: Clock monitor mode selected.

uint8_t enableMode
: Enable mode, OR'ed value of _scg_sosc_enable_mode.

scg_async_clk_div_t div1
: SOSCDIV1 value.

scg_async_clk_div_t div2
: SOSCDIV2 value.

scg_async_clk_div_t div3
: SOSCDIV3 value.

scg_sosc_mode_t workMode
: OSC work mode.

uint32_t enableMode
: Enable mode, OR'ed value of _scg_sirc_enable_mode.

scg_async_clk_div_t div1
: SIRCDIV1 value.

scg_async_clk_div_t div2
: SIRCDIV2 value.

scg_async_clk_div_t div3
: SIRCDIV3 value.

scg_sirc_range_t range
: Slow IRC frequency range.

scg_firc_trim_mode_t trimMode
FIRC trim mode.

scg_firc_trim_src_t trimSrc
Trim source.

scg_firc_trim_div_t trimDiv
Trim predivided value for the system OSC.

uint8_t trimCoar
Trim coarse value; Irrelevant if trimMode is kSCG_FircTrimUpdate.

uint8_t trimFine
Trim fine value; Irrelevant if trimMode is kSCG_FircTrimUpdate.

uint32_t enableMode
Enable mode, OR'ed value of _scg_firc_enable_mode.

scg_async_clk_div_t div1
FIRCDIV1 value.

scg_async_clk_div_t div2
FIRCDIV2 value.

scg_async_clk_div_t div3
FIRCDIV3 value.

scg_firc_range_t range
Fast IRC frequency range.

const *scg_firc_trim_config_t* *trimConfig
Pointer to the FIRC trim configuration; set NULL to disable trim.

uint8_t enableMode
Enable mode, OR'ed value of _scg_spll_enable_mode

scg_async_clk_div_t div1
SPLLDIV1 value.

scg_async_clk_div_t div2
SPLLDIV2 value.

scg_async_clk_div_t div3
SPLLDIV3 value.

scg_spll_src_t src
Clock source.

bool isPfdSelected
SPLL PFD output clock selected.

uint8_t prediv
PLL reference clock divider.

scg_spll_pfd_clkout_t pfdClkout
PLL PFD clouk out select.

uint8_t mult
System PLL multiplier.

scg_rosc_monitor_mode_t monitorMode
Clock monitor mode selected.

uint8_t enableMode
: Enable mode, OR'ed value of _scg_apll_enable_mode

scg_async_clk_div_t div1
: APLLDIV1 value.

scg_async_clk_div_t div2
: APLLDIV2 value.

scg_async_clk_div_t div3
: APLLDIV3 value.

scg_apll_src_t src
: Clock source.

bool isPfdSelected
: APLL PFD output clock selected.

uint8_t prediv
: PLL reference clock divider.

scg_apll_pfd_clkout_t pfdClkout
: SCG auxiliary PLL PFD clouk out select.

uint8_t mult
: Auxiliary PLL multiplier.

scg_sys_clk_div_t pllPostdiv1
: Auxiliary PLL Post Clock Divide1 Ratio.

scg_sys_clk_div_t pllPostdiv2
: Auxiliary PLL Post Clock Divide2 Ratio.

uint32_t num
: 30-bit numerator of the Auxiliary PLL Fractional-Loop divider.

uint32_t denom
: 30-bit denominator of the Auxiliary PLL Fractional-Loop divider.

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL
: Configure whether driver controls clock.

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

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

  ---

struct _scg_sys_clk_config
: *#include <fsl_clock.h>* SCG system clock configuration.

struct _scg_sosc_config
: *#include <fsl_clock.h>* SCG system OSC configuration.

struct _scg_sirc_config
: *#include <fsl_clock.h>* SCG slow IRC clock configuration.

struct _scg_firc_trim_config
: *#include <fsl_clock.h>* SCG fast IRC clock trim configuration.

struct _scg_firc_config_t
#include <fsl_clock.h> SCG fast IRC clock configuration.

struct _scg_spll_config
#include <fsl_clock.h> SCG system PLL configuration.

struct _scg_rosc_config
#include <fsl_clock.h> SCG RTC OSC configuration.

struct _scg_apll_config
#include <fsl_clock.h> SCG auxiliary PLL configuration.

2.4 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION
CRC driver version. Version 2.1.0.

Current version: 2.1.0

Change log:

- Version 2.1.0
  - Choosing CRC clocks from CRC clock array according to instance instead of hard-coded value.
- Version 2.0.5
  - Fix CERT-C issue with boolean-to-unsigned integer conversion.
- Version 2.0.4
  - Release peripheral from reset if necessary in init function.
- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues
- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

enum _crc_bits
CRC bit width.

Values:

enumerator kCrcBits16
Generate 16-bit CRC code

enumerator kCrcBits32
Generate 32-bit CRC code

enum _crc_result
CRC result type.

Values:

enumerator kCrcFinalChecksum

CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

enumerator kCrcIntermediateChecksum

CRC data register read value is intermediate checksum (raw value). Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for CRC_Init() to continue adding data to this checksum.

typedef enum *_crc_bits* crc_bits_t

CRC bit width.

typedef enum *_crc_result* crc_result_t

CRC result type.

typedef struct *_crc_config* crc_config_t

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void CRC_Init(CRC_Type *base, const *crc_config_t* *config)

Enables and configures the CRC peripheral module.

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

- base – CRC peripheral address.
- config – CRC module configuration structure.

void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

This function disables the clock gate in the SIM module for the CRC peripheral.

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

Loads default values to the CRC protocol configuration structure. The default values are as follows.

```
config->polynomial = 0x1021;
config->seed = 0xFFFF;
config->reflectIn = false;
config->reflectOut = false;
config->complementChecksum = false;
config->crcBits = kCrcBits16;
config->crcResult = kCrcFinalChecksum;
```

Parameters

- config – CRC protocol configuration structure.

void CRC_WriteData(CRC_Type *base, const uint8_t *data, size_t dataSize)

Writes data to the CRC module.

Writes input data buffer bytes to the CRC data register. The configured type of transpose is applied.

Parameters

- base – CRC peripheral address.
- data – Input data stream, MSByte in data[0].
- dataSize – Size in bytes of the input data buffer.

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 32-bit checksum, after configured transpose and complement operations.

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 16-bit checksum, after configured transpose and complement operations.

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

Default configuration structure filled by CRC_GetDefaultConfig(). Use CRC16-CCIT-FALSE as defeault.

struct __crc_config

#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

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

uint32_t seed

Starting checksum value

bool reflectIn

Reflect bits on input.

bool reflectOut

Reflect bits on output.

bool complementChecksum

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

crc_bits_t crcBits

Selects 16- or 32- bit CRC protocol.

crc_result_t crcResult
Selects final or intermediate checksum return from CRC_Get16bitResult() or CRC_Get32bitResult()

2.5 DAC12: 12-bit Digital-to-Analog Converter Driver

void DAC12_GetHardwareInfo(DAC_Type *base, *dac12_hardware_info_t* *info)
Get hardware information about this module.

Parameters

- base – DAC12 peripheral base address.
- info – Pointer to info structure, see to dac12_hardware_info_t.

void DAC12_Init(DAC_Type *base, const *dac12_config_t* *config)
Initialize the DAC12 module.

Parameters

- base – DAC12 peripheral base address.
- config – Pointer to configuration structure, see to dac12_config_t.

void DAC12_GetDefaultConfig(*dac12_config_t* *config)
Initializes the DAC12 user configuration structure.

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

```
config->fifoWatermarkLevel = 0U;
config->fifoWorkMode = kDAC12_FIFODisabled;
config->referenceVoltageSource = kDAC12_ReferenceVoltageSourceAlt1;
config->fifoTriggerMode = kDAC12_FIFOTriggerByHardwareMode;
config->referenceCurrentSource = kDAC12_ReferenceCurrentSourceAlt0;
config->speedMode = kDAC12_SpeedLowMode;
config->speedMode = false;
config->currentReferenceInternalTrimValue = 0x4;
```

Parameters

- config – Pointer to the configuration structure. See “dac12_config_t”.

void DAC12_Deinit(DAC_Type *base)
De-initialize the DAC12 module.

Parameters

- base – DAC12 peripheral base address.

static inline void DAC12_Enable(DAC_Type *base, bool enable)
Enable the DAC12’s converter or not.

Parameters

- base – DAC12 peripheral base address.
- enable – Enable the DAC12’s converter or not.

static inline void DAC12_ResetConfig(DAC_Type *base)
Reset all internal logic and registers.

Parameters

- base – DAC12 peripheral base address.

static inline void DAC12_ResetFIFO(DAC_Type *base)

Reset the FIFO pointers.

FIFO pointers should only be reset when the DAC12 is disabled. This function can be used to configure both pointers to the same address to reset the FIFO as empty.

Parameters

- base – DAC12 peripheral base address.

static inline uint32_t DAC12_GetStatusFlags(DAC_Type *base)

Get status flags.

Parameters

- base – DAC12 peripheral base address.

Returns

Mask of current status flags. See to _dac12_status_flags.

static inline void DAC12_ClearStatusFlags(DAC_Type *base, uint32_t flags)

Clear status flags.

Note: Not all the flags can be cleared by this API. Several flags need special condition to clear them according to target chip's reference manual document.

Parameters

- base – DAC12 peripheral base address.
- flags – Mask of status flags to be cleared. See to _dac12_status_flags.

static inline void DAC12_EnableInterrupts(DAC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – DAC12 peripheral base address.
- mask – Mask value of interrupts to be enabled. See to _dac12_interrupt_enable.

static inline void DAC12_DisableInterrupts(DAC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – DAC12 peripheral base address.
- mask – Mask value of interrupts to be disabled. See to _dac12_interrupt_enable.

static inline void DAC12_EnableDMA(DAC_Type *base, bool enable)

Enable DMA or not.

When DMA is enabled, the DMA request will be generated by original interrupts. The interrupts will not be presented on this module at the same time.

static inline void DAC12_SetData(DAC_Type *base, uint32_t value)

Set data into the entry of FIFO buffer.

When the DAC FIFO is disabled, and the one entry buffer is enabled, the DAC converts the data in the buffer to analog output voltage. Any write to the DATA register will replace the data in the buffer and push data to analog conversion without trigger support. When the DAC FIFO is enabled. Writing data would increase the write pointer of FIFO. Also, the data would be restored into the FIFO buffer.

Parameters

- base – DAC12 peripheral base address.
- value – Setting value into FIFO buffer.

static inline void DAC12_DoSoftwareTrigger(DAC_Type *base)

Do trigger the FIFO by software.

When the DAC FIFO is enabled, and software trigger is used. Doing trigger would increase the read pointer, and the data in the entry pointed by read pointer would be converted as new output.

Parameters

- base – DAC12 peripheral base address.

static inline uint32_t DAC12_GetFIFOReadPointer(DAC_Type *base)

Get the current read pointer of FIFO.

Parameters

- base – DAC12 peripheral base address.

Returns

Read pointer index of FIFO buffer.

static inline uint32_t DAC12_GetFIFOWritePointer(DAC_Type *base)

Get the current write pointer of FIFO.

Parameters

- base – DAC12 peripheral base address.

Returns

Write pointer index of FIFO buffer

FSL_DAC12_DRIVER_VERSION

DAC12 driver version 2.1.2.

enum _dac12_status_flags

DAC12 flags.

Values:

enumerator kDAC12_OverflowFlag

FIFO overflow status flag, which indicates that more data has been written into FIFO than it can hold.

enumerator kDAC12_UnderflowFlag

FIFO underflow status flag, which means that there is a new trigger after the FIFO is nearly empty.

enumerator kDAC12_WatermarkFlag

FIFO wartermark status flag, which indicates the remaining FIFO data is less than the watermark setting.

enumerator kDAC12_NearlyEmptyFlag

FIFO nearly empty flag, which means there is only one data remaining in FIFO.

enumerator kDAC12_FullFlag

FIFO full status flag, which means that the FIFO read pointer equals the write pointer, as the write pointer increase.

enum _dac12_interrupt_enable

DAC12 interrupts.

Values:

enumerator kDAC12_UnderOrOverflowInterruptEnable
Underflow and overflow interrupt enable.

enumerator kDAC12_WatermarkInterruptEnable
Watermark interrupt enable.

enumerator kDAC12_NearlyEmptyInterruptEnable
Nearly empty interrupt enable.

enumerator kDAC12_FullInterruptEnable
Full interrupt enable.

enum _dac12_fifo_size_info
DAC12 FIFO size information provided by hardware.

Values:

enumerator kDAC12_FIFOSize2
FIFO depth is 2.

enumerator kDAC12_FIFOSize4
FIFO depth is 4.

enumerator kDAC12_FIFOSize8
FIFO depth is 8.

enumerator kDAC12_FIFOSize16
FIFO depth is 16.

enumerator kDAC12_FIFOSize32
FIFO depth is 32.

enumerator kDAC12_FIFOSize64
FIFO depth is 64.

enumerator kDAC12_FIFOSize128
FIFO depth is 128.

enumerator kDAC12_FIFOSize256
FIFO depth is 256.

enum _dac12_fifo_work_mode
DAC12 FIFO work mode.

Values:

enumerator kDAC12_FIFODisabled
FIFO disabled and only one level buffer is enabled. Any data written from this buffer goes to conversion.

enumerator kDAC12_FIFOWorkAsNormalMode
Data will first read from FIFO to buffer then go to conversion.

enumerator kDAC12_FIFOWorkAsSwingMode
In Swing mode, the FIFO must be set up to be full. In Swing back mode, a trigger changes the read pointer to make it swing between the FIFO Full and Nearly Empty state. That is, the trigger increases the read pointer till FIFO is nearly empty and decreases the read pointer till the FIFO is full.

enum _dac12_reference_voltage_source
DAC12 reference voltage source.

Values:

enumerator kDAC12_ReferenceVoltageSourceAlt1
The DAC selects DACREF_1 as the reference voltage.

enumerator kDAC12_ReferenceVoltageSourceAlt2
The DAC selects DACREF_2 as the reference voltage.

enum _dac12_fifo_trigger_mode
DAC12 FIFO trigger mode.

Values:

enumerator kDAC12_FIFOTriggerByHardwareMode
Buffer would be triggered by hardware.

enumerator kDAC12_FIFOTriggerBySoftwareMode
Buffer would be triggered by software.

enum _dac12_reference_current_source
DAC internal reference current source.

Analog module needs reference current to keep working . Such reference current can generated by IP itself, or by on-chip PMC's "reference part". If no current reference be selected, analog module can't working normally ,even when other register can still be assigned, DAC would waste current but no function. To make the DAC work, either kDAC12_ReferenceCurrentSourceAltx should be selected.

Values:

enumerator kDAC12_ReferenceCurrentSourceDisabled
None of reference current source is enabled.

enumerator kDAC12_ReferenceCurrentSourceAlt0
Use the internal reference current generated by the module itself.

enumerator kDAC12_ReferenceCurrentSourceAlt1
Use the ZTC(Zero Temperature Coefficient) reference current generated by on-chip power management module.

enumerator kDAC12_ReferenceCurrentSourceAlt2
Use the PTAT(Proportional To Absolution Temperature) reference current generated by power management module.

enum _dac12_speed_mode
DAC analog buffer speed mode for conversion.

Values:

enumerator kDAC12_SpeedLowMode
Low speed mode.

enumerator kDAC12_SpeedMiddleMode
Middle speed mode.

enumerator kDAC12_SpeedHighMode
High speed mode.

typedef enum *_dac12_fifo_size_info* dac12_fifo_size_info_t
DAC12 FIFO size information provided by hardware.

typedef enum *_dac12_fifo_work_mode* dac12_fifo_work_mode_t
DAC12 FIFO work mode.

typedef enum *_dac12_reference_voltage_source* dac12_reference_voltage_source_t
DAC12 reference voltage source.

typedef enum *_dac12_fifo_trigger_mode* dac12_fifo_trigger_mode_t

DAC12 FIFO trigger mode.

typedef enum *_dac12_reference_current_source* dac12_reference_current_source_t

DAC internal reference current source.

Analog module needs reference current to keep working . Such reference current can generated by IP itself, or by on-chip PMC's "reference part". If no current reference be selected, analog module can't working normally ,even when other register can still be assigned, DAC would waste current but no function. To make the DAC work, either kDAC12_ReferenceCurrentSourceAltx should be selected.

typedef enum *_dac12_speed_mode* dac12_speed_mode_t

DAC analog buffer speed mode for conversion.

typedef struct *_dac12_hardware_info* dac12_hardware_info_t

DAC12 hardware information.

DAC12_CR_W1C_FLAGS_MASK

Define "write 1 to clear" flags.

DAC12_CR_ALL_FLAGS_MASK

Define all the flag bits in DACx_CR register.

struct _dac12_hardware_info

#include <fsl_dac12.h> DAC12 hardware information.

Public Members

dac12_fifo_size_info_t fifoSizeInfo

The number of words in this device's DAC buffer.

struct dac12_config_t

#include <fsl_dac12.h> DAC12 module configuration.

Actually, the most fields are for FIFO buffer.

Public Members

uint32_t fifoWatermarkLevel

FIFO's watermark, the max value can be the hardware FIFO size.

dac12_fifo_work_mode_t fifoWorkMode

FIFI's work mode about pointers.

dac12_reference_voltage_source_t referenceVoltageSource

Select the reference voltage source.

dac12_reference_current_source_t referenceCurrentSource

Select the trigger mode for FIFO. Select the reference current source.

dac12_speed_mode_t speedMode

Select the speed mode for conversion.

bool enableAnalogBuffer

Enable analog buffer for high drive.

2.6 DMAMUX: Direct Memory Access Multiplexer Driver

void DMAMUX_Init(DMAMUX_Type *base)

Initializes the DMAMUX peripheral.

This function ungates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

void DMAMUX_Deinit(DMAMUX_Type *base)

Deinitializes the DMAMUX peripheral.

This function gates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

static inline void DMAMUX_EnableChannel(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX channel.

This function enables the DMAMUX channel.

Parameters

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

static inline void DMAMUX_DisableChannel(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX channel.

This function disables the DMAMUX channel.

Note: The user must disable the DMAMUX channel before configuring it.

Parameters

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

static inline void DMAMUX_SetSource(DMAMUX_Type *base, uint32_t channel, int32_t source)

Configures the DMAMUX channel source.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- source – Channel source, which is used to trigger the DMA transfer.User need to use the dma_request_source_t type as the input parameter.

static inline void DMAMUX_EnablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX period trigger.

This function enables the DMAMUX period trigger feature.

Parameters

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

static inline void DMAMUX_DisablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX period trigger.

This function disables the DMAMUX period trigger.

Parameters

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

static inline void DMAMUX_EnableAlwaysOn(DMAMUX_Type *base, uint32_t channel, bool enable)

Enables the DMA channel to be always ON.

This function enables the DMAMUX channel always ON feature.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- enable – Switcher of the always ON feature. "true" means enabled, "false" means disabled.

FSL_DMAMUX_DRIVER_VERSION

DMAMUX driver version 2.1.4.

DMAMUX_CHANNEL_ENDIAN_CONVERTn(channel)

Macro used for dmamux channel endian convert.

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

void EDMA_Init(DMA_Type *base, const *edma_config_t* *config)

Initializes the eDMA peripheral.

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

Note: This function enables the minor loop map feature.

Parameters

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

void EDMA_Deinit(DMA_Type *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

void EDMA_InstallTCD(DMA_Type *base, uint32_t channel, *edma_tcd_t* *tcd)

Push content of TCD structure into hardware TCD register.

Parameters

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

void EDMA_GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

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

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

Parameters

- config – A pointer to the eDMA configuration structure.

static inline void EDMA_EnableContinuousChannelLinkMode(DMA_Type *base, bool enable)

Enable/Disable continuous channel link mode.

Note: Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

static inline void EDMA_EnableMinorLoopMapping(DMA_Type *base, bool enable)

Enable/Disable minor loop mapping.

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

void EDMA_ResetChannel(DMA_Type *base, uint32_t channel)

Sets all TCD registers to default values.

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

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

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.

- channel – eDMA channel number.

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

Configures the eDMA transfer attribute.

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

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

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

Parameters

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

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

Configures the eDMA minor offset feature.

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

Parameters

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

void EDMA_SetChannelPreemptionConfig(DMA_Type *base, uint32_t channel, const *edma_channel_Preemption_config_t* *config)

Configures the eDMA channel preemption feature.

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

Parameters

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

void EDMA_SetChannelLink(DMA_Type *base, uint32_t channel, *edma_channel_link_type_t* linkType, uint32_t linkedChannel)

Sets the channel link for the eDMA transfer.

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

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

Parameters

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

void EDMA_SetBandWidth(DMA_Type *base, uint32_t channel, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA transfer.

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

Parameters

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

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

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

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

Parameters

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

static inline void EDMA_EnableAsyncRequest(DMA_Type *base, uint32_t channel, bool enable)

Enables an async request for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.

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

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

Enables an auto stop request for the eDMA transfer.

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

Parameters

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

void EDMA_EnableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)

Enables the interrupt source for the eDMA transfer.

Parameters

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

void EDMA_DisableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)

Disables the interrupt source for the eDMA transfer.

Parameters

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

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

Configures the eDMA channel TCD major offset feature.

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

Parameters

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

void EDMA_TcdReset(*edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

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

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

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

Configures the eDMA TCD transfer attribute.

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

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

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

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

Parameters

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

void EDMA_TcdSetMinorOffsetConfig(*edma_tcd_t* *tcd, const *edma_minor_offset_config_t* *config)

Configures the eDMA TCD minor offset feature.

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

Parameters

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

void EDMA_TcdSetChannelLink(*edma_tcd_t* *tcd, *edma_channel_link_type_t* linkType, uint32_t linkedChannel)

Sets the channel link for the eDMA TCD.

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

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

Parameters

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

static inline void EDMA_TcdSetBandWidth(*edma_tcd_t* *tcd, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

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

Parameters

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

void EDMA_TcdSetModulo(*edma_tcd_t* *tcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

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

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

Parameters

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

static inline void EDMA_TcdEnableAutoStopRequest(*edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

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

Parameters

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

void EDMA_TcdEnableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

Parameters

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

void EDMA_TcdDisableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Disables the interrupt source for the eDMA TCD.

Parameters

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

void EDMA_TcdSetMajorOffsetConfig(*edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

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

Parameters

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

static inline void EDMA_EnableChannelRequest(DMA_Type *base, uint32_t channel)

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

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

static inline void EDMA_DisableChannelRequest(DMA_Type *base, uint32_t channel)

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

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

static inline void EDMA_TriggerChannelStart(DMA_Type *base, uint32_t channel)

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

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

uint32_t EDMA_GetRemainingMajorLoopCount(DMA_Type *base, uint32_t channel)

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

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

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

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

Parameters

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

Returns

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

static inline uint32_t EDMA_GetErrorStatusFlags(DMA_Type *base)

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

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

uint32_t EDMA_GetChannelStatusFlags(DMA_Type *base, uint32_t channel)

Gets the eDMA channel status flags.

Parameters

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

Returns

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

void EDMA_ClearChannelStatusFlags(DMA_Type *base, uint32_t channel, uint32_t mask)

Clears the eDMA channel status flags.

Parameters

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

void EDMA_CreateHandle(*edma_handle_t* *handle, DMA_Type *base, uint32_t channel)

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.

- channel – eDMA channel number.

void EDMA_InstallTCDMemory(*edma_handle_t* *handle, *edma_tcd_t* *tcdPool, uint32_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the EDMA_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA_SubmitTransfer.

Parameters

- handle – eDMA handle pointer.
- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

void EDMA_SetCallback(*edma_handle_t* *handle, *edma_callback* callback, void *userData)

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.
- userData – A parameter for the callback function.

void EDMA_PrepareTransferConfig(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type edma_transfer_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.

void EDMA_PrepareTransfer(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, *edma_transfer_type_t* transferType)

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type edma_transfer_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- transferType – eDMA transfer type.

status_t EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

void EDMA_StartTransfer(*edma_handle_t* *handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

void EDMA_StopTransfer(*edma_handle_t* *handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call EDMA_StartTransfer() again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

void EDMA_AbortTransfer(*edma_handle_t* *handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns

The next TCD address.

void EDMA_HandleIRQ(*edma_handle_t* *handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the “tcdUsed” updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the “eDMA basic data flow” in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- handle – eDMA handle pointer.

void EDMA_DriverIRQHandler(uint32_t instance, uint32_t channel)

EDMA IRQ handler for a given instance and channel.

This is a common parameterized IRQ handler entry that dispatches to the appropriate channel handle based on the instance and channel number.

Parameters

- instance – EDMA instance number.
- channel – EDMA channel number.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.4.9.

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enum _edma_modulo

eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable

Disable modulo

enumerator kEDMA_Modulo2bytes

Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes

Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes

Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes

Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes

Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes

Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone
No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle
eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type
Channel link type.

Values:

enumerator kEDMA_LinkNone
No channel link

enumerator kEDMA_MinorLink
Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.

Values:

enumerator kEDMA_DoneFlag
DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag
eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag
eDMA interrupt flag, set while an interrupt occurred of this channel

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag
Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_ChannelPriorityErrorFlag
Channel priority is not unique.

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

enumerator kEDMA_ValidFlag
No error occurred, this bit is 0. Otherwise, it is 1.

enum _edma_interrupt_enable
eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type
eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull
TCD queue is full.

enumerator kStatus_EDMA_Busy
Channel is busy and can't handle the transfer request.

typedef enum *_edma_transfer_size* edma_transfer_size_t
eDMA transfer configuration

typedef enum *_edma_modulo* edma_modulo_t
eDMA modulo configuration

typedef enum *_edma_bandwidth* edma_bandwidth_t
Bandwidth control.

typedef enum *_edma_channel_link_type* edma_channel_link_type_t
Channel link type.

typedef enum *_edma_interrupt_enable* edma_interrupt_enable_t
eDMA interrupt source

typedef enum *_edma_transfer_type* edma_transfer_type_t
eDMA transfer type

typedef struct *_edma_config* edma_config_t
eDMA global configuration structure.

typedef struct *_edma_transfer_config* edma_transfer_config_t
eDMA transfer configuration

This structure configures the source/destination transfer attribute.

typedef struct *_edma_channel_Preemption_config* edma_channel_Preemption_config_t
eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* edma_minor_offset_config_t
eDMA minor offset configuration

typedef struct *_edma_tcd* edma_tcd_t
eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

typedef void (*edma_callback)(struct *_edma_handle* *handle, void *userData, bool transferDone, uint32_t tcds)

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Param handle
EDMA handle pointer, users shall not touch the values inside.

Param userData
The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

Param transferDone
If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer

block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

Param tcds
How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

typedef struct *_edma_handle* edma_handle_t

eDMA transfer handle structure

DMA_DCHPRI_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct _edma_config

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

Public Members

bool enableContinuousLinkMode

Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

bool enableHaltOnError

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

bool enableRoundRobinArbitration

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection

bool enableDebugMode

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

struct _edma_transfer_config

#include <fsl_edma.h> eDMA transfer configuration

This structure configures the source/destination transfer attribute.

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

int16_t destOffset

Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

uint32_t minorLoopBytes

Bytes to transfer in a minor loop

uint32_t majorLoopCounts

Major loop iteration count.

struct _edma_channel_Preemption_config

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption

If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility

If true: a channel can suspend other channel with low priority

uint8_t channelPriority

Channel priority

struct _edma_minor_offset_config

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

bool enableSrcMinorOffset

Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset

Enable(true) or Disable(false) destination minor loop offset.

uint32_t minorOffset

Offset for a minor loop mapping.

struct _edma_tcd

#include <fsl_edma.h> eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

Public Members

___IO uint32_t SADDR

SADDR register, used to save source address

___IO uint16_t SOFF

SOFF register, save offset bytes every transfer

___IO uint16_t ATTR

ATTR register, source/destination transfer size and modulo

___IO uint32_t NBYTES

Nbytes register, minor loop length in bytes

___IO uint32_t SLAST
SLAST register

___IO uint32_t DADDR
DADDR register, used for destination address

___IO uint16_t DOFF
DOFF register, used for destination offset

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

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

___IO uint16_t CSR
CSR register, for TCD control status

___IO uint16_t BITER
BITER register, begin minor loop count.

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

Public Members

edma_callback callback
Callback function for major count exhausted.

void *userData
Callback function parameter.

DMA_Type *base
eDMA peripheral base address.

edma_tcd_t *tcdPool
Pointer to memory stored TCDs.

uint8_t channel
eDMA channel number.

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

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

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

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

uint8_t flags
The status of the current channel.

2.8 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const *ewm_config_t* *config)

Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

ewm_config_t

Parameters

- config – Pointer to the EWM configuration structure.

static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address

- mask – The interrupts to enable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to disable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

_ewm_status_flags_t

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)

Services the EWM.

This function resets the EWM counter to zero.

Parameters

- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION

EWM driver version 2.0.4.

enum _ewm_lpo_clock_source

Describes EWM clock source.

Values:

enumerator kEWM_LpoClockSource0

EWM clock sourced from lpo_clk[0]

enumerator kEWM_LpoClockSource1

EWM clock sourced from lpo_clk[1]

enumerator kEWM_LpoClockSource2

EWM clock sourced from lpo_clk[2]

enumerator kEWM_LpoClockSource3
EWM clock sourced from lpo_clk[3]

enum _ewm_interrupt_enable_t
EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable
Enable the EWM to generate an interrupt

enum _ewm_status_flags_t
EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator kEWM_RunningFlag
Running flag, set when EWM is enabled

typedef enum *_ewm_lpo_clock_source* ewm_lpo_clock_source_t
Describes EWM clock source.

typedef struct *_ewm_config* ewm_config_t
Data structure for EWM configuration.

This structure is used to configure the EWM.

struct _ewm_config
#include <fsl_ewm.h> Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

bool enableEwm
Enable EWM module

bool enableEwmInput
Enable EWM_in input

bool setInputAssertLogic
EWM_in signal assertion state

bool enableInterrupt
Enable EWM interrupt

ewm_lpo_clock_source_t clockSource
Clock source select

uint8_t prescaler
Clock prescaler value

uint8_t compareLowValue
Compare low-register value

uint8_t compareHighValue
Compare high-register value

2.9 FGPIO Driver

void FGPIO_PinInit(FGPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)

Initializes a FGPIO pin used by the board.

To initialize the FGPIO driver, define a pin configuration, as either input or output, in the user file. Then, call the FGPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration:

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalOutput,
  0,
}
```

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- pin – FGPIO port pin number
- config – FGPIO pin configuration pointer

static inline void FGPIO_PinWrite(FGPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple FGPIO pins to the logic 1 or 0.

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- pin – FGPIO pin number
- output – FGPIOpin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void FGPIO_PortSet(FGPIO_Type *base, uint32_t mask)

Sets the output level of the multiple FGPIO pins to the logic 1.

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- mask – FGPIO pin number macro

static inline void FGPIO_PortClear(FGPIO_Type *base, uint32_t mask)

Sets the output level of the multiple FGPIO pins to the logic 0.

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- mask – FGPIO pin number macro

static inline void FGPIO_PortToggle(FGPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple FGPIO pins.

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- mask – FGPIO pin number macro

static inline uint32_t FGPIO_PinRead(FGPIO_Type *base, uint32_t pin)

Reads the current input value of the FGPIO port.

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- pin – FGPIO pin number

Return values

FGPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

uint32_t FGPIO_PortGetInterruptFlags(FGPIO_Type *base)

Reads the FGPIO port interrupt status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level-sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)

Return values

The – current FGPIO port interrupt status flags, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

void FGPIO_PortClearInterruptFlags(FGPIO_Type *base, uint32_t mask)

Clears the multiple FGPIO pin interrupt status flag.

Parameters

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- mask – FGPIO pin number macro

2.10 FlexIO: FlexIO Driver

2.11 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

- userConfig – pointer to flexio_config_t structure

void FLEXIO_Init(FLEXIO_Type *base, const *flexio_config_t* *userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_GetDefaultConfig().

Example

```
flexio_config_t config = {
.enableFlexio = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio_config_t structure

void FLEXIO_Deinit(FLEXIO_Type *base)

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

Note: After calling this API, call the FLEXO_Init to use the FlexIO module.

Parameters

- base – FlexIO peripheral base address

uint32_t FLEXIO_GetInstance(FLEXIO_Type *base)

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

void FLEXIO_Reset(FLEXIO_Type *base)

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

static inline void FLEXIO_Enable(FLEXIO_Type *base, bool enable)

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns
FlexIO pin input data

static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns
current State pointer

void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const *flexio_shifter_config_t* *shifterConfig)

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
.timerSelect = 0,
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const *flexio_timer_config_t* *timerConfig)

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
```

(continues on next page)

(continued from previous page)

```
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index, *flexio_timer_decrement_source_t* clocksource)

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)

Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)

Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)

Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)
- enable – True to enable, false to disable.

uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, *flexio_shifter_buffer_type_t* type, uint8_t index)

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio_shifter_buffer_type_t
- index – Shifter index

Returns

Corresponding shifter buffer index

status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, *flexio_isr_t* isr)

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_UnregisterHandleIRQ(void *base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 0.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 1.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- output – FLEXIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)

Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)

Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL_FLEXIO_DRIVER_VERSION

FlexIO driver version.

enum _flexio_timer_trigger_polarity

Define time of timer trigger polarity.

Values:

enumerator kFLEXIO_TimerTriggerPolarityActiveHigh

Active high.

enumerator kFLEXIO_TimerTriggerPolarityActiveLow

Active low.

enum _flexio_timer_trigger_source

Define type of timer trigger source.

Values:

enumerator kFLEXIO_TimerTriggerSourceExternal

External trigger selected.

enumerator kFLEXIO_TimerTriggerSourceInternal
Internal trigger selected.

enum _flexio_pin_config
Define type of timer/shifter pin configuration.

Values:

enumerator kFLEXIO_PinConfigOutputDisabled
Pin output disabled.

enumerator kFLEXIO_PinConfigOpenDrainOrBidirection
Pin open drain or bidirectional output enable.

enumerator kFLEXIO_PinConfigBidirectionOutputData
Pin bidirectional output data.

enumerator kFLEXIO_PinConfigOutput
Pin output.

enum _flexio_pin_polarity
Definition of pin polarity.

Values:

enumerator kFLEXIO_PinActiveHigh
Active high.

enumerator kFLEXIO_PinActiveLow
Active low.

enum _flexio_timer_mode
Define type of timer work mode.

Values:

enumerator kFLEXIO_TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO_TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO_TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO_TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO_TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum _flexio_timer_output
Define type of timer initial output or timer reset condition.

Values:

enumerator kFLEXIO_TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO_TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum _flexio_timer_decrement_source
Define type of timer decrement.

Values:

enumerator kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput
Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum _flexio_timer_reset_condition
Define type of timer reset condition.

Values:

enumerator kFLEXIO_TimerResetNever
Timer never reset.

enumerator kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput
Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput
Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerPinRisingEdge
Timer reset on Timer Pin rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerRisingEdge
Timer reset on Trigger rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerBothEdge
Timer reset on Trigger rising or falling edge.

enum _flexio_timer_disable_condition
Define type of timer disable condition.

Values:

enumerator kFLEXIO_TimerDisableNever
Timer never disabled.

enumerator kFLEXIO_TimerDisableOnPreTimerDisable
Timer disabled on Timer N-1 disable.

enumerator kFLEXIO_TimerDisableOnTimerCompare
Timer disabled on Timer compare.

enumerator kFLEXIO_TimerDisableOnTimerCompareTriggerLow
Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO_TimerDisableOnPinBothEdge
Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh
Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO_TimerDisableOnTriggerFallingEdge
Timer disabled on Trigger falling edge.

enum _flexio_timer_enable_condition
Define type of timer enable condition.

Values:

enumerator kFLEXIO_TimerEnabledAlways
Timer always enabled.

enumerator kFLEXIO_TimerEnableOnPrevTimerEnable
Timer enabled on Timer N-1 enable.

enumerator kFLEXIO_TimerEnableOnTriggerHigh
Timer enabled on Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerHighPinHigh
Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO_TimerEnableOnPinRisingEdge
Timer enabled on Pin rising edge.

enumerator kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh
Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerRisingEdge
Timer enabled on Trigger rising edge.

enumerator kFLEXIO_TimerEnableOnTriggerBothEdge
Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition
Define type of timer stop bit generate condition.

Values:

enumerator kFLEXIO_TimerStopBitDisabled
Stop bit disabled.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompare
Stop bit is enabled on timer compare.

enumerator kFLEXIO_TimerStopBitEnableOnTimerDisable
Stop bit is enabled on timer disable.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompareDisable
Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition
Define type of timer start bit generate condition.

Values:

enumerator kFLEXIO_TimerStartBitDisabled
Start bit disabled.

enumerator kFLEXIO_TimerStartBitEnabled
Start bit enabled.

enum _flexio_timer_output_state

FlexIO as PWM channel output state.

Values:

enumerator kFLEXIO_PwmLow

The output state of PWM channel is low

enumerator kFLEXIO_PwmHigh

The output state of PWM channel is high

enum _flexio_shifter_timer_polarity

Define type of timer polarity for shifter control.

Values:

enumerator kFLEXIO_ShifterTimerPolarityOnPositive

Shift on positive edge of shift clock.

enumerator kFLEXIO_ShifterTimerPolarityOnNegitive

Shift on negative edge of shift clock.

enum _flexio_shifter_mode

Define type of shifter working mode.

Values:

enumerator kFLEXIO_ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO_ShifterModeReceive

Receive mode.

enumerator kFLEXIO_ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO_ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO_ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO_ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO_ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio_shifter_input_source

Define type of shifter input source.

Values:

enumerator kFLEXIO_ShifterInputFromPin

Shifter input from pin.

enumerator kFLEXIO_ShifterInputFromNextShifterOutput

Shifter input from Shifter N+1.

enum _flexio_shifter_stop_bit

Define of STOP bit configuration.

Values:

enumerator kFLEXIO_ShifterStopBitDisable
Disable shifter stop bit.

enumerator kFLEXIO_ShifterStopBitLow
Set shifter stop bit to logic low level.

enumerator kFLEXIO_ShifterStopBitHigh
Set shifter stop bit to logic high level.

enum _flexio_shifter_start_bit
Define type of START bit configuration.

Values:

enumerator kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable
Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO_ShifterStartBitDisabledLoadDataOnShift
Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO_ShifterStartBitLow
Set shifter start bit to logic low level.

enumerator kFLEXIO_ShifterStartBitHigh
Set shifter start bit to logic high level.

enum _flexio_shifter_buffer_type
Define FlexIO shifter buffer type.

Values:

enumerator kFLEXIO_ShifterBuffer
Shifter Buffer N Register.

enumerator kFLEXIO_ShifterBufferBitSwapped
Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferByteSwapped
Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferBitByteSwapped
Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO_ShifterBufferNibbleByteSwapped
Shifter Buffer N Nibble Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferHalfWordSwapped
Shifter Buffer N Half Word Swapped Register.

enumerator kFLEXIO_ShifterBufferNibbleSwapped
Shifter Buffer N Nibble Swapped Register.

enum _flexio_gpio_direction
FLEXIO gpio direction definition.

Values:

enumerator kFLEXIO_DigitalInput
Set current pin as digital input

enumerator kFLEXIO_DigitalOutput
Set current pin as digital output

enum _flexio_pin_input_config

FLEXIO gpio input config.

Values:

enumerator kFLEXIO_InputInterruptDisabled

Interrupt request is disabled.

enumerator kFLEXIO_InputInterruptEnable

Interrupt request is enable.

enumerator kFLEXIO_FlagRisingEdgeEnable

Input pin flag on rising edge.

enumerator kFLEXIO_FlagFallingEdgeEnable

Input pin flag on falling edge.

typedef enum *_flexio_timer_trigger_polarity* flexio_timer_trigger_polarity_t

Define time of timer trigger polarity.

typedef enum *_flexio_timer_trigger_source* flexio_timer_trigger_source_t

Define type of timer trigger source.

typedef enum *_flexio_pin_config* flexio_pin_config_t

Define type of timer/shifter pin configuration.

typedef enum *_flexio_pin_polarity* flexio_pin_polarity_t

Definition of pin polarity.

typedef enum *_flexio_timer_mode* flexio_timer_mode_t

Define type of timer work mode.

typedef enum *_flexio_timer_output* flexio_timer_output_t

Define type of timer initial output or timer reset condition.

typedef enum *_flexio_timer_decrement_source* flexio_timer_decrement_source_t

Define type of timer decrement.

typedef enum *_flexio_timer_reset_condition* flexio_timer_reset_condition_t

Define type of timer reset condition.

typedef enum *_flexio_timer_disable_condition* flexio_timer_disable_condition_t

Define type of timer disable condition.

typedef enum *_flexio_timer_enable_condition* flexio_timer_enable_condition_t

Define type of timer enable condition.

typedef enum *_flexio_timer_stop_bit_condition* flexio_timer_stop_bit_condition_t

Define type of timer stop bit generate condition.

typedef enum *_flexio_timer_start_bit_condition* flexio_timer_start_bit_condition_t

Define type of timer start bit generate condition.

typedef enum *_flexio_timer_output_state* flexio_timer_output_state_t

FlexIO as PWM channel output state.

typedef enum *_flexio_shifter_timer_polarity* flexio_shifter_timer_polarity_t

Define type of timer polarity for shifter control.

typedef enum *_flexio_shifter_mode* flexio_shifter_mode_t

Define type of shifter working mode.

typedef enum *_flexio_shifter_input_source* flexio_shifter_input_source_t
Define type of shifter input source.

typedef enum *_flexio_shifter_stop_bit* flexio_shifter_stop_bit_t
Define of STOP bit configuration.

typedef enum *_flexio_shifter_start_bit* flexio_shifter_start_bit_t
Define type of START bit configuration.

typedef enum *_flexio_shifter_buffer_type* flexio_shifter_buffer_type_t
Define FlexIO shifter buffer type.

typedef struct *_flexio_config_* flexio_config_t
Define FlexIO user configuration structure.

typedef struct *_flexio_timer_config* flexio_timer_config_t
Define FlexIO timer configuration structure.

typedef struct *_flexio_shifter_config* flexio_shifter_config_t
Define FlexIO shifter configuration structure.

typedef enum *_flexio_gpio_direction* flexio_gpio_direction_t
FLEXIO gpio direction definition.

typedef enum *_flexio_pin_input_config* flexio_pin_input_config_t
FLEXIO gpio input config.

typedef struct *_flexio_gpio_config* flexio_gpio_config_t
The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

typedef void (*flexio_isr_t)(void *base, void *handle)
typedef for FlexIO simulated driver interrupt handler.

FLEXIO_Type *const s_flexioBases[]
Pointers to flexio bases for each instance.

const *clock_ip_name_t* s_flexioClocks[]
Pointers to flexio clocks for each instance.

void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, *flexio_gpio_config_t* *config)
Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the FLEXIO_SetPinConfig() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
  kFLEXIO_DigitalInput,
  0U,
  kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
  kFLEXIO_DigitalOutput,
  0U,
```

(continues on next page)

(continued from previous page)

```
 0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)

Calculate FlexIO timer trigger.

FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(x)

FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)

struct _flexio_config_

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members

bool enableFlexio

Enable/disable FlexIO module

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

struct _flexio_timer_config

#include <fsl_flexio.h> Define FlexIO timer configuration structure.

Public Members

uint32_t triggerSelect

The internal trigger selection number using MACROs.

flexio_timer_trigger_polarity_t triggerPolarity

Trigger Polarity.

flexio_timer_trigger_source_t triggerSource

Trigger Source, internal (see 'trgsel') or external.

flexio_pin_config_t pinConfig

Timer Pin Configuration.

uint32_t pinSelect

Timer Pin number Select.

flexio_pin_polarity_t pinPolarity

Timer Pin Polarity.

flexio_timer_mode_t timerMode
Timer work Mode.

flexio_timer_output_t timerOutput
Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

flexio_timer_decrement_source_t timerDecrement
Configures the source of the Timer decrement and the source of the Shift clock.

flexio_timer_reset_condition_t timerReset
Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

flexio_timer_disable_condition_t timerDisable
Configures the condition that causes the Timer to be disabled and stop decrementing.

flexio_timer_enable_condition_t timerEnable
Configures the condition that causes the Timer to be enabled and start decrementing.

flexio_timer_stop_bit_condition_t timerStop
Timer STOP Bit generation.

flexio_timer_start_bit_condition_t timerStart
Timer STRAT Bit generation.

uint32_t timerCompare
Value for Timer Compare N Register.

struct _flexio_shifter_config
#include <fsl_flexio.h> Define FlexIO shifter configuration structure.

Public Members

uint32_t timerSelect
Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

flexio_shifter_timer_polarity_t timerPolarity
Timer Polarity.

flexio_pin_config_t pinConfig
Shifter Pin Configuration.

uint32_t pinSelect
Shifter Pin number Select.

flexio_pin_polarity_t pinPolarity
Shifter Pin Polarity.

flexio_shifter_mode_t shifterMode
Configures the mode of the Shifter.

uint32_t parallelWidth
Configures the parallel width when using parallel mode.

flexio_shifter_input_source_t inputSource
Selects the input source for the shifter.

flexio_shifter_stop_bit_t shifterStop
Shifter STOP bit.

flexio_shifter_start_bit_t shifterStart

Shifter START bit.

struct _flexio_gpio_config

#include <fsl_flexio.h> The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

flexio_gpio_direction_t pinDirection

FLEXIO pin direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

uint8_t inputConfig

Set an input config

2.12 FlexIO eDMA I2S Driver

void FLEXIO_I2S_TransferTxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferRxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.

- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferSetFormatEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

status_t FLEXIO_I2S_TransferSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S transfer using DMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA send successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_TxBusy – FlexIO I2S is busy sending data.

status_t FLEXIO_I2S_TransferReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S receive using eDMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA receive successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_RxBusy – FlexIO I2S is busy receiving data.

void FLEXIO_I2S_TransferAbortSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S transfer using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

void FLEXIO_I2S_TransferAbortReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S receive using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

status_t FLEXIO_I2S_TransferGetSendCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Gets the remaining bytes to be sent.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

Return values

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

status_t FLEXIO_I2S_TransferGetReceiveCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Get the remaining bytes to be received.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes received.

Return values

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

FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION

FlexIO I2S EDMA driver version 2.1.9.

typedef struct *_flexio_i2s_edma_handle* flexio_i2s_edma_handle_t

typedef void (*flexio_i2s_edma_callback_t)(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2S eDMA transfer callback function for finish and error.

struct _flexio_i2s_edma_handle

#include <fsl_flexio_i2s_edma.h> FlexIO I2S DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for FlexIO I2S send

uint8_t bytesPerFrame

Bytes in a frame

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint32_t state

Internal state for FlexIO I2S eDMA transfer

flexio_i2s_edma_callback_t callback

Callback for users while transfer finish or error occurred

void *userData

User callback parameter

edma_tcd_t tcd[(4U) + 1U]

TCD pool for eDMA transfer.

flexio_i2s_transfer_t queue[(4U)]

Transfer queue storing queued transfer.

size_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.13 FlexIO eDMA SPI Driver

status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txHandle, *edma_handle_t* *rxHandle)

Initializes the FlexIO SPI master eDMA handle.

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_SPI_MasterTransferEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

status_t FLEXIO_SPI_MasterTransferGetCountEDMA(*FLEXIO_SPI_Type* *base,
flexio_spi_master_edma_handle_t *handle,
size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_edma_handle_t *handle,
flexio_spi_slave_edma_transfer_callback_t callback, void *userData,
edma_handle_t *txHandle,
edma_handle_t *rxHandle)

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

status_t FLEXIO_SPI_SlaveTransferEDMA(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_edma_handle_t *handle,
flexio_spi_transfer_t *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCountEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION

FlexIO SPI EDMA driver version.

typedef struct *_flexio_spi_master_edma_handle* flexio_spi_master_edma_handle_t

typedef for flexio_spi_master_edma_handle_t in advance.

typedef *flexio_spi_master_edma_handle_t* flexio_spi_slave_edma_handle_t

Slave handle is the same with master handle.

typedef void (*flexio_spi_master_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI slave callback for finished transmit.

struct _flexio_spi_master_edma_handle

#include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch the content of the handle.

Public Members

size_t transferSize

Total bytes to be transferred.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

bool txInProgress

Send transfer in progress

bool rxInProgress

Receive transfer in progress

edma_handle_t *txHandle

DMA handler for SPI send

edma_handle_t *rxHandle
DMA handler for SPI receive

flexio_spi_master_edma_transfer_callback_t callback
Callback for SPI DMA transfer

void *userData
User Data for SPI DMA callback

2.14 FlexIO eDMA UART Driver

status_t FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_UART_Type.
- handle – Pointer to flexio_uart_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

status_t FLEXIO_UART_TransferReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

void FLEXIO_UART_TransferAbortSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

void FLEXIO_UART_TransferAbortReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

status_t FLEXIO_UART_TransferGetSendCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferGetReceiveCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

FSL_FLEXIO_UART_EDMA_DRIVER_VERSION

FlexIO UART EDMA driver version.

typedef struct *_flexio_uart_edma_handle* flexio_uart_edma_handle_t

typedef void (*flexio_uart_edma_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct flexio_uart_edma_handle

#include <fsl_flexio_uart_edma.h> UART eDMA handle.

Public Members

flexio_uart_edma_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

size_t txDataSizeAll

Total bytes to be sent.

size_t rxDataSizeAll

Total bytes to be received.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.15 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND _flexio_i2c_master_status_flags can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_I2C_RxFullFlag
  - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ

- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t* direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
  - kFLEXIO_I2C_Write: Transmit
  - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the

transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

status_t FLEXIO_I2C_MasterTransferGetCount(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_I2C_MasterTransferAbort(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy
I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle
I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak
NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout
Timeout polling status flags.

enum _flexio_i2c_master_interrupt
Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable
Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable
Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags
Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag
Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag
Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag
Receive NAK flag.

enum _flexio_i2c_direction
Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write
Master send to slave.

enumerator kFLEXIO_I2C_Read
Master receive from slave.

typedef enum *_flexio_i2c_direction* flexio_i2c_direction_t
Direction of master transfer.

typedef struct *_flexio_i2c_type* FLEXIO_I2C_Type
Define FlexIO I2C master access structure typedef.

typedef struct *_flexio_i2c_master_config* flexio_i2c_master_config_t
Define FlexIO I2C master user configuration structure.

typedef struct *_flexio_i2c_master_transfer* flexio_i2c_master_transfer_t

Define FlexIO I2C master transfer structure.

typedef struct *_flexio_i2c_master_handle* flexio_i2c_master_handle_t

FlexIO I2C master handle typedef.

typedef void (*flexio_i2c_master_transfer_callback_t)(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2C master transfer callback typedef.

I2C_RETRY_TIMES

Retry times for waiting flag.

struct _flexio_i2c_type

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

uint8_t SDAPinIndex

Pin select for I2C SDA.

uint8_t SCLPinIndex

Pin select for I2C SCL.

uint8_t shifterIndex[2]

Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]

Timer index used in FlexIO I2C.

uint32_t baudrate

Master transfer baudrate, used to calculate delay time.

struct _flexio_i2c_master_config

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster

Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

struct _flexio_i2c_master_transfer

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags
Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress
7-bit slave address.

flexio_i2c_direction_t direction
Transfer direction, read or write.

uint32_t subaddress
Sub address. Transferred MSB first.

uint8_t subaddressSize
Size of sub address.

uint8_t volatile *data
Transfer buffer.

volatile size_t dataSize
Transfer size.

struct __flexio_i2c_master_handle
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer
FlexIO I2C master transfer copy.

size_t transferSize
Total bytes to be transferred.

uint8_t state
Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback
Callback function called at transfer event. Callback function called at transfer event.

void *userData
Callback parameter passed to callback function.

bool needRestart
Whether master needs to send re-start signal.

2.16 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)
Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO_I2S_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in FLEXIO_I2S_Init(). Users may use the initialized structure unchanged in FLEXIO_I2S_Init() or modify some fields of the structure before calling FLEXIO_I2S_Init().

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXO_I2S_Init to use the FlexIO I2S module.

Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO_I2S_Enable(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S module operation.

Parameters

- base – Pointer to FLEXIO_I2S_Type
- enable – True to enable, false dose not have any effect.

uint32_t FLEXIO_I2S_GetStatusFlags(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S status flags.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

Status flag, which are ORed by the enumerators in the _flexio_i2s_status_flags.

void FLEXIO_I2S_EnableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

void FLEXIO_I2S_DisableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

static inline void FLEXIO_I2S_TxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S Tx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO_I2S_RxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S Rx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

FlexIO i2s send data register address.

static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

FlexIO i2s receive data register address.

void FLEXIO_I2S_MasterSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – I2S master clock source frequency in Hz.

void FLEXIO_I2S_SlaveSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format)

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- format – Pointer to FlexIO I2S audio data format structure.

status_t FLEXIO_I2S_WriteBlocking(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint8_t *txData, size_t size)

Sends data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

static inline void FLEXIO_I2S_WriteData(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint32_t data)

Writes data into a data register.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- data – Data to be written.

status_t FLEXIO_I2S_ReadBlocking(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint8_t *rxData, size_t size)

Receives a piece of data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- rxData – Pointer to the data to be read.
- size – Bytes to be read.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

static inline uint32_t FLEXIO_I2S_ReadData(*FLEXIO_I2S_Type* *base)

Reads a data from the data register.

Parameters

- base – FlexIO I2S base pointer

Returns

Data read from data register.

void FLEXIO_I2S_TransferTxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- handle – Pointer to flexio_i2s_handle_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

void FLEXIO_I2S_TransferSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

void FLEXIO_I2S_TransferRxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

status_t FLEXIO_I2S_TransferSendNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXIO_I2S_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t FLEXIO_I2S_TransferReceiveNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data receive.
- kStatus_FLEXIO_I2S_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

void FLEXIO_I2S_TransferAbortSend(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current send.

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

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

void FLEXIO_I2S_TransferAbortReceive(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current receive.

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

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

status_t FLEXIO_I2S_TransferGetSendCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, size_t *count)

Gets the remaining bytes to be sent.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes sent.

Return values

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

status_t FLEXIO_I2S_TransferGetReceiveCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, size_t *count)

Gets the remaining bytes to be received.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes recieved.

Return values

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

Returns

count Bytes received.

void FLEXIO_I2S_TransferTxHandleIRQ(void *i2sBase, void *i2sHandle)

Tx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure

void FLEXIO_I2S_TransferRxHandleIRQ(void *i2sBase, void *i2sHandle)

Rx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure.

FSL_FLEXIO_I2S_DRIVER_VERSION

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator kStatus_FLEXIO_I2S_Idle
FlexIO I2S is in idle state

enumerator kStatus_FLEXIO_I2S_TxBusy
FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_RxBusy
FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_Error
FlexIO I2S error occurred

enumerator kStatus_FLEXIO_I2S_QueueFull
FlexIO I2S transfer queue is full.

enumerator kStatus_FLEXIO_I2S_Timeout
FlexIO I2S timeout polling status flags.

enum _flexio_i2s_master_slave
Master or slave mode.

Values:

enumerator kFLEXIO_I2S_Master
Master mode

enumerator kFLEXIO_I2S_Slave
Slave mode

_flexio_i2s_interrupt_enable Define FlexIO FlexIO I2S interrupt mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_I2S_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

_flexio_i2s_status_flags Define FlexIO FlexIO I2S status mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_I2S_RxDataRegFullFlag
Receive buffer full flag.

enum _flexio_i2s_sample_rate
Audio sample rate.

Values:

enumerator kFLEXIO_I2S_SampleRate8KHz
Sample rate 8000Hz

enumerator kFLEXIO_I2S_SampleRate11025Hz
Sample rate 11025Hz

enumerator kFLEXIO_I2S_SampleRate12KHz
Sample rate 12000Hz

enumerator kFLEXIO_I2S_SampleRate16KHz
Sample rate 16000Hz

enumerator kFLEXIO_I2S_SampleRate22050Hz
Sample rate 22050Hz

enumerator kFLEXIO_I2S_SampleRate24KHz
Sample rate 24000Hz

enumerator kFLEXIO_I2S_SampleRate32KHz
Sample rate 32000Hz

enumerator kFLEXIO_I2S_SampleRate44100Hz
Sample rate 44100Hz

enumerator kFLEXIO_I2S_SampleRate48KHz
Sample rate 48000Hz

enumerator kFLEXIO_I2S_SampleRate96KHz
Sample rate 96000Hz

enum _flexio_i2s_word_width
Audio word width.

Values:

enumerator kFLEXIO_I2S_WordWidth8bits
Audio data width 8 bits

enumerator kFLEXIO_I2S_WordWidth16bits
Audio data width 16 bits

enumerator kFLEXIO_I2S_WordWidth24bits
Audio data width 24 bits

enumerator kFLEXIO_I2S_WordWidth32bits
Audio data width 32 bits

typedef struct *_flexio_i2s_type* FLEXIO_I2S_Type
Define FlexIO I2S access structure typedef.

typedef enum *_flexio_i2s_master_slave* flexio_i2s_master_slave_t
Master or slave mode.

typedef struct *_flexio_i2s_config* flexio_i2s_config_t
FlexIO I2S configure structure.

typedef struct *_flexio_i2s_format* flexio_i2s_format_t
FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

typedef enum *_flexio_i2s_sample_rate* flexio_i2s_sample_rate_t
Audio sample rate.

typedef enum *_flexio_i2s_word_width* flexio_i2s_word_width_t
Audio word width.

typedef struct *_flexio_i2s_transfer* flexio_i2s_transfer_t

Define FlexIO I2S transfer structure.

typedef struct *_flexio_i2s_handle* flexio_i2s_handle_t

typedef void (*flexio_i2s_callback_t)(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2S xfer callback prototype.

I2S_RETRY_TIMES

Retry times for waiting flag.

FLEXIO_I2S_XFER_QUEUE_SIZE

FlexIO I2S transfer queue size, user can refine it according to use case.

struct flexio_i2s_type

#include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer

uint8_t txPinIndex

Tx data pin index in FlexIO pins

uint8_t rxPinIndex

Rx data pin index

uint8_t bclkPinIndex

Bit clock pin index

uint8_t fsPinIndex

Frame sync pin index

uint8_t txShifterIndex

Tx data shifter index

uint8_t rxShifterIndex

Rx data shifter index

uint8_t bclkTimerIndex

Bit clock timer index

uint8_t fsTimerIndex

Frame sync timer index

struct flexio_i2s_config

#include <fsl_flexio_i2s.h> FlexIO I2S configure structure.

Public Members

bool enableI2S

Enable FlexIO I2S

flexio_i2s_master_slave_t masterSlave

Master or slave

flexio_pin_polarity_t txPinPolarity

Tx data pin polarity, active high or low

flexio_pin_polarity_t rxPinPolarity
Rx data pin polarity

flexio_pin_polarity_t bclkPinPolarity
Bit clock pin polarity

flexio_pin_polarity_t fsPinPolarity
Frame sync pin polarity

flexio_shifter_timer_polarity_t txTimerPolarity
Tx data valid on bclk rising or falling edge

flexio_shifter_timer_polarity_t rxTimerPolarity
Rx data valid on bclk rising or falling edge

struct __flexio_i2s_format
#include <fsl_flexio_i2s.h> FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

Public Members

uint8_t bitWidth
Bit width of audio data, always 8/16/24/32 bits

uint32_t sampleRate_Hz
Sample rate of the audio data

struct __flexio_i2s_transfer
#include <fsl_flexio_i2s.h> Define FlexIO I2S transfer structure.

Public Members

uint8_t *data
Data buffer start pointer

size_t dataSize
Bytes to be transferred.

struct __flexio_i2s_handle
#include <fsl_flexio_i2s.h> Define FlexIO I2S handle structure.

Public Members

uint32_t state
Internal state

flexio_i2s_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32bits

flexio_i2s_transfer_t queue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.17 FlexIO SPI Driver

void FLEXIO_SPI_MasterInit(*FLEXIO_SPI_Type* *base, *flexio_spi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 3*2=6. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – status flag The parameter can be any combination of the following values:
  - kFLEXIO_SPI_TxEmptyFlag
  - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_DisableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_EnableDMA(*FLEXIO_SPI_Type* *base, uint32_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

static inline void FLEXIO_SPI_Enable(*FLEXIO_SPI_Type* *base, bool enable)

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.
- enable – True to enable, false does not have any effect.

void FLEXIO_SPI_MasterSetBaudRate(*FLEXIO_SPI_Type* *base, uint32_t baudRate_Bps, uint32_t srcClockHz)

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

static inline void FLEXIO_SPI_WriteData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint32_t data)

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

static inline uint32_t FLEXIO_SPI_ReadData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

status_t FLEXIO_SPI_WriteBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, const uint8_t *buffer, size_t size)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint8_t *buffer, size_t size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle)

Aborts the master data transfer, which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_SlaveTransferCreateHandle(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle,
flexio_spi_slave_transfer_callback_t callback,
void *userData)

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_SlaveTransferNonBlocking(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle,
flexio_spi_transfer_t *xfer)

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbort(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle)

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCount(*FLEXIO_SPI_Type* *base,
flexio_spi_slave_handle_t *handle,
size_t *count)

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI slave IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

FSL_FLEXIO_SPI_DRIVER_VERSION

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator kStatus_FLEXIO_SPI_Busy

FlexIO SPI is busy.

enumerator kStatus_FLEXIO_SPI_Idle

SPI is idle

enumerator kStatus_FLEXIO_SPI_Error

FlexIO SPI error.

enumerator kStatus_FLEXIO_SPI_Timeout

FlexIO SPI timeout polling status flags.

enum _flexio_spi_clock_phase

FlexIO SPI clock phase configuration.

Values:

enumerator kFLEXIO_SPI_ClockPhaseFirstEdge

First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge

First edge on SPSCK occurs at the start of the first cycle of a data transfer.

enum _flexio_spi_shift_direction

FlexIO SPI data shifter direction options.

Values:

enumerator kFLEXIO_SPI_MsbFirst

Data transfers start with most significant bit.

enumerator kFLEXIO_SPI_LsbFirst

Data transfers start with least significant bit.

enum _flexio_spi_data_bitcount_mode

FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode

8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags
Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag
Receive buffer full flag.

enum _flexio_spi_dma_enable
Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable
Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable
Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable
All DMA request source

enum _flexio_spi_transfer_flags
Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb
FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb
FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb
FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb
FlexIO SPI 16-bit LSB first

enumerator kFLEXIO_SPI_32bitMsb
FlexIO SPI 32-bit MSB first

enumerator kFLEXIO_SPI_32bitLsb
FlexIO SPI 32-bit LSB first

enumerator kFLEXIO_SPI_csContinuous
Enable the CS signal continuous mode

typedef enum *_flexio_spi_clock_phase* flexio_spi_clock_phase_t
FlexIO SPI clock phase configuration.

typedef enum *_flexio_spi_shift_direction* flexio_spi_shift_direction_t
FlexIO SPI data shifter direction options.

typedef enum *_flexio_spi_data_bitcount_mode* flexio_spi_data_bitcount_mode_t
FlexIO SPI data length mode options.

typedef struct *_flexio_spi_type* FLEXIO_SPI_Type
Define FlexIO SPI access structure typedef.

typedef struct *_flexio_spi_master_config* flexio_spi_master_config_t
Define FlexIO SPI master configuration structure.

typedef struct *_flexio_spi_slave_config* flexio_spi_slave_config_t
Define FlexIO SPI slave configuration structure.

typedef struct *_flexio_spi_transfer* flexio_spi_transfer_t
Define FlexIO SPI transfer structure.

typedef struct *_flexio_spi_master_handle* flexio_spi_master_handle_t
typedef for flexio_spi_master_handle_t in advance.

typedef *flexio_spi_master_handle_t* flexio_spi_slave_handle_t
Slave handle is the same with master handle.

typedef void (*flexio_spi_master_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *status_t* status, void *userData)
FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *status_t* status, void *userData)
FlexIO SPI slave callback for finished transmit.

FLEXIO_SPI_DUMMYDATA
FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

SPI_RETRY_TIMES
Retry times for waiting flag.

FLEXIO_SPI_XFER_DATA_FORMAT(flag)
Get the transfer data format of width and bit order.

struct flexio_spi_type
#include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t SDOPinIndex
: Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex
: Pin select for data input.

uint8_t SCKPinIndex
: Pin select for clock.

uint8_t CSnPinIndex
: Pin select for enable.

uint8_t shifterIndex[2]
: Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]
: Timer index used in FlexIO SPI.

struct __flexio_spi_master_config
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster
: Enable/disable FlexIO SPI master after configuration.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode.

bool enableInDebug
: Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

flexio_spi_clock_phase_t phase
: Clock phase.

flexio_spi_data_bitcount_mode_t dataMode
: 8bit or 16bit mode.

struct __flexio_spi_slave_config
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave
: Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode.

bool enableInDebug
: Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct __flexio_spi_transfer

#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

size_t dataSize

Transfer bytes.

uint8_t flags

FlexIO SPI control flag, MSB first or LSB first.

struct __flexio_spi_master_handle

#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData

Transfer buffer.

uint8_t *rxData

Receive buffer.

size_t transferSize

Total bytes to be transferred.

volatile size_t txRemainingBytes

Send data remaining in bytes.

volatile size_t rxRemainingBytes

Receive data remaining in bytes.

volatile uint32_t state

FlexIO SPI internal state.

uint8_t bytePerFrame

SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction

Shift direction.

flexio_spi_master_transfer_callback_t callback

FlexIO SPI callback.

void *userData

Callback parameter.

bool isCsContinuous

Is current transfer using CS continuous mode.

uint32_t timer1Cfg

TIMER1 TIMCFG regiser value backup.

2.18 FlexIO UART Driver

status_t FLEXIO_UART_Init(*FLEXIO_UART_Type* *base, const *flexio_uart_config_t* *userConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_GetDefaultConfig().

Example

```
FLEXIO_UART_Type base = {
.flexioBase = FLEXIO,
.TxPinIndex = 0,
.RxPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_uart_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_UART_TxDataRegEmptyFlag
  - kFLEXIO_UART_RxEmptyFlag
  - kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UARt transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns
FlexIO UART transmit data register address.

static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns
FlexIO UART receive data register address.

static inline void FLEXIO_UART_EnableTxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_EnableRxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_Enable(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

static inline void FLEXIO_UART_WriteByte(*FLEXIO_UART_Type* *base, const uint8_t *buffer)

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

static inline void FLEXIO_UART_ReadByte(*FLEXIO_UART_Type* *base, uint8_t *buffer)

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.

- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

void FLEXIO_UART_TransferStartRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void FLEXIO_UART_TransferStopRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferSendNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See flexio_uart_transfer_t.

Return values

- kStatus_Success – Successfully starts the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished, data not written to the TX register.

void FLEXIO_UART_TransferAbortSend(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetSendCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferReceiveNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to xfer->data. This function returns with the parameter receivedBytes set to 5. For the last 5 bytes, newly arrived data is saved from the

xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See flexio_uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

void FLEXIO_UART_TransferAbortReceive(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetReceiveCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy

Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy

Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle

UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR

ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun

UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout

UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport

Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char

FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar

7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar

8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar

9-bit data characters

enum _flexio_uart_interrupt_enable

Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
Define FlexIO UART status mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag
Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag
Receive buffer over run flag.

typedef enum *_flexio_uart_bit_count_per_char* flexio_uart_bit_count_per_char_t
FlexIO UART bit count per char.

typedef struct *_flexio_uart_type* FLEXIO_UART_Type
Define FlexIO UART access structure typedef.

typedef struct *_flexio_uart_config* flexio_uart_config_t
Define FlexIO UART user configuration structure.

typedef struct *_flexio_uart_transfer* flexio_uart_transfer_t
Define FlexIO UART transfer structure.

typedef struct *_flexio_uart_handle* flexio_uart_handle_t

typedef void (*flexio_uart_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *status_t* status, void *userData)
FlexIO UART transfer callback function.

UART_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_uart_type
#include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t TxPinIndex
Pin select for UART_Tx.

uint8_t RxPinIndex
Pin select for UART_Rx.

uint8_t shifterIndex[2]
Shifter index used in FlexIO UART.

uint8_t timerIndex[2]
Timer index used in FlexIO UART.

struct _flexio_uart_config
#include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

Public Members

bool enableUart
: Enable/disable FlexIO UART TX & RX.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode

bool enableInDebug
: Enable/disable FlexIO operation in debug mode

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

flexio_uart_bit_count_per_char_t bitCountPerChar
: number of bits, 7/8/9 -bit

struct __flexio_uart_transfer
: *#include <fsl_flexio_uart.h>* Define FlexIO UART transfer structure.

Public Members

size_t dataSize
: Transfer size

struct __flexio_uart_handle
: *#include <fsl_flexio_uart.h>* Define FLEXIO UART handle structure.

Public Members

const uint8_t *volatile txData
: Address of remaining data to send.

volatile size_t txDataSize
: Size of the remaining data to send.

uint8_t *volatile rxData
: Address of remaining data to receive.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t txDataSizeAll
: Total bytes to be sent.

size_t rxDataSizeAll
: Total bytes to be received.

uint8_t *rxRingBuffer
: Start address of the receiver ring buffer.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
Callback function.

void *userData
UART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

union ___unnamed38___

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.

2.19 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION
GPIO driver version.

enum _gpio_pin_direction
GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput
Set current pin as digital input

enumerator kGPIO_DigitalOutput
Set current pin as digital output

enum _gpio_checker_attribute
GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW
User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW
User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW
User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO_IgnoreAttributeCheck
Ignores the attribute check

enum _gpio_interrupt_selection
Configures the selection of interrupt/DMA request/trigger output.

Values:

enumerator kGPIO_InterruptOutput0
Interrupt/DMA request/trigger output 0.

enumerator kGPIO_InterruptOutput1
Interrupt/DMA request/trigger output 1.

enum gpio_pin_interrupt_control_t
GPIO pin and interrupt control.

Values:

enumerator kGPIO_PinControlNonSecure
Pin Control Non-Secure.

enumerator kGPIO_InterruptControlNonSecure
Interrupt Control Non-Secure.

enumerator kGPIO_PinControlNonPrivilege
Pin Control Non-Privilege.

enumerator kGPIO_InterruptControlNonPrivilege
Interrupt Control Non-Privilege.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_checker_attribute* gpio_checker_attribute_t
GPIO checker attribute.

typedef struct *_gpio_pin_config* gpio_pin_config_t
The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

typedef enum *_gpio_interrupt_selection* gpio_interrupt_selection_t
Configures the selection of interrupt/DMA request/trigger output.

typedef struct *_gpio_version_info* gpio_version_info_t
GPIO version information.

GPIO_FIT_REG(value)

struct _gpio_pin_config

#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Public Members

gpio_pin_direction_t pinDirection

GPIO direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

struct _gpio_version_info

#include <fsl_gpio.h> GPIO version information.

Public Members

uint16_t feature

Feature Specification Number.

uint8_t minor

Minor Version Number.

uint8_t major

Major Version Number.

2.20 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PortDenit(GPIO_Type *base)

Denitializes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalOutput,
  0,
}
```

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

void GPIO_GetVersionInfo(GPIO_Type *base, *gpio_version_info_t* *verInfo)

Get GPIO version information.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- verInfo – GPIO version information

static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, *gpio_pin_interrupt_control_t* mask)

lock or unlock secure privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Disable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputEnable(GPIO_Type *base, uint32_t mask)
Enable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputDisable(GPIO_Type *base, uint32_t mask)
Disable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)
Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

uint32_t GPIO_PortGetInterruptFlags(GPIO_Type *base)

Reads the GPIO port interrupt status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

Return values

The – current GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

void GPIO_PortClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears multiple GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

- mask – GPIO pin number macro

void GPIO_CheckAttributeBytes(GPIO_Type *base, *gpio_checker_attribute_t* attribute)

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

2.21 Iomuxc_driver

static inline void IOMUXC_SetPinMux(uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t inputInv)

Sets the IOMUXC pin mux mode.

This is an example to set the PTA2 as the lpuart0_tx:

```
IOMUXC_SetPinMux(IOMUXC_PTA2_LPUART0_TX, 0);
```

This is an example to set the PTB6 as i2s1_txd3:

```
IOMUXC_SetPinMux(IOMUXC_PTB6_I2S1_TXD3, 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.
- inputInv – The pad->module input inversion.

static inline void IOMUXC_SetPinConfig(uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t configValue)

Sets the IOMUXC pin configuration.

This is an example to set pin configuration for IOMUXC_PTA7_LPUART1_RX:

```
IOMUXC_SetPinConfig(IOMUXC_PTA7_LPUART1_RX,
                    IOMUXC0_SW_MUX_CTL_PAD_PE_MASK |
                    IOMUXC0_SW_MUX_CTL_PAD_PS_MASK)
```

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_PTA0_CMP0_IN1_3V

IOMUXC_PTA0_PTA0

IOMUXC_PTA0_LPSPI0_PCS1

IOMUXC_PTA0_LPUART0_CTS_B

IOMUXC_PTA0_LPI2C0_SCL

IOMUXC_PTA0_TPM0_CLKIN

IOMUXC_PTA0_I2S0_RX_BCLK

IOMUXC_PTA0_LLWU0_P0

IOMUXC_PTA1_CMP0_IN2_3V

IOMUXC_PTA1_PTA1

IOMUXC_PTA1_LPSPI0_PCS2

IOMUXC_PTA1_LPUART0_RTS_B

IOMUXC_PTA1_LPI2C0_SDA

IOMUXC_PTA1_TPM0_CH0

IOMUXC_PTA1_I2S0_RX_FS

IOMUXC_PTA2_CMP1_IN2_3V

IOMUXC_PTA2_PTA2

IOMUXC_PTA2_LPSPI0_PCS3

IOMUXC_PTA2_LPUART0_TX

IOMUXC_PTA2_LPI2C0_HREQ

IOMUXC_PTA2_TPM0_CH1

IOMUXC_PTA2_I2S0_RXD0

IOMUXC_PTA3_CMP1_IN4_3V

IOMUXC_PTA3_PTA3

IOMUXC_PTA3_LPSPI0_PCS0

IOMUXC_PTA3_LPUART0_RX

IOMUXC_PTA3_TPM0_CH2

IOMUXC_PTA3_I2S0_RXD1

IOMUXC_PTA3_CMP0_OUT

IOMUXC_PTA3_LLWU0_P1

IOMUXC_PTA4_ADC1_CH3A

IOMUXC_PTA4_PTA4

IOMUXC_PTA4_LPSPI0_SIN

IOMUXC_PTA4_LPUART1_CTS_B

IOMUXC_PTA4_LPI2C1_SCL

IOMUXC_PTA4_TPM0_CH3

IOMUXC_PTA4_I2S0_MCLK

IOMUXC_PTA5_ADC1_CH3B

IOMUXC_PTA5_PTA5

IOMUXC_PTA5_LPSPI0_SOUT

IOMUXC_PTA5_LPUART1_RTS_B

IOMUXC_PTA5_LPI2C1_SDA

IOMUXC_PTA5_TPM0_CH4

IOMUXC_PTA5_I2S0_TX_BCLK

IOMUXC_PTA6_ADC1_CH4A

IOMUXC_PTA6_PTA6

IOMUXC_PTA6_LPSPI0_SCK

IOMUXC_PTA6_LPUART1_TX

IOMUXC_PTA6_LPI2C1_HREQ

IOMUXC_PTA6_TPM0_CH5

IOMUXC_PTA6_I2S0_TX_FS

IOMUXC_PTA7_ADC1_CH4B

IOMUXC_PTA7_PTA7

IOMUXC_PTA7_LPUART1_RX

IOMUXC_PTA7_TPM1_CH1

IOMUXC_PTA7_I2S0_TXD0

IOMUXC_PTA8_ADC1_CH5A

IOMUXC_PTA8_PTA8

IOMUXC_PTA8_LPSPI1_PCS1

IOMUXC_PTA8_LPUART2_CTS_B

IOMUXC_PTA8_LPI2C2_SCL

IOMUXC_PTA8_TPM1_CLKIN

IOMUXC_PTA8_I2S0_TXD1

IOMUXC_PTA9_ADC1_CH5B

IOMUXC_PTA9_PTA9

IOMUXC_PTA9_LPSPI1_PCS2

IOMUXC_PTA9_LPUART2_RTS_B

IOMUXC_PTA9_LPI2C2_SDA

IOMUXC_PTA9_TPM1_CH0

IOMUXC_PTA9_NMI0_B

IOMUXC_PTA10_ADC1_CH6A

IOMUXC_PTA10_PTA10

IOMUXC_PTA10_LPSPI1_PCS3

IOMUXC_PTA10_LPUART2_TX

IOMUXC_PTA10_LPI2C2_HREQ

IOMUXC_PTA10_TPM2_CLKIN

IOMUXC_PTA10_I2S0_RX_BCLK

IOMUXC_PTA11_ADC1_CH6B

IOMUXC_PTA11_PTA11

IOMUXC_PTA11_LPUART2_RX

IOMUXC_PTA11_TPM2_CH0

IOMUXC_PTA11_I2S0_RX_FS

IOMUXC_PTA12_ADC1_CH7A

IOMUXC_PTA12_PTA12

IOMUXC_PTA12_LPSPI1_SIN

IOMUXC_PTA12_LPUART3_CTS_B

IOMUXC_PTA12_LPI2C3_SCL

IOMUXC_PTA12_TPM2_CH1

IOMUXC_PTA12_I2S0_RXD0

IOMUXC_PTA13_ADC1_CH7B

IOMUXC_PTA13_PTA13

IOMUXC_PTA13_LPSPI1_SOUT

IOMUXC_PTA13_LPUART3_RTS_B

IOMUXC_PTA13_LPI2C3_SDA

IOMUXC_PTA13_TPM3_CLKIN

IOMUXC_PTA13_I2S0_RXD1

IOMUXC_PTA13_CMP0_OUT

IOMUXC_PTA13_LLWU0_P2

IOMUXC_PTA14_ADC1_CH8A

IOMUXC_PTA14_PTA14

IOMUXC_PTA14_LPSPI1_SCK

IOMUXC_PTA14_LPUART3_TX

IOMUXC_PTA14_LPI2C3_HREQ

IOMUXC_PTA14_TPM3_CH0

IOMUXC_PTA14_I2S0_MCLK

IOMUXC_PTA14_LLWU0_P3

IOMUXC_PTA15_ADC1_CH8B

IOMUXC_PTA15_PTA15

IOMUXC_PTA15_LPSPI1_PCS0

IOMUXC_PTA15_LPUART3_RX

IOMUXC_PTA15_TPM3_CH1

IOMUXC_PTA15_I2S0_TX_BCLK

IOMUXC_PTA16_CMP1_IN5_3V

IOMUXC_PTA16_PTA16

IOMUXC_PTA16_FXIO0_D0

IOMUXC_PTA16_LPSPI0_SOUT

IOMUXC_PTA16_LPUART0_CTS_B

IOMUXC_PTA16_LPI2C0_SCL

IOMUXC_PTA16_TPM3_CH2

IOMUXC_PTA16_I2S0_TX_FS

IOMUXC_PTA17_CMP1_IN6_3V

IOMUXC_PTA17_PTA17

IOMUXC_PTA17_FXIO0_D1

IOMUXC_PTA17_LPSPI0_SCK

IOMUXC_PTA17_LPUART0_RTS_B

IOMUXC_PTA17_LPI2C0_SDA

IOMUXC_PTA17_TPM3_CH3

IOMUXC_PTA17_I2S0_TXD0

IOMUXC_PTA18_CMP1_IN1_3V

IOMUXC_PTA18_PTA18

IOMUXC_PTA18_FXIO0_D2

IOMUXC_PTA18_LPSPI0_PCS0

IOMUXC_PTA18_LPUART0_TX

IOMUXC_PTA18_LPI2C0_HREQ

IOMUXC_PTA18_TPM3_CH4

IOMUXC_PTA18_I2S0_TXD1

IOMUXC_PTA18_LLWU0_P4

IOMUXC_PTA19_CMP1_IN3_3V

IOMUXC_PTA19_PTA19

IOMUXC_PTA19_FXIO0_D3

IOMUXC_PTA19_LPUART0_RX

IOMUXC_PTA19_TPM3_CH5

IOMUXC_PTA19_I2S1_RX_BCLK

IOMUXC_PTA19_LPTMR0_ALT3

IOMUXC_PTA19_LLWU0_P5

IOMUXC_PTA20_ADC0_CH10A

IOMUXC_PTA20_PTA20

IOMUXC_PTA20_FXIO0_D4

IOMUXC_PTA20_LPSPI0_SIN

IOMUXC_PTA20_LPUART1_CTS_B

IOMUXC_PTA20_LPI2C1_SCL

IOMUXC_PTA20_TPM0_CLKIN

IOMUXC_PTA20_I2S1_RX_FS

IOMUXC_PTA21_ADC0_CH10B

IOMUXC_PTA21_PTA21

IOMUXC_PTA21_FXIO0_D5

IOMUXC_PTA21_LPSPI0_PCS1

IOMUXC_PTA21_LPUART1_RTS_B

IOMUXC_PTA21_LPI2C1_SDA

IOMUXC_PTA21_TPM0_CH0

IOMUXC_PTA21_I2S1_RXD0

IOMUXC_PTA22_ADC0_CH9A

IOMUXC_PTA22_PTA22

IOMUXC_PTA22_FXIO0_D6

IOMUXC_PTA22_LPSPI0_PCS2

IOMUXC_PTA22_LPUART1_TX

IOMUXC_PTA22_LPI2C1_HREQ

IOMUXC_PTA22_TPM0_CH1

IOMUXC_PTA22_I2S1_RXD1

IOMUXC_PTA22_LPTMR0_ALT2

IOMUXC_PTA22_EWM_OUT_B

IOMUXC_PTA23_ADC0_CH9B

IOMUXC_PTA23_PTA23

IOMUXC_PTA23_FXIO0_D7

IOMUXC_PTA23_LPSPI0_PCS3

IOMUXC_PTA23_LPUART1_RX

IOMUXC_PTA23_TPM0_CH2

IOMUXC_PTA23_I2S1_MCLK

IOMUXC_PTA23_LLWU0_P6

IOMUXC_PTA24_ADC0_CH8A

IOMUXC_PTA24_PTA24

IOMUXC_PTA24_FXIO0_D8

IOMUXC_PTA24_LPSPI1_PCS1

IOMUXC_PTA24_LPUART2_CTS_B

IOMUXC_PTA24_LPI2C2_SCL

IOMUXC_PTA24_TPM0_CH3

IOMUXC_PTA24_I2S1_TX_BCLK

IOMUXC_PTA25_ADC0_CH8B

IOMUXC_PTA25_PTA25

IOMUXC_PTA25_FXIO0_D9

IOMUXC_PTA25_LPSPI1_PCS2

IOMUXC_PTA25_LPUART2_RTS_B

IOMUXC_PTA25_LPI2C2_SDA

IOMUXC_PTA25_TPM0_CH4

IOMUXC_PTA25_I2S1_TX_FS

IOMUXC_PTA26_PTA26

IOMUXC_PTA26_JTAG_TMS_SWD_DIO

IOMUXC_PTA26_FXIO0_D10

IOMUXC_PTA26_LPSPI1_PCS3

IOMUXC_PTA26_LPUART2_TX

IOMUXC_PTA26_LPI2C2_HREQ

IOMUXC_PTA26_TPM0_CH5

IOMUXC_PTA26_I2S1_RXD2

IOMUXC_PTA27_PTA27

IOMUXC_PTA27_JTAG_TDO

IOMUXC_PTA27_FXIO0_D11

IOMUXC_PTA27_LPUART2_RX

IOMUXC_PTA27_TPM1_CH1

IOMUXC_PTA27_I2S1_RXD3

IOMUXC_PTA28_PTA28

IOMUXC_PTA28_JTAG_TDI

IOMUXC_PTA28_FXIO0_D12

IOMUXC_PTA28_LPSPI1_SIN

IOMUXC_PTA28_LPUART3_CTS_B

IOMUXC_PTA28_LPI2C3_SCL

IOMUXC_PTA28_TPM1_CLKIN

IOMUXC_PTA28_I2S1_TXD2

IOMUXC_PTA29_PTA29

IOMUXC_PTA29_JTAG_TCLK_SWD_CLK

IOMUXC_PTA29_FXIO0_D13

IOMUXC_PTA29_LPSPI1_SOUT

IOMUXC_PTA29_LPUART3_RTS_B

IOMUXC_PTA29_LPI2C3_SDA

IOMUXC_PTA29_TPM1_CH0

IOMUXC_PTA29_I2S1_TXD3

IOMUXC_PTA30_ADC0_CH1A

IOMUXC_PTA30_PTA30

IOMUXC_PTA30_FXIO0_D14

IOMUXC_PTA30_LPSPI1_SCK

IOMUXC_PTA30_LPUART3_TX

IOMUXC_PTA30_LPI2C3_HREQ

IOMUXC_PTA30_TPM2_CLKIN

IOMUXC_PTA30_I2S1_TXD0

IOMUXC_PTA30_JTAG_TRST_B

IOMUXC_PTA31_ADC0_CH1B

IOMUXC_PTA31_PTA31

IOMUXC_PTA31_FXIO0_D15

IOMUXC_PTA31_LPSPI1_PCS0

IOMUXC_PTA31_LPUART3_RX

IOMUXC_PTA31_TPM2_CH0

IOMUXC_PTA31_I2S1_TXD1

IOMUXC_PTA31_LPTMR0_ALT1

IOMUXC_PTA31_EWM_IN

IOMUXC_PTA31_LLWU0_P7

IOMUXC_PTB0_ADC0_CH0A

IOMUXC_PTB0_PTB0

IOMUXC_PTB0_FXIO0_D16

IOMUXC_PTB0_LPSPI0_SIN

IOMUXC_PTB0_LPUART0_TX

IOMUXC_PTB0_TPM2_CH1

IOMUXC_PTB0_CLKOUT0

IOMUXC_PTB0_CMP1_OUT

IOMUXC_PTB0_EWM_OUT_B

IOMUXC_PTB1_ADC0_CH0B

IOMUXC_PTB1_PTB1

IOMUXC_PTB1_FXIO0_D17

IOMUXC_PTB1_LPSPI0_SOUT

IOMUXC_PTB1_LPUART0_RX

IOMUXC_PTB1_TPM3_CLKIN

IOMUXC_PTB1_I2S1_TX_BCLK

IOMUXC_PTB1_RTC_CLKOUT

IOMUXC_PTB1_EWM_IN

IOMUXC_PTB1_LLWU0_P8

IOMUXC_PTB2_ADC0_CH6A

IOMUXC_PTB2_PTB2

IOMUXC_PTB2_FXIO0_D18

IOMUXC_PTB2_LPSPI0_SCK

IOMUXC_PTB2_LPUART1_TX

IOMUXC_PTB2_TPM3_CH0

IOMUXC_PTB2_I2S1_TX_FS

IOMUXC_PTB2_TRACE_CLKOUT

IOMUXC_PTB3_ADC0_CH6B

IOMUXC_PTB3_PTB3

IOMUXC_PTB3_FXIO0_D19

IOMUXC_PTB3_LPSPI0_PCS0

IOMUXC_PTB3_LPUART1_RX

IOMUXC_PTB3_TPM3_CH1

IOMUXC_PTB3_I2S1_TXD0

IOMUXC_PTB3_TRACE_D0

IOMUXC_PTB3_LPTMR1_ALT2

IOMUXC_PTB3_LLWU0_P9

IOMUXC_PTB4_PTB4

IOMUXC_PTB4_FXIO0_D20

IOMUXC_PTB4_LPSPI0_PCS1

IOMUXC_PTB4_LPUART2_TX

IOMUXC_PTB4_LPI2C0_HREQ

IOMUXC_PTB4_TPM3_CH2

IOMUXC_PTB4_I2S1_TXD1

IOMUXC_PTB4_QSPIA_DATA7

IOMUXC_PTB4_TRACE_D1

IOMUXC_PTB4_SEC_VIO_B

IOMUXC_PTB5_PTB5

IOMUXC_PTB5_FXIO0_D21

IOMUXC_PTB5_LPSPI0_PCS2

IOMUXC_PTB5_LPUART2_RX

IOMUXC_PTB5_LPI2C1_HREQ

IOMUXC_PTB5_TPM3_CH3

IOMUXC_PTB5_I2S1_TXD2

IOMUXC_PTB5_QSPIA_DATA6

IOMUXC_PTB5_TRACE_D2

IOMUXC_PTB5_RTC_CLKOUT

IOMUXC_PTB6_ADC1_CH1A

IOMUXC_PTB6_PTB6

IOMUXC_PTB6_FXIO0_D22

IOMUXC_PTB6_LPSPI0_PCS3

IOMUXC_PTB6_LPUART3_TX

IOMUXC_PTB6_LPI2C0_SCL

IOMUXC_PTB6_TPM3_CH4

IOMUXC_PTB6_I2S1_TXD3

IOMUXC_PTB6_QSPIA_DATA5

IOMUXC_PTB6_TRACE_D3

IOMUXC_PTB6_LPTMR1_ALT3

IOMUXC_PTB6_LLWU0_P10

IOMUXC_PTB7_ADC1_CH1B

IOMUXC_PTB7_PTB7

IOMUXC_PTB7_FXIO0_D23

IOMUXC_PTB7_LPSPI1_SIN

IOMUXC_PTB7_LPUART3_RX

IOMUXC_PTB7_LPI2C0_SDA

IOMUXC_PTB7_TPM3_CH5

IOMUXC_PTB7_I2S1_MCLK

IOMUXC_PTB7_QSPIA_SS1_B

IOMUXC_PTB7_CMP1_OUT

IOMUXC_PTB7_LLWU0_P11

IOMUXC_PTB8_ADC0_CH14A_CMP0_IN0

IOMUXC_PTB8_PTB8

IOMUXC_PTB8_FXIO0_D24

IOMUXC_PTB8_LPSPI1_SOUT

IOMUXC_PTB8_LPI2C1_SCL

IOMUXC_PTB8_TPM0_CLKIN

IOMUXC_PTB8_I2S1_RX_BCLK

IOMUXC_PTB8_QSPIA_SS0_B

IOMUXC_PTB8_RTC_CLKOUT

IOMUXC_PTB9_ADC0_CH14B_CMP0_IN2

IOMUXC_PTB9_PTB9

IOMUXC_PTB9_FXIO0_D25

IOMUXC_PTB9_LPSPI1_SCK

IOMUXC_PTB9_LPI2C1_SDA

IOMUXC_PTB9_TPM0_CH0

IOMUXC_PTB9_I2S1_RX_FS

IOMUXC_PTB9_QSPIA_DQS

IOMUXC_PTB9_LLWU0_P12

IOMUXC_PTB10_CMP0_IN1

IOMUXC_PTB10_PTB10

IOMUXC_PTB10_FXIO0_D26

IOMUXC_PTB10_LPSPI1_PCS0

IOMUXC_PTB10_LPI2C2_SCL

IOMUXC_PTB10_TPM0_CH1

IOMUXC_PTB10_I2S1_RXD0

IOMUXC_PTB10_TRACE_D4

IOMUXC_PTB11_CMP0_IN3

IOMUXC_PTB11_PTB11

IOMUXC_PTB11_FXIO0_D27

IOMUXC_PTB11_LPSPI1_PCS1

IOMUXC_PTB11_LPI2C2_SDA

IOMUXC_PTB11_TPM1_CLKIN

IOMUXC_PTB11_I2S1_RXD1

IOMUXC_PTB11_TRACE_D5

IOMUXC_PTB12_ADC1_CH13A_CMP1_IN0

IOMUXC_PTB12_PTB12

IOMUXC_PTB12_FXIO0_D28

IOMUXC_PTB12_LPSPI1_PCS2

IOMUXC_PTB12_LPUART2_TX

IOMUXC_PTB12_LPI2C3_SCL

IOMUXC_PTB12_TPM1_CH0

IOMUXC_PTB12_I2S1_RXD2

IOMUXC_PTB12_TRACE_D6

IOMUXC_PTB13_ADC1_CH13B_CMP1_IN1

IOMUXC_PTB13_PTB13

IOMUXC_PTB13_FXIO0_D29

IOMUXC_PTB13_LPSPI1_PCS3

IOMUXC_PTB13_LPUART2_RX

IOMUXC_PTB13_LPI2C3_SDA

IOMUXC_PTB13_TPM1_CH1

IOMUXC_PTB13_I2S1_RXD3

IOMUXC_PTB13_QSPIA_DATA4

IOMUXC_PTB13_TRACE_D7

IOMUXC_PTB14_ADC1_CH2A

IOMUXC_PTB14_PTB14

IOMUXC_PTB14_FXIO0_D30

IOMUXC_PTB14_LPI2C2_HREQ

IOMUXC_PTB14_TPM2_CLKIN

IOMUXC_PTB14_QSPIA_SS1_B

IOMUXC_PTB14_QSPIA_SCLK_B

IOMUXC_PTB14_RTC_CLKOUT

IOMUXC_PTB14_LLWU0_P13

IOMUXC_PTB15_ADC1_CH2B

IOMUXC_PTB15_PTB15

IOMUXC_PTB15_FXIO0_D31

IOMUXC_PTB15_LPI2C3_HREQ

IOMUXC_PTB15_TPM2_CH0

IOMUXC_PTB15_QSPIA_SCLK

IOMUXC_PTB16_ADC0_CH4A

IOMUXC_PTB16_PTB16

IOMUXC_PTB16_TPM2_CH1

IOMUXC_PTB16_QSPIA_DATA3

IOMUXC_PTB16_LLWU0_P14

IOMUXC_PTB17_ADC0_CH4B

IOMUXC_PTB17_PTB17

IOMUXC_PTB17_TPM3_CLKIN

IOMUXC_PTB17_QSPIA_DATA2

IOMUXC_PTB18_ADC0_CH5A

IOMUXC_PTB18_PTB18

IOMUXC_PTB18_TPM3_CH0

IOMUXC_PTB18_QSPIA_DATA1

IOMUXC_PTB19_ADC0_CH5B

IOMUXC_PTB19_PTB19

IOMUXC_PTB19_TPM3_CH1

IOMUXC_PTB19_QSPIA_DATA0

IOMUXC_PTB19_USB0_ID

IOMUXC_PTB19_LLWU0_P15

IOMUXC_PTC0_PTC0

IOMUXC_PTC0_LPUART4_CTS_B

IOMUXC_PTC0_LPI2C4_SCL

IOMUXC_PTC0_TPM4_CLKIN

IOMUXC_PTC0_FB_AD0

IOMUXC_PTC0_TRACE_D15

IOMUXC_PTC1_PTC1

IOMUXC_PTC1_LPUART4_RTS_B

IOMUXC_PTC1_LPI2C4_SDA

IOMUXC_PTC1_TPM4_CH0

IOMUXC_PTC1_FB_AD1

IOMUXC_PTC1_TRACE_D14

IOMUXC_PTC2_PTC2

IOMUXC_PTC2_LPUART4_TX

IOMUXC_PTC2_LPI2C4_HREQ

IOMUXC_PTC2_TPM4_CH1

IOMUXC_PTC2_FB_AD2

IOMUXC_PTC2_TRACE_D13

IOMUXC_PTC3_PTC3

IOMUXC_PTC3_LPUART4_RX

IOMUXC_PTC3_TPM4_CH2

IOMUXC_PTC3_FB_AD3

IOMUXC_PTC3_TRACE_D12

IOMUXC_PTC4_PTC4

IOMUXC_PTC4_FXIO1_D0

IOMUXC_PTC4_LPSPI2_PCS1

IOMUXC_PTC4_LPUART5_CTS_B

IOMUXC_PTC4_LPI2C5_SCL

IOMUXC_PTC4_TPM4_CH3

IOMUXC_PTC4_FB_AD4

IOMUXC_PTC4_TRACE_D11

IOMUXC_PTC5_PTC5

IOMUXC_PTC5_FXIO1_D1

IOMUXC_PTC5_LPSPI2_PCS2

IOMUXC_PTC5_LPUART5_RTS_B

IOMUXC_PTC5_LPI2C5_SDA

IOMUXC_PTC5_TPM4_CH4

IOMUXC_PTC5_FB_AD5

IOMUXC_PTC5_TRACE_D10

IOMUXC_PTC6_PTC6

IOMUXC_PTC6_FXIO1_D2

IOMUXC_PTC6_LPSPI2_PCS3

IOMUXC_PTC6_LPUART5_TX

IOMUXC_PTC6_LPI2C5_HREQ

IOMUXC_PTC6_TPM4_CH5

IOMUXC_PTC6_FB_AD6

IOMUXC_PTC6_TRACE_D9

IOMUXC_PTC7_PTC7

IOMUXC_PTC7_FXIO1_D3

IOMUXC_PTC7_LPUART5_RX

IOMUXC_PTC7_TPM5_CH1

IOMUXC_PTC7_FB_AD7

IOMUXC_PTC7_TRACE_D8

IOMUXC_PTC8_PTC8

IOMUXC_PTC8_FXIO1_D4

IOMUXC_PTC8_LPSPI2_SIN

IOMUXC_PTC8_LPUART6_CTS_B

IOMUXC_PTC8_LPI2C6_SCL

IOMUXC_PTC8_TPM5_CLKIN

IOMUXC_PTC8_FB_AD8

IOMUXC_PTC8_TRACE_D7

IOMUXC_PTC9_PTC9

IOMUXC_PTC9_FXIO1_D5

IOMUXC_PTC9_LPSPI2_SOUT

IOMUXC_PTC9_LPUART6_RTS_B

IOMUXC_PTC9_LPI2C6_SDA

IOMUXC_PTC9_TPM5_CH0

IOMUXC_PTC9_FB_AD9

IOMUXC_PTC9_TRACE_D6

IOMUXC_PTC10_PTC10

IOMUXC_PTC10_FXIO1_D6

IOMUXC_PTC10_LPSPI2_SCK

IOMUXC_PTC10_LPUART6_TX

IOMUXC_PTC10_LPI2C6_HREQ

IOMUXC_PTC10_TPM7_CH3

IOMUXC_PTC10_FB_AD10

IOMUXC_PTC10_TRACE_D5

IOMUXC_PTC11_PTC11

IOMUXC_PTC11_FXIO1_D7

IOMUXC_PTC11_LPSPI2_PCS0

IOMUXC_PTC11_LPUART6_RX

IOMUXC_PTC11_TPM7_CH4

IOMUXC_PTC11_FB_AD11

IOMUXC_PTC11_TRACE_D4

IOMUXC_PTC12_PTC12

IOMUXC_PTC12_FXIO1_D8

IOMUXC_PTC12_LPSPI3_PCS1

IOMUXC_PTC12_LPUART7_CTS_B

IOMUXC_PTC12_LPI2C7_SCL

IOMUXC_PTC12_TPM7_CH5

IOMUXC_PTC12_FB_AD12

IOMUXC_PTC12_TRACE_D3

IOMUXC_PTC13_PTC13

IOMUXC_PTC13_FXIO1_D9

IOMUXC_PTC13_LPSPI3_PCS2

IOMUXC_PTC13_LPUART7_RTS_B

IOMUXC_PTC13_LPI2C7_SDA

IOMUXC_PTC13_TPM7_CLKIN

IOMUXC_PTC13_FB_AD13

IOMUXC_PTC13_TRACE_D2

IOMUXC_PTC13_USB0_ID

IOMUXC_PTC14_PTC14

IOMUXC_PTC14_FXIO1_D10

IOMUXC_PTC14_LPSPI3_PCS3

IOMUXC_PTC14_LPUART7_TX

IOMUXC_PTC14_LPI2C7_HREQ

IOMUXC_PTC14_TPM7_CH0

IOMUXC_PTC14_FB_AD14

IOMUXC_PTC14_TRACE_D1

IOMUXC_PTC15_PTC15

IOMUXC_PTC15_FXIO1_D11

IOMUXC_PTC15_LPUART7_RX

IOMUXC_PTC15_TPM7_CH1

IOMUXC_PTC15_FB_AD15

IOMUXC_PTC15_TRACE_D0

IOMUXC_PTC16_PTC16

IOMUXC_PTC16_FXIO1_D12

IOMUXC_PTC16_LPSPI3_SIN

IOMUXC_PTC16_TPM7_CH2

IOMUXC_PTC16_FB_ALE_FB_CS1_B_FB_TS_B

IOMUXC_PTC16_TRACE_CLKOUT

IOMUXC_PTC16_USB1_OC2

IOMUXC_PTC17_PTC17

IOMUXC_PTC17_FXIO1_D13

IOMUXC_PTC17_LPSPI3_SOUT

IOMUXC_PTC17_TPM6_CLKIN

IOMUXC_PTC17_FB_CS0_B

IOMUXC_PTC18_PTC18

IOMUXC_PTC18_FXIO1_D14

IOMUXC_PTC18_LPSPI3_SCK

IOMUXC_PTC18_TPM6_CH0

IOMUXC_PTC18_FB_OE_B

IOMUXC_PTC18_USB0_ID

IOMUXC_PTC18_VIU_DE

IOMUXC_PTC19_PTC19

IOMUXC_PTC19_FXIO1_D15

IOMUXC_PTC19_LPSPI3_PCS0

IOMUXC_PTC19_TPM6_CH1

IOMUXC_PTC19_FB_A16

IOMUXC_PTC19_USB0_ID

IOMUXC_PTC19_USB1_PWR2

IOMUXC_PTC19_VIU_DE

IOMUXC_PTD0_PTD0

IOMUXC_PTD0_SDHC0_RESET_B

IOMUXC_PTD1_PTD1

IOMUXC_PTD1_SDHC0_CMD

IOMUXC_PTD2_PTD2

IOMUXC_PTD2_SDHC0_CLK

IOMUXC_PTD3_PTD3

IOMUXC_PTD3_SDHC0_D7

IOMUXC_PTD4_PTD4

IOMUXC_PTD4_SDHC0_D6

IOMUXC_PTD5_PTD5

IOMUXC_PTD5_SDHC0_D5

IOMUXC_PTD6_PTD6

IOMUXC_PTD6_SDHC0_D4

IOMUXC_PTD7_PTD7

IOMUXC_PTD7_SDHC0_D3

IOMUXC_PTD8_PTD8

IOMUXC_PTD8_TPM4_CLKIN

IOMUXC_PTD8_SDHC0_D2

IOMUXC_PTD9_PTD9

IOMUXC_PTD9_TPM4_CH0

IOMUXC_PTD9_SDHC0_D1

IOMUXC_PTD10_PTD10

IOMUXC_PTD10_TPM4_CH1

IOMUXC_PTD10_SDHC0_D0

IOMUXC_PTD11_PTD11

IOMUXC_PTD11_TPM4_CH2

IOMUXC_PTD11_SDHC0_DQS

IOMUXC_PTE0_PTE0

IOMUXC_PTE0_FXIO1_D31

IOMUXC_PTE0_LPSPI2_PCS1

IOMUXC_PTE0_LPUART4_CTS_B

IOMUXC_PTE0_LPI2C4_SCL

IOMUXC_PTE0_SDHC1_D1

IOMUXC_PTE0_FB_A25

IOMUXC_PTE1_PTE1

IOMUXC_PTE1_FXIO1_D30

IOMUXC_PTE1_LPSPI2_PCS2

IOMUXC_PTE1_LPUART4_RTS_B

IOMUXC_PTE1_LPI2C4_SDA

IOMUXC_PTE1_SDHC1_D0

IOMUXC_PTE1_FB_A26

IOMUXC_PTE2_PTE2

IOMUXC_PTE2_FXIO1_D29

IOMUXC_PTE2_LPSPI2_PCS3

IOMUXC_PTE2_LPUART4_TX

IOMUXC_PTE2_LPI2C4_HREQ

IOMUXC_PTE2_SDHC1_CLK

IOMUXC_PTE3_PTE3

IOMUXC_PTE3_FXIO1_D28

IOMUXC_PTE3_LPUART4_RX

IOMUXC_PTE3_TPM5_CH1

IOMUXC_PTE3_SDHC1_CMD

IOMUXC_PTE4_PTE4

IOMUXC_PTE4_FXIO1_D27

IOMUXC_PTE4_LPSPI2_SIN

IOMUXC_PTE4_LPUART5_CTS_B

IOMUXC_PTE4_LPI2C5_SCL

IOMUXC_PTE4_TPM5_CLKIN

IOMUXC_PTE4_SDHC1_D3

IOMUXC_PTE5_PTE5

IOMUXC_PTE5_FXIO1_D26

IOMUXC_PTE5_LPSPI2_SOUT

IOMUXC_PTE5_LPUART5_RTS_B

IOMUXC_PTE5_LPI2C5_SDA

IOMUXC_PTE5_TPM5_CH0

IOMUXC_PTE5_SDHC1_D2

IOMUXC_PTE5_VIU_DE

IOMUXC_PTE6_PTE6

IOMUXC_PTE6_FXIO1_D25

IOMUXC_PTE6_LPSPI2_SCK

IOMUXC_PTE6_LPUART5_TX

IOMUXC_PTE6_LPI2C5_HREQ

IOMUXC_PTE6_TPM7_CH3

IOMUXC_PTE6_SDHC1_D4

IOMUXC_PTE6_FB_A17

IOMUXC_PTE6_USB0_OC

IOMUXC_PTE7_PTE7

IOMUXC_PTE7_FXIO1_D24

IOMUXC_PTE7_LPSPI2_PCS0

IOMUXC_PTE7_LPUART5_RX

IOMUXC_PTE7_TPM7_CH4

IOMUXC_PTE7_SDHC1_D5

IOMUXC_PTE7_FB_A18

IOMUXC_PTE7_TRACE_D7

IOMUXC_PTE7_USB0_PWR

IOMUXC_PTE7_VIU_FID

IOMUXC_PTE8_PTE8

IOMUXC_PTE8_TRACE_D6

IOMUXC_PTE8_VIU_D16

IOMUXC_PTE8_FXIO1_D23

IOMUXC_PTE8_LPSPI3_PCS1

IOMUXC_PTE8_LPUART6_CTS_B

IOMUXC_PTE8_LPI2C6_SCL

IOMUXC_PTE8_TPM7_CH5

IOMUXC_PTE8_SDHC1_WP

IOMUXC_PTE8_SDHC1_D6

IOMUXC_PTE8_FB_CS3_B_FB_BE7_0_BLS31_24_B

IOMUXC_PTE9_PTE9

IOMUXC_PTE9_TRACE_D5

IOMUXC_PTE9_VIU_D17

IOMUXC_PTE9_FXIO1_D22

IOMUXC_PTE9_LPSPI3_PCS2

IOMUXC_PTE9_LPUART6_RTS_B

IOMUXC_PTE9_LPI2C6_SDA

IOMUXC_PTE9_TPM7_CLKIN

IOMUXC_PTE9_SDHC1_CD

IOMUXC_PTE9_SDHC1_D7

IOMUXC_PTE9_FB_TBST_B_FB_CS2_B_FB_BE15_8_BLS23_16_B

IOMUXC_PTE10_PTE10

IOMUXC_PTE10_TRACE_D4

IOMUXC_PTE10_VIU_D18

IOMUXC_PTE10_FXIO1_D21

IOMUXC_PTE10_LPSPI3_PCS3

IOMUXC_PTE10_LPUART6_TX

IOMUXC_PTE10_LPI2C6_HREQ

IOMUXC_PTE10_TPM7_CH0

IOMUXC_PTE10_SDHC1_VS

IOMUXC_PTE10_SDHC1_DQS

IOMUXC_PTE10_FB_A19

IOMUXC_PTE11_PTE11

IOMUXC_PTE11_TRACE_D3

IOMUXC_PTE11_VIU_D19

IOMUXC_PTE11_FXIO1_D20

IOMUXC_PTE11_LPUART6_RX

IOMUXC_PTE11_TPM7_CH1

IOMUXC_PTE11_SDHC1_RESET_B

IOMUXC_PTE11_FB_A20

IOMUXC_PTE12_PTE12

IOMUXC_PTE12_FXIO1_D19

IOMUXC_PTE12_LPSPI3_SIN

IOMUXC_PTE12_LPUART7_CTS_B

IOMUXC_PTE12_LPI2C7_SCL

IOMUXC_PTE12_TPM7_CH2

IOMUXC_PTE12_SDHC1_WP

IOMUXC_PTE12_FB_A21

IOMUXC_PTE12_TRACE_D2

IOMUXC_PTE12_USB1_OC2

IOMUXC_PTE12_VIU_D20

IOMUXC_PTE13_PTE13

IOMUXC_PTE13_FXIO1_D18

IOMUXC_PTE13_LPSPI3_SOUT

IOMUXC_PTE13_LPUART7_RTS_B

IOMUXC_PTE13_LPI2C7_SDA

IOMUXC_PTE13_TPM6_CLKIN

IOMUXC_PTE13_SDHC1_CD

IOMUXC_PTE13_FB_A22

IOMUXC_PTE13_TRACE_D1

IOMUXC_PTE13_USB1_PWR2

IOMUXC_PTE13_VIU_D21

IOMUXC_PTE14_PTE14

IOMUXC_PTE14_FXIO1_D17

IOMUXC_PTE14_LPSPI3_SCK

IOMUXC_PTE14_LPUART7_TX

IOMUXC_PTE14_LPI2C7_HREQ

IOMUXC_PTE14_TPM6_CH0

IOMUXC_PTE14_SDHC1_VS

IOMUXC_PTE14_FB_A23

IOMUXC_PTE14_TRACE_D0

IOMUXC_PTE14_USB0_OC

IOMUXC_PTE14_VIU_D22

IOMUXC_PTE15_PTE15

IOMUXC_PTE15_FXIO1_D16

IOMUXC_PTE15_LPSPI3_PCS0

IOMUXC_PTE15_LPUART7_RX

IOMUXC_PTE15_TPM6_CH1

IOMUXC_PTE15_FB_A24

IOMUXC_PTE15_TRACE_CLKOUT

IOMUXC_PTE15_USB0_PWR

IOMUXC_PTE15_VIU_D23

IOMUXC_PTF0_PTF0

IOMUXC_PTF0_LPUART4_CTS_B

IOMUXC_PTF0_LPI2C4_SCL

IOMUXC_PTF0_TPM4_CLKIN

IOMUXC_PTF0_FB_RW_B

IOMUXC_PTF0_VIU_DE

IOMUXC_PTF1_PTF1

IOMUXC_PTF1_LPUART4_RTS_B

IOMUXC_PTF1_LPI2C4_SDA

IOMUXC_PTF1_TPM4_CH0

IOMUXC_PTF1_CLKOUT

IOMUXC_PTF1_VIU_HSYNC

IOMUXC_PTF2_PTF2

IOMUXC_PTF2_LPUART4_TX

IOMUXC_PTF2_LPI2C4_HREQ

IOMUXC_PTF2_TPM4_CH1

IOMUXC_PTF2_FB_TSIZ1_FB_CS5_B_FB_BE23_16_BLS15_8_B

IOMUXC_PTF2_VIU_VSYNC

IOMUXC_PTF3_PTF3

IOMUXC_PTF3_LPUART4_RX

IOMUXC_PTF3_TPM4_CH2

IOMUXC_PTF3_FB_AD16

IOMUXC_PTF3_VIU_PCLK

IOMUXC_PTF4_PTF4

IOMUXC_PTF4_FXIO1_D0

IOMUXC_PTF4_LPSPI2_PCS1

IOMUXC_PTF4_LPUART5_CTS_B

IOMUXC_PTF4_LPI2C5_SCL

IOMUXC_PTF4_TPM4_CH3

IOMUXC_PTF4_FB_AD17

IOMUXC_PTF4_VIU_D0

IOMUXC_PTF5_PTF5

IOMUXC_PTF5_FXIO1_D1

IOMUXC_PTF5_LPSPI2_PCS2

IOMUXC_PTF5_LPUART5_RTS_B

IOMUXC_PTF5_LPI2C5_SDA

IOMUXC_PTF5_TPM4_CH4

IOMUXC_PTF5_FB_AD18

IOMUXC_PTF5_VIU_D1

IOMUXC_PTF6_PTF6

IOMUXC_PTF6_FXIO1_D2

IOMUXC_PTF6_LPSPI2_PCS3

IOMUXC_PTF6_LPUART5_TX

IOMUXC_PTF6_LPI2C5_HREQ

IOMUXC_PTF6_TPM4_CH5

IOMUXC_PTF6_FB_AD19

IOMUXC_PTF6_VIU_D2

IOMUXC_PTF7_PTF7

IOMUXC_PTF7_FXIO1_D3

IOMUXC_PTF7_LPUART5_RX

IOMUXC_PTF7_TPM5_CH1

IOMUXC_PTF7_FB_AD20

IOMUXC_PTF7_VIU_D3

IOMUXC_PTF8_PTF8

IOMUXC_PTF8_FXIO1_D4

IOMUXC_PTF8_LPSPI2_SIN

IOMUXC_PTF8_LPUART6_CTS_B

IOMUXC_PTF8_LPI2C6_SCL

IOMUXC_PTF8_TPM5_CLKIN

IOMUXC_PTF8_FB_AD21

IOMUXC_PTF8_USB1_CLK

IOMUXC_PTF8_VIU_D4

IOMUXC_PTF9_PTF9

IOMUXC_PTF9_FXIO1_D5

IOMUXC_PTF9_LPSPI2_SOUT

IOMUXC_PTF9_LPUART6_RTS_B

IOMUXC_PTF9_LPI2C6_SDA

IOMUXC_PTF9_TPM5_CH0

IOMUXC_PTF9_FB_AD22

IOMUXC_PTF9_USB1_NXT

IOMUXC_PTF9_VIU_D5

IOMUXC_PTF10_PTF10

IOMUXC_PTF10_FXIO1_D6

IOMUXC_PTF10_LPSPI2_SCK

IOMUXC_PTF10_LPUART6_TX

IOMUXC_PTF10_LPI2C6_HREQ

IOMUXC_PTF10_TPM7_CH3

IOMUXC_PTF10_FB_AD23

IOMUXC_PTF10_USB1_STP

IOMUXC_PTF10_VIU_D6

IOMUXC_PTF11_PTF11

IOMUXC_PTF11_FXIO1_D7

IOMUXC_PTF11_LPSPI2_PCS0

IOMUXC_PTF11_LPUART6_RX

IOMUXC_PTF11_TPM7_CH4

IOMUXC_PTF11_FB_CS4_B_FB_TSIZ0_FB_BE31_24_BLS7_0_B

IOMUXC_PTF11_USB1_DIR

IOMUXC_PTF11_VIU_D7

IOMUXC_PTF12_PTF12

IOMUXC_PTF12_FXIO1_D8

IOMUXC_PTF12_LPSPI3_PCS1

IOMUXC_PTF12_LPUART7_CTS_B

IOMUXC_PTF12_LPI2C7_SCL

IOMUXC_PTF12_TPM7_CH5

IOMUXC_PTF12_FB_AD24

IOMUXC_PTF12_USB1_DATA0

IOMUXC_PTF12_VIU_D8

IOMUXC_PTF13_PTF13

IOMUXC_PTF13_FXIO1_D9

IOMUXC_PTF13_LPSPI3_PCS2

IOMUXC_PTF13_LPUART7_RTS_B

IOMUXC_PTF13_LPI2C7_SDA

IOMUXC_PTF13_TPM7_CLKIN

IOMUXC_PTF13_FB_AD25

IOMUXC_PTF13_USB1_DATA1

IOMUXC_PTF13_VIU_D9

IOMUXC_PTF14_PTF14

IOMUXC_PTF14_FXIO1_D10

IOMUXC_PTF14_LPSPI3_PCS3

IOMUXC_PTF14_LPUART7_TX

IOMUXC_PTF14_LPI2C7_HREQ

IOMUXC_PTF14_TPM7_CH0

IOMUXC_PTF14_FB_AD26

IOMUXC_PTF14_USB1_DATA2

IOMUXC_PTF14_VIU_D10

IOMUXC_PTF15_PTF15

IOMUXC_PTF15_FXIO1_D11

IOMUXC_PTF15_LPUART7_RX

IOMUXC_PTF15_TPM7_CH1

IOMUXC_PTF15_FB_AD27

IOMUXC_PTF15_USB1_DATA3

IOMUXC_PTF15_VIU_D11

IOMUXC_PTF16_PTF16

IOMUXC_PTF16_USB1_DATA4

IOMUXC_PTF16_VIU_D12

IOMUXC_PTF16_FXIO1_D12

IOMUXC_PTF16_LPSPI3_SIN

IOMUXC_PTF16_TPM7_CH2

IOMUXC_PTF16_FB_AD28

IOMUXC_PTF17_PTF17

IOMUXC_PTF17_USB1_DATA5

IOMUXC_PTF17_VIU_D13

IOMUXC_PTF17_FXIO1_D13

IOMUXC_PTF17_LPSPI3_SOUT

IOMUXC_PTF17_TPM6_CLKIN

IOMUXC_PTF17_FB_AD29

IOMUXC_PTF18_PTF18

IOMUXC_PTF18_USB1_DATA6

IOMUXC_PTF18_VIU_D14

IOMUXC_PTF18_FXIO1_D14

IOMUXC_PTF18_LPSPI3_SCK

IOMUXC_PTF18_TPM6_CH0

IOMUXC_PTF18_FB_AD30

IOMUXC_PTF19_PTF19

IOMUXC_PTF19_USB1_DATA7

IOMUXC_PTF19_VIU_D15

IOMUXC_PTF19_FXIO1_D15

IOMUXC_PTF19_LPSPI3_PCS0

IOMUXC_PTF19_TPM6_CH1

IOMUXC_PTF19_FB_AD31

IOMUXC_RESET0_b_RESET0_B

IOMUXC_RESET1_b_RESET1_B

IOMUXC_DDR_DQ0

IOMUXC_DDR_DQ1

IOMUXC_DDR_DQ2

IOMUXC_DDR_DQ3

IOMUXC_DDR_DQ4

IOMUXC_DDR_DQ5

IOMUXC_DDR_DQ6

IOMUXC_DDR_DQ7

IOMUXC_DDR_DQ8

IOMUXC_DDR_DQ9

IOMUXC_DDR_DQ10

IOMUXC_DDR_DQ11

IOMUXC_DDR_DQ12

IOMUXC_DDR_DQ13

IOMUXC_DDR_DQ14

IOMUXC_DDR_DQ15

IOMUXC_DDR_DQ16

IOMUXC_DDR_DQ17

IOMUXC_DDR_DQ18

IOMUXC_DDR_DQ19

IOMUXC_DDR_DQ20

IOMUXC_DDR_DQ21

IOMUXC_DDR_DQ22

IOMUXC_DDR_DQ23

IOMUXC_DDR_DQ24

IOMUXC_DDR_DQ25

IOMUXC_DDR_DQ26

IOMUXC_DDR_DQ27

IOMUXC_DDR_DQ28

IOMUXC_DDR_DQ29

IOMUXC_DDR_DQ30

IOMUXC_DDR_DQ31

IOMUXC_DDR_DQS0

IOMUXC_DDR_DQS1

IOMUXC_DDR_DQS2

IOMUXC_DDR_DQS3

IOMUXC_DDR_DQM0

IOMUXC_DDR_DQM1

IOMUXC_DDR_DQM2

IOMUXC_DDR_DQM3

IOMUXC_DDR_CA0

IOMUXC_DDR_CA1

IOMUXC_DDR_CA2

IOMUXC_DDR_CA3

IOMUXC_DDR_CA4

IOMUXC_DDR_CA5

IOMUXC_DDR_CA6

IOMUXC_DDR_CA7

IOMUXC_DDR_CA8

IOMUXC_DDR_CA9

IOMUXC_DDR_CS0_B

IOMUXC_DDR_CS1_B

IOMUXC_DDR_CKE0

IOMUXC_DDR_CKE1

IOMUXC_DDR_CLK0

IOMUXC_DDR_ODT

IOMUXC_DDR_ZQ0

IOMUXC_DDR_ZQ1

IOMUXC_HSIC_DATA

IOMUXC_HSIC_STROBE

FSL_COMPONENT_ID

2.22 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.

UINTPTR_SIZE

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

MCUX_MASK_INVERT_8(mask)
: 8-bit mask inversion.

MCUX_MASK_INVERT_16(mask)
: 16-bit mask inversion.

MCUX_MASK_INVERT_32(mask)
: 32-bit mask inversion for completeness.

MCUX_REG_WRITE8(reg, value)
: 8-bit register write macro

MCUX_REG_WRITE16(reg, value)
: 16-bit register write macro

MCUX_REG_WRITE32(reg, value)
: 32-bit register write macro

MCUX_REG_READ8(reg)
: 8-bit register read macro

MCUX_REG_READ16(reg)
: 16-bit register read macro

MCUX_REG_READ32(reg)
: 32-bit register read macro

MCUX_REG_BIT_SET8(reg, mask)
: 8-bit register bit set macro

MCUX_REG_BIT_SET16(reg, mask)
: 16-bit register bit set macro

MCUX_REG_BIT_SET32(reg, mask)
: 32-bit register bit set macro

MCUX_REG_BIT_CLEAR8(reg, mask)
: 8-bit register bit clear macro

MCUX_REG_BIT_CLEAR16(reg, mask)
: 16-bit register bit clear macro

MCUX_REG_BIT_CLEAR32(reg, mask)
: 32-bit register bit clear macro

MCUX_REG_BIT_GET8(reg, mask)
: 8-bit register bit get macro

MCUX_REG_BIT_GET16(reg, mask)
: 16-bit register bit get macro

MCUX_REG_BIT_GET32(reg, mask)
: 32-bit register bit get macro

MCUX_REG_MODIFY8(reg, mask, value)
: 32-bit register read-modify-write macro

MCUX_REG_MODIFY16(reg, mask, value)
: 16-bit register read-modify-write macro

MCUX_REG_MODIFY32(reg, mask, value)
: 32-bit register read-modify-write macro

SDK_ATOMIC_LOCAL_ADD(addr, val)
: Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)
: Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)
: Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)
: Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)
: Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)
: For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_L1DCACHE_ALIGN(var)

Macro to define a variable with L1 d-cache line size alignment

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value (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.

CACHE_LINE_DATA

AT_QUICKACCESS_SECTION_CODE(func)
Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)
Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)
Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

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.

MCUX_DEPRECATED
Deprecated APIs.

MCUX_DEPRECATED_MACRO
Deprecated macros.

MCUX_EXPERIMENTAL
Experimental APIs.

MCUX_EXPERIMENTAL_MACRO
Experimental macros.

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.

enumerator kStatusGroup_EVENT_CTRL
Group number for Event controller 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.

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.

2.23 LLWU: Low-Leakage Wakeup Unit Driver

static inline void LLWU_GetVersionId(LLWU_Type *base, *llwu_version_id_t* *versionId)
Gets the LLWU version ID.

This function gets the LLWU version ID, including the major version number, the minor version number, and the feature specification number.

Parameters

- base – LLWU peripheral base address.
- versionId – A pointer to the version ID structure.

static inline void LLWU_GetParam(LLWU_Type *base, *llwu_param_t* *param)
Gets the LLWU parameter.

This function gets the LLWU parameter, including a wakeup pin number, a module number, a DMA number, and a pin filter number.

Parameters

- base – LLWU peripheral base address.
- param – A pointer to the LLWU parameter structure.

void LLWU_SetExternalWakeupPinMode(LLWU_Type *base, uint32_t pinIndex, *llwu_external_pin_mode_t* pinMode)

Sets the external input pin source mode.

This function sets the external input pin source mode that is used as a wake up source.

Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index to be enabled as an external wakeup source starting from 1.
- pinMode – A pin configuration mode defined in the llwu_external_pin_modes_t.

bool LLWU_GetExternalWakeupPinFlag(LLWU_Type *base, uint32_t pinIndex)

Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index, which starts from 1.

Returns

True if the specific pin is a wakeup source.

void LLWU_ClearExternalWakeupPinFlag(LLWU_Type *base, uint32_t pinIndex)

Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index, which starts from 1.

static inline void LLWU_EnableInternalModuleInterruptWakup(LLWU_Type *base, uint32_t moduleIndex, bool enable)

Enables/disables the internal module source.

This function enables/disables the internal module source mode that is used as a wake up source.

Parameters

- base – LLWU peripheral base address.
- moduleIndex – A module index to be enabled as an internal wakeup source starting from 1.
- enable – An enable or a disable setting

static inline void LLWU_EnableInternalModuleDmaRequestWakup(LLWU_Type *base, uint32_t moduleIndex, bool enable)

Enables/disables the internal module DMA wakeup source.

This function enables/disables the internal DMA that is used as a wake up source.

Parameters

- base – LLWU peripheral base address.
- moduleIndex – An internal module index which is used as a DMA request source, starting from 1.

- enable – Enable or disable the DMA request source

void LLWU_SetPinFilterMode(LLWU_Type *base, uint32_t filterIndex, *llwu_external_pin_filter_mode_t* filterMode)

Sets the pin filter configuration.

This function sets the pin filter configuration.

Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index used to enable/disable the digital filter, starting from 1.
- filterMode – A filter mode configuration

bool LLWU_GetPinFilterFlag(LLWU_Type *base, uint32_t filterIndex)

Gets the pin filter configuration.

This function gets the pin filter flag.

Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

void LLWU_ClearPinFilterFlag(LLWU_Type *base, uint32_t filterIndex)

Clears the pin filter configuration.

This function clears the pin filter flag.

Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

void LLWU_SetResetPinMode(LLWU_Type *base, bool pinEnable, bool pinFilterEnable)

Sets the reset pin mode.

This function determines how the reset pin is used as a low leakage mode exit source.

Parameters

- base – LLWU peripheral base address.
- pinEnable – Enable reset the pin filter
- pinFilterEnable – Specify whether the pin filter is enabled in Low-Leakage power mode.

FSL_LLWU_DRIVER_VERSION

LLWU driver version.

enum _llwu_external_pin_mode

External input pin control modes.

Values:

enumerator kLLWU_ExternalPinDisable

Pin disabled as a wakeup input.

enumerator kLLWU_ExternalPinRisingEdge

Pin enabled with the rising edge detection.

enumerator kLLWU_ExternalPinFallingEdge
Pin enabled with the falling edge detection.

enumerator kLLWU_ExternalPinAnyEdge
Pin enabled with any change detection.

enum _llwu_pin_filter_mode
Digital filter control modes.

Values:

enumerator kLLWU_PinFilterDisable
Filter disabled.

enumerator kLLWU_PinFilterRisingEdge
Filter positive edge detection.

enumerator kLLWU_PinFilterFallingEdge
Filter negative edge detection.

enumerator kLLWU_PinFilterAnyEdge
Filter any edge detection.

typedef enum *_llwu_external_pin_mode* llwu_external_pin_mode_t
External input pin control modes.

typedef enum *_llwu_pin_filter_mode* llwu_pin_filter_mode_t
Digital filter control modes.

typedef struct *_llwu_version_id* llwu_version_id_t
IP version ID definition.

typedef struct *_llwu_param* llwu_param_t
IP parameter definition.

typedef struct *_llwu_external_pin_filter_mode* llwu_external_pin_filter_mode_t
An external input pin filter control structure.

LLWU_REG_VAL(x)

struct _llwu_version_id
#include <fsl_llwu.h> IP version ID definition.

Public Members

uint16_t feature
A feature specification number.

uint8_t minor
The minor version number.

uint8_t major
The major version number.

struct _llwu_param
#include <fsl_llwu.h> IP parameter definition.

Public Members

uint8_t filters
: A number of the pin filter.

uint8_t dmas
: A number of the wakeup DMA.

uint8_t modules
: A number of the wakeup module.

uint8_t pins
: A number of the wake up pin.

struct __llwu_external_pin_filter_mode
: *#include <fsl_llwu.h>* An external input pin filter control structure.

Public Members

uint32_t pinIndex
: A pin number

llwu_pin_filter_mode_t filterMode
: Filter mode

2.24 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum __lpadc_status_flags
: Define hardware flags of the module.

Values:

enumerator kLPADC_ResultFIFO0OverflowFlag
: Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC_ResultFIFO0ReadyFlag
: Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC_TriggerExceptionFlag
: Indicates that a trigger exception event has occurred.

enumerator kLPADC_TriggerCompletionFlag
: Indicates that a trigger completion event has occurred.

enumerator kLPADC_CalibrationReadyFlag
: Indicates that the calibration process is done.

enumerator kLPADC_ActiveFlag
: Indicates that the ADC is in active state.

enumerator kLPADC_ResultFIFOOverflowFlag
: To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowFlag as instead.

enumerator kLPADC_ResultFIFOReadyFlag
: To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0ReadyFlag as instead.

enum _lpadc_interrupt_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowInterruptEnable as instead.

enumerator kLPADC_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_FIFO0WatermarkInterruptEnable as instead.

enumerator kLPADC_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kLPADC_Trigger2CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 2 completion.

enumerator kLPADC_Trigger3CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 3 completion.

enumerator kLPADC_Trigger4CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 4 completion.

enumerator kLPADC_Trigger5CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 5 completion.

enumerator kLPADC_Trigger6CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 6 completion.

enumerator kLPADC_Trigger7CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 7 completion.

enumerator kLPADC_Trigger8CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 8 completion.

enumerator kLPADC_Trigger9CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 9 completion.

enumerator kLPADC_Trigger10CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 10 completion.

enumerator kLPADC_Trigger11CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 11 completion.

enumerator kLPADC_Trigger12CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 12 completion.

enumerator kLPADC_Trigger13CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 13 completion.

enumerator kLPADC_Trigger14CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 14 completion.

enumerator kLPADC_Trigger15CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 15 completion.

enum _lpadc_trigger_status_flags
The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_Trigger0InterruptedFlag
Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC_Trigger1InterruptedFlag
Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC_Trigger2InterruptedFlag
Trigger 2 is interrupted by a high priority exception.

enumerator kLPADC_Trigger3InterruptedFlag
Trigger 3 is interrupted by a high priority exception.

enumerator kLPADC_Trigger4InterruptedFlag
Trigger 4 is interrupted by a high priority exception.

enumerator kLPADC_Trigger5InterruptedFlag
Trigger 5 is interrupted by a high priority exception.

enumerator kLPADC_Trigger6InterruptedFlag
Trigger 6 is interrupted by a high priority exception.

enumerator kLPADC_Trigger7InterruptedFlag
Trigger 7 is interrupted by a high priority exception.

enumerator kLPADC_Trigger8InterruptedFlag
Trigger 8 is interrupted by a high priority exception.

enumerator kLPADC_Trigger9InterruptedFlag
Trigger 9 is interrupted by a high priority exception.

enumerator kLPADC_Trigger10InterruptedFlag
Trigger 10 is interrupted by a high priority exception.

enumerator kLPADC_Trigger11InterruptedFlag
Trigger 11 is interrupted by a high priority exception.

enumerator kLPADC_Trigger12InterruptedFlag
Trigger 12 is interrupted by a high priority exception.

enumerator kLPADC_Trigger13InterruptedFlag
Trigger 13 is interrupted by a high priority exception.

enumerator kLPADC_Trigger14InterruptedFlag
Trigger 14 is interrupted by a high priority exception.

enumerator kLPADC_Trigger15InterruptedFlag
Trigger 15 is interrupted by a high priority exception.

enumerator kLPADC_Trigger0CompletedFlag
Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC_Trigger1CompletedFlag
Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kLPADC_Trigger2CompletedFlag
Trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kLPADC_Trigger3CompletedFlag
Trigger 3 is completed and trigger 3 has enabled completion interrupts.

enumerator kLPADC_Trigger4CompletedFlag
Trigger 4 is completed and trigger 4 has enabled completion interrupts.

enumerator kLPADC_Trigger5CompletedFlag
Trigger 5 is completed and trigger 5 has enabled completion interrupts.

enumerator kLPADC_Trigger6CompletedFlag
Trigger 6 is completed and trigger 6 has enabled completion interrupts.

enumerator kLPADC_Trigger7CompletedFlag
Trigger 7 is completed and trigger 7 has enabled completion interrupts.

enumerator kLPADC_Trigger8CompletedFlag
Trigger 8 is completed and trigger 8 has enabled completion interrupts.

enumerator kLPADC_Trigger9CompletedFlag
Trigger 9 is completed and trigger 9 has enabled completion interrupts.

enumerator kLPADC_Trigger10CompletedFlag
Trigger 10 is completed and trigger 10 has enabled completion interrupts.

enumerator kLPADC_Trigger11CompletedFlag
Trigger 11 is completed and trigger 11 has enabled completion interrupts.

enumerator kLPADC_Trigger12CompletedFlag
Trigger 12 is completed and trigger 12 has enabled completion interrupts.

enumerator kLPADC_Trigger13CompletedFlag
Trigger 13 is completed and trigger 13 has enabled completion interrupts.

enumerator kLPADC_Trigger14CompletedFlag
Trigger 14 is completed and trigger 14 has enabled completion interrupts.

enumerator kLPADC_Trigger15CompletedFlag
Trigger 15 is completed and trigger 15 has enabled completion interrupts.

enum __lpadc_sample_scale_mode
Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Values:

enumerator kLPADC_SamplePartScale
Use divided input voltage signal. (For scale select,please refer to the reference manual).

enumerator kLPADC_SampleFullScale
Full scale (Factor of 1).

enum _lpadc_sample_channel_mode
Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Values:

enumerator kLPADC_SampleChannelSingleEndSideA
Single-end mode, only A-side channel is converted.

enumerator kLPADC_SampleChannelSingleEndSideB
Single-end mode, only B-side channel is converted.

enumerator kLPADC_SampleChannelDiffBothSideAB
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDiffBothSideBA
Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC_SampleChannelDiffBothSide
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDualSingleEndBothSide
Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum _lpadc_hardware_average_mode
Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Values:

enumerator kLPADC_HardwareAverageCount1
Single conversion.

enumerator kLPADC_HardwareAverageCount2
2 conversions averaged.

enumerator kLPADC_HardwareAverageCount4
4 conversions averaged.

enumerator kLPADC_HardwareAverageCount8
8 conversions averaged.

enumerator kLPADC_HardwareAverageCount16
16 conversions averaged.

enumerator kLPADC_HardwareAverageCount32
32 conversions averaged.

enumerator kLPADC_HardwareAverageCount64
64 conversions averaged.

enumerator kLPADC_HardwareAverageCount128
128 conversions averaged.

enum _lpadc_sample_time_mode
Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Values:

enumerator kLPADC_SampleTimeADCK3
3 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK5
5 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK7
7 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK11
11 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK19
19 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK35
35 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK67
69 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK131
131 ADCK cycles total sample time.

enum _lpadc_hardware_compare_mode
Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Values:

enumerator kLPADC_HardwareCompareDisabled
Compare disabled.

enumerator kLPADC_HardwareCompareStoreOnTrue
Compare enabled. Store on true.

enumerator kLPADC_HardwareCompareRepeatUntilTrue
Compare enabled. Repeat channel acquisition until true.

enum _lpadc_conversion_resolution_mode
Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc_sample_channel_mode_t

Values:

enumerator kLPADC_ConversionResolutionStandard

Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator kLPADC_ConversionResolutionHigh

High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum _lpadc_conversion_average_mode

Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

Values:

enumerator kLPADC_ConversionAverage1

Single conversion.

enumerator kLPADC_ConversionAverage2

2 conversions averaged.

enumerator kLPADC_ConversionAverage4

4 conversions averaged.

enumerator kLPADC_ConversionAverage8

8 conversions averaged.

enumerator kLPADC_ConversionAverage16

16 conversions averaged.

enumerator kLPADC_ConversionAverage32

32 conversions averaged.

enumerator kLPADC_ConversionAverage64

64 conversions averaged.

enumerator kLPADC_ConversionAverage128

128 conversions averaged.

enumerator kLPADC_ConversionAverageMax

enum _lpadc_reference_voltage_mode

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

Values:

enumerator kLPADC_ReferenceVoltageAlt1

Option 1 setting.

enumerator kLPADC_ReferenceVoltageAlt2

Option 2 setting.

enumerator kLPADC_ReferenceVoltageAlt3

Option 3 setting.

enum _lpadc_power_level_mode

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Values:

enumerator kLPADC_PowerLevelAlt1

Lowest power setting.

enumerator kLPADC_PowerLevelAlt2

Next lowest power setting.

enumerator kLPADC_PowerLevelAlt3

...

enumerator kLPADC_PowerLevelAlt4

Highest power setting.

enum _lpadc_offset_calibration_mode

Define enumeration of offset calibration mode.

Values:

enumerator kLPADC_OffsetCalibration12bitMode

12 bit offset calibration mode.

enumerator kLPADC_OffsetCalibration16bitMode

16 bit offset calibration mode.

enum _lpadc_trigger_priority_policy

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

Values:

enumerator kLPADC_ConvPreemptImmediatelyNotAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator kLPADC_ConvPreemptSoftlyNotAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoRestarted

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptSoftlyAutoRestarted

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator kLPADC_ConvPreemptSoftlyAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator kLPADC_TriggerPriorityPreemptImmediately

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityPreemptSoftly

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityExceptionDisabled

High priority trigger exception disabled.

enum _lpadc_tune_value

Define enumeration of tune value.

Values:

enumerator kLPADC_TuneValue0

Tune value 0.

enumerator kLPADC_TuneValue1

Tune value 1.

enumerator kLPADC_TuneValue2

Tune value 2.

enumerator kLPADC_TuneValue3

Tune value 3.

typedef enum *_lpadc_sample_scale_mode* lpadc_sample_scale_mode_t

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

typedef enum *_lpadc_sample_channel_mode* lpadc_sample_channel_mode_t

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

typedef enum *_lpadc_hardware_average_mode* lpadc_hardware_average_mode_t

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

typedef enum *_lpadc_sample_time_mode* lpadc_sample_time_mode_t

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

typedef enum *_lpadc_hardware_compare_mode* lpadc_hardware_compare_mode_t

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

typedef enum *_lpadc_conversion_resolution_mode* lpadc_conversion_resolution_mode_t

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc_sample_channel_mode_t

typedef enum *_lpadc_conversion_average_mode* lpadc_conversion_average_mode_t

Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

---

typedef enum *_lpadc_reference_voltage_mode* lpadc_reference_voltage_source_t

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

typedef enum *_lpadc_power_level_mode* lpadc_power_level_mode_t

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

typedef enum *_lpadc_offset_calibration_mode* lpadc_offset_calibration_mode_t

Define enumeration of offset calibration mode.

typedef enum *_lpadc_trigger_priority_policy* lpadc_trigger_priority_policy_t

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

typedef enum *_lpadc_tune_value* lpadc_tune_value_t

Define enumeration of tune value.

typedef struct *_lpadc_calibration_value* lpadc_calibration_value_t

A structure of calibration value.

LPADC_CONVERSION_COMPLETE_TIMEOUT

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

LPADC_CALIBRATION_READY_TIMEOUT

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

LPADC_GAIN_CAL_READY_TIMEOUT

Max loops to wait for LPADC gain calibration GAIN_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN_CAL ready. This parameter defines how many loops to check the gain calibration GAIN_CAL ready. If defined as 0, driver will wait forever until ready.

ADC_OFSTRIM_OFSTRIM_MAX

ADC_OFSTRIM_OFSTRIM_SIGN

LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

void LPADC_Init(ADC_Type *base, const *lpadc_config_t* *config)

Initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See "lpadc_config_t".

void LPADC_GetDefaultConfig(*lpadc_config_t* *config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode        = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay            = 0x80;
config->referenceVoltageSource  = kLPADC_ReferenceVoltageAlt1;
```

(continues on next page)

(continued from previous page)

```
config->powerLevelMode          = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause         = false;
config->convPauseDelay          = 0U;
config->FIFOWatermark           = 0U;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_Deinit(ADC_Type *base)

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_Enable(ADC_Type *base, bool enable)

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC_DoResetFIFO(ADC_Type *base)

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_DoResetConfig(ADC_Type *base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns

status flags' mask. See to _lpadc_status_flags.

static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clear status flags.

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

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to _lpadc_status_flags.

static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Note: On some devices, the trigger completion status may be asserted before the final command in a chained trigger sequence starts to execute. When using chained commands, do not rely on trigger completion status alone to guarantee that all conversion results are already available in the FIFO. Use FIFO ready indication together with result tags, or stall the last command with WAIT_TRIG when that sequencing model is acceptable for the application.

Parameters

- base – LPADC peripheral base address.

Returns

The OR'ed value of _lpadc_trigger_status_flags.

static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of _lpadc_trigger_status_flags.

static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)

Enable interrupts.

Note: When enabling trigger completion interrupts (kLPADC_TriggerXCompletionInterruptEnable) on some devices, the interrupt may occur before the final command in a chained trigger sequence starts to execute. For multi-command trigger sequences, do not use the trigger completion interrupt alone as the indication that all expected results are already stored in the FIFO.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to _lpadc_interrupt_enable.

static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to _lpadc_interrupt_enable.

static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns

The count of result kept in conversion FIFO.

bool LPADC_GetConvResult(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns

Status whether FIFO entry is valid.

void LPADC_GetConvResultBlocking(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const *lpadc_conv_trigger_config_t* *config)

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to lpadc_conv_trigger_config_t.

void LPADC_GetDefaultConvTriggerConfig(*lpadc_conv_trigger_config_t* *config)

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

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId       = 0U;
config->delayPower            = 0U;
config->priority              = 0U;
config->channelAFIFOSelect    = 0U;
config->channelBFIFOSelect    = 0U;
config->enableHardwareTrigger = false;
```

Parameters

- config – Pointer to configuration structure.

static inline void LPADC_DoSoftwareTrigger(ADC_Type *base, uint32_t triggerIdMask)

Do software trigger to conversion command.

Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

static inline void LPADC_EnableHardwareTriggerCommandSelection(ADC_Type *base, uint32_t triggerId, bool enable)

Enable hardware trigger command selection.

This function will use the hardware trigger command from ADC_ETC.The trigger command is then defined by ADC hardware trigger command selection field in ADC_ETC->TRIGx_CHAINy_z_n[CSEL].

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- enable – True to enable or flase to disable.

void LPADC_SetConvCommandConfig(ADC_Type *base, uint32_t commandId, const *lpadc_conv_command_config_t* *config)

Configure conversion command.

Note: The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

Parameters

- base – LPADC peripheral base address.
- commandId – ID for command in command buffer. Typically, the available value range is 1 - 15.
- config – Pointer to configuration structure. See to lpadc_conv_command_config_t.

void LPADC_GetDefaultConvCommandConfig(*lpadc_conv_command_config_t* *config)

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

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode             = kLPADC_SampleFullScale;
config->channelBScaleMode           = kLPADC_SampleFullScale;
config->sampleChannelMode           = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber               = 0U;
config->channelBNumber              = 0U;
config->chainedNextCommandNumber    = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount                   = 0U;
config->hardwareAverageMode         = kLPADC_HardwareAverageCount1;
config->sampleTimeMode              = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode         = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh    = 0U;
config->hardwareCompareValueLow     = 0U;
```

(continues on next page)

(continued from previous page)

```
config->conversionResolutionMode   = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger          = false;
config->enableChannelB             = false;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_EnableCalibration(ADC_Type *base, bool enable)

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6- bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

Parameters

- base – LPADC peripheral base address.
- value – Setting offset value.

status_t LPADC_DoAutoCalibration(ADC_Type *base)

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including:

- LPADC_EnableCalibration(...)
- LPADC_SetOffsetValue(...)
- LPADC_SetConvCommandConfig(...)
- LPADC_SetConvTriggerConfig(...)

Parameters

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_SetOffsetValue(ADC_Type *base, int16_t value)

Set trim value for offset.

Note: For 16-bit conversions, each increment is 1/2 LSB resulting in a programmable offset range of -256 LSB to 255.5 LSB; For 12-bit conversions, each increment is 1/32 LSB resulting in a programmable offset range of -16 LSB to 15.96875 LSB.

Parameters

- base – LPADC peripheral base address.
- value – Offset trim value, is a 10-bit signed value between -512 and 511.

static inline void LPADC_GetOffsetValue(ADC_Type *base, int16_t *pValue)

Get trim value of offset.

Parameters

- base – LPADC peripheral base address.
- pValue – Pointer to the variable in type of int16_t to store offset value.

static inline void LPADC_EnableOffsetCalibration(ADC_Type *base, bool enable)

Enable the offset calibration function.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetCalibrationMode(ADC_Type *base,
lpadc_offset_calibration_mode_t mode)

Set offset calibration mode.

Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to lpadc_offset_calibration_mode_t .

status_t LPADC_DoOffsetCalibration(ADC_Type *base)

Do offset calibration.

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_PrepareAutoCalibration(ADC_Type *base)

Prepare auto calibration, LPADC_FinishAutoCalibration has to be called before using the LPADC. LPADC_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

- base – LPADC peripheral base address.

status_t LPADC_FinishAutoCalibration(ADC_Type *base)

Finish auto calibration start with LPADC_PrepareAutoCalibration.

Note: This feature is used for LPADC with CTRL[CALOFSMODE].

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_GetCalibrationValue(ADC_Type *base, *lpadc_calibration_value_t* *ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

Note: Please note the ADC will be disabled temporary.

Note: This function should be used after finish calibration.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure, this memory block should be always powered on even in low power modes.

status_t LPADC_SetCalibrationValue(ADC_Type *base, const *lpadc_calibration_value_t* *ptrCalibrationValue)

Set calibration value into ADC calibration registers.

Note: Please note the ADC will be disabled temporary.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure which contains ADC's calibration value.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_RequestHighSpeedModeTrim(ADC_Type *base)

Request high speed mode trim calculation.

Parameters

- base – LPADC peripheral base address.

static inline int8_t LPADC_GetHighSpeedTrimValue(ADC_Type *base)

Get high speed mode trim value, the result is a 5-bit signed value between -16 and 15.

Note: The high speed mode trim value is used to minimize offset for high speed conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The calculated high speed mode trim value.

static inline void LPADC_SetHighSpeedTrimValue(ADC_Type *base, int8_t trimValue)

Set high speed mode trim value.

Note: If is possible to set the trim value manually, but it is recommended to use the LPADC_RequestHighSpeedModeTrim.

Parameters

- base – LPADC peripheral base address.
- trimValue – The trim value to be set.

static inline void LPADC_EnableHighSpeedConversionMode(ADC_Type *base, bool enable)

Enable/disable high speed conversion mode, if enabled conversions complete 2 or 3 ADCK cycles sooner compared to conversion cycle counts when high speed mode is disabled.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable high speed conversion mode:
  - **true** Enable high speed conversion mode;
  - **false** Disable high speed conversion mode.

static inline void LPADC_EnableExtraCycle(ADC_Type *base, bool enable)

Enable/disable an additional ADCK cycle to conversion.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable an additional ADCK cycle to conversion:
  - **true** Enable an additional ADCK cycle to conversion;
  - **false** Disable an additional ADCK cycle to conversion.

static inline void LPADC_SetTuneValue(ADC_Type *base, *lpadc_tune_value_t* tuneValue)

Set tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.
- tuneValue – The tune value to be set, please refer to lpadc_tune_value_t.

static inline *lpadc_tune_value_t* LPADC_GetTuneValue(ADC_Type *base)

Get tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The tune value, please refer to lpadc_tune_value_t.

static inline void LPADC_EnableJustifiedLeft(ADC_Type *base, bool enable)

Enable/disable left-justify format in 12-bit single-end mode.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable left-justify format in 12-bit single-end mode:
  - **true** Enable left-justify format in 12-bit single-end mode;
  - **false** Disable left-justify format in 12-bit single-end mode.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.10.0.

struct lpadc_config_t

#include <fsl_lpadc.h> LPADC global configuration.

This structure would used to keep the settings for initialization.

Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay * 4) ADCK cycles must result in a longer delay than the analog startup time.

lpadc_reference_voltage_source_t referenceVoltageSource

Selects the voltage reference high used for conversions.

lpadc_power_level_mode_t powerLevelMode

Power Configuration Selection.

lpadc_trigger_priority_policy_t triggerPriorityPolicy

Control how higher priority triggers are handled, see to lpadc_trigger_priority_policy_t.

bool enableConvPause

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in "Compare Until True" configuration.

uint32_t convPauseDelay

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (convPauseDelay*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

uint32_t FIFOWatermark

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

struct lpadc_conv_command_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion command.

Public Members

lpadc_sample_scale_mode_t sampleScaleMode

Sample scale mode.

lpadc_sample_scale_mode_t channelBScaleMode

Alternate channe B Scale mode.

lpadc_sample_channel_mode_t sampleChannelMode

Channel sample mode.

uint32_t channelNumber

Channel number, select the channel or channel pair.

uint32_t channelBNumber

Alternate Channel B number, select the channel.

uint32_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.

uint32_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

lpadc_hardware_average_mode_t hardwareAverageMode

Hardware average selection.

lpadc_sample_time_mode_t sampleTimeMode

Sample time selection.

lpadc_hardware_compare_mode_t hardwareCompareMode

Hardware compare selection.

uint32_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

lpadc_conversion_resolution_mode_t conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

struct lpadc_conv_trigger_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion trigger.

Public Members

uint32_t targetCommandId

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

uint32_t delayPower

Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is 2^delayPower ADCK cycles. The available value range is 4-bit.

uint32_t priority

Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

bool enableHardwareTrigger

Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. THe software trigger is always available.

struct lpadc_conv_result_t

#include <fsl_lpadc.h> Define the structure to keep the conversion result.

Public Members

uint32_t commandIdSource

Indicate the command buffer being executed that generated this result.

uint32_t loopCountIndex

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

uint32_t triggerIdSource

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

uint16_t convValue
Data result.

struct _lpadc_calibration_value
#include <fsl_lpadc.h> A structure of calibration value.

2.25 LPI2C: Low Power Inter-Integrated Circuit Driver

void LPI2C_DriverIRQHandler(uint32_t instance)
LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

Parameters

- instance – LPI2C instance.

FSL_LPI2C_DRIVER_VERSION
LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy
The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle
The slave driver is idle.

enumerator kStatus_LPI2C_Nak
The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError
FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError
Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost
Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout
SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress
Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail
DMA request failed.

enumerator kStatus_LPI2C_Timeout
Timeout polling status flags.

IRQn_Type const kLpi2cMasterIrqs[]
Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

IRQn_Type const kLpi2cSlaveIrqs[]

lpi2c_master_isr_t s_lpi2cMasterIsr

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

void *s_lpi2cMasterHandle[]

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

uint32_t LPI2C_GetInstance(LPI2C_Type *base)

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- base – The LPI2C peripheral base address.

Returns

LPI2C instance number starting from 0.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.26 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster            = true;
masterConfig->debugEnable             = false;
masterConfig->ignoreAck               = false;
masterConfig->pinConfig               = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz             = 100000U;
masterConfig->busIdleTimeout_ns       = 0;
masterConfig->pinLowTimeout_ns        = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable      = false;
masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to lpi2c_master_config_t.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.

- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

Convert provided flags to status code, and clear any errors if present.

Parameters

- base – The LPI2C peripheral base address.
- status – Current status flags value that will be checked.

Return values

- kStatus_Success –
- kStatus_LPI2C_PinLowTimeout –
- kStatus_LPI2C_ArbitrationLost –
- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags().

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted,

followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_LPI2C_Nak.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.

- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag
Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag
End Packet flag

enumerator kLPI2C_MasterStopDetectFlag
Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag
NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag
Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag
FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag
Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag
Data match flag

enumerator kLPI2C_MasterBusyFlag
Master busy flag

enumerator kLPI2C_MasterBusBusyFlag
Bus busy flag

enumerator kLPI2C_MasterClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
Errors to check for.

enum _lpi2c_direction
Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write
Master transmit.

enumerator kLPI2C_Read
Master receive.

enum _lpi2c_master_pin_config
LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain
LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Values:

enumerator kLPI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferNoStopFlag

Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t

Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t

LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t

LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t

LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t

LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t

LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

LPI2C master descriptor of the transfer.

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

LPI2C master handle of the transfer.

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base
The LPI2C peripheral base address.

Param handle
Pointer to the LPI2C master driver handle.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct _lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest

Host request options.

struct _lpi2c_match_config

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
Data match configuration setting.

bool rxDataMatchOnly
When set to true, received data is ignored until a successful match.

uint32_t match0
Match value 0.

uint32_t match1
Match value 1.

struct __lpi2c_master_transfer
#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
Bit mask of options for the transfer. See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefaultFlag for normal transfers.

uint16_t slaveAddress
The 7-bit slave address.

lpi2c_direction_t direction
Either kLPI2C_Read or kLPI2C_Write.

uint32_t subaddress
Sub address. Transferred MSB first.

size_t subaddressSize
Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
Pointer to data to transfer.

size_t dataSize
Number of bytes to transfer.

struct __lpi2c_master_handle
#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
Transfer state machine current state.

uint16_t remainingBytes
Remaining byte count in current state.

uint8_t *buf
Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

uint16_t chunkSize

Remaining byte count in current chunk.

struct hostRequest

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.27 LPI2C Master DMA Driver

void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle, *lpi2c_master_edma_transfer_callback_t* callback, void *userData)

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

typedef struct *_lpi2c_master_edma_handle* lpi2c_master_edma_handle_t

LPI2C master EDMA handle of the transfer.

typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *status_t* completionStatus, void *userData)

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Param base
The LPI2C peripheral base address.

Param handle
Handle associated with the completed transfer.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

struct __lpi2c_master_edma_handle

#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base

LPI2C base pointer.

bool isBusy

Transfer state machine current state.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[20]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

edma_handle_t *rx

Handle for receive DMA channel.

edma_handle_t *tx

Handle for transmit DMA channel.

edma_tcd_t tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.28 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave              = true;
slaveConfig->address0                 = 0U;
slaveConfig->address1                 = 0U;
slaveConfig->addressMatchMode         = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable         = true;
slaveConfig->filterEnable             = true;
slaveConfig->enableGeneralCall        = false;
slaveConfig->sclStall.enableAck       = false;
slaveConfig->sclStall.enableTx        = true;
slaveConfig->sclStall.enableRx        = true;
slaveConfig->sclStall.enableAddress   = true;
slaveConfig->ignoreAck                = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns  = 0;
slaveConfig->sclGlitchFilterWidth_ns  = 0;
slaveConfig->dataValidDelay_ns        = 0;
slaveConfig->clockHoldTime_ns         = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t.

void LPI2C_SlaveInit(LPI2C_Type *base, const *lpi2c_slave_config_t* *slaveConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use LPI2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_slave_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_slave_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags().

static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.

- interruptMask – Bit mask of interrupts to enable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the kLPI2C_SlaveTransmitAckFlag is asserted. This only happens if you enable the sclStall.enableAck field of the lpi2c_slave_config_t configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

- ackOrNack – Pass true for an ACK or false for a NAK.

static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL,false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns

Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns

Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of lpi2c_slave_transfer_event_t enumerators for the events you wish to receive. The kLPI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kLPI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together lpi2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kLPI2C_SlaveAllEvents to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_LPI2C_Busy – Slave transfers have already been started on this handle.

status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag

Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag

Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag
Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag
Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag
Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag
FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag
Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag
Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag
General call flag

enumerator kLPI2C_SlaveBusyFlag
Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag
Bus busy flag

enumerator kLPI2C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags
Errors to check for.

enum _lpi2c_slave_address_match
LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0
Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1
Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1
Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent
Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent
Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent
Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent
Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C_SlaveRepeatedStartEvent
A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent
A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents
Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t
LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t
Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t
LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t
LPI2C slave handle structure.

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)
Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the LPI2C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct __lpi2c_slave_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave
Enable slave mode.

uint8_t address0
Slave's 7-bit address.

uint8_t address1
Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode
Address matching options.

bool filterDozeEnable
Enable digital glitch filter in doze mode.

bool filterEnable
Enable digital glitch filter.

bool enableGeneralCall
Enable general call address matching.

struct *_lpi2c_slave_config* sclStall
SCL stall enable options.

bool ignoreAck
Continue transfers after a NACK is detected.

bool enableReceivedAddressRead
Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns
Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns
Width in nanoseconds of the clock hold time.

struct __lpi2c_slave_transfer

#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event
Reason the callback is being invoked.

uint8_t receivedAddress
Matching address send by master.

uint8_t *data
Transfer buffer

size_t dataSize
Transfer size

status_t completionStatus
Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount
Number of bytes actually transferred since start or last repeated start.

struct __lpi2c_slave_handle
#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer
LPI2C slave transfer copy.

bool isBusy
Whether transfer is busy.

bool wasTransmit
Whether the last transfer was a transmit.

uint32_t eventMask
Mask of enabled events.

uint32_t transferredCount
Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck
Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

2.29 LPIT: Low-Power Interrupt Timer

void LPIT_Init(LPIT_Type *base, const *lpit_config_t* *config)

Ungates the LPIT clock and configures the peripheral for a basic operation.

This function issues a software reset to reset all channels and registers except the Module Control register.

Note: This API should be called at the beginning of the application using the LPIT driver.

Parameters

- base – LPIT peripheral base address.
- config – Pointer to the user configuration structure.

void LPIT_Deinit(LPIT_Type *base)

Disables the module and gates the LPIT clock.

Parameters

- base – LPIT peripheral base address.

void LPIT_GetDefaultConfig(*lpit_config_t* *config)

Fills in the LPIT configuration structure with default settings.

The default values are:

```
config->enableRunInDebug = false;
config->enableRunInDoze = false;
```

Parameters

- config – Pointer to the user configuration structure.

status_t LPIT_SetupChannel(LPIT_Type *base, *lpit_chnl_t* channel, const *lpit_chnl_params_t* *chnlSetup)

Sets up an LPIT channel based on the user's preference.

This function sets up the operation mode to one of the options available in the enumeration lpit_timer_modes_t. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

- base – LPIT peripheral base address.
- channel – Channel that is being configured.

- chnlSetup – Configuration parameters.

static inline void LPIT_EnableInterrupts(LPIT_Type *base, uint32_t mask)

Enables the selected PIT interrupts.

Parameters

- base – LPIT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration lpit_interrupt_enable_t

static inline void LPIT_DisableInterrupts(LPIT_Type *base, uint32_t mask)

Disables the selected PIT interrupts.

Parameters

- base – LPIT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration lpit_interrupt_enable_t

static inline uint32_t LPIT_GetEnabledInterrupts(LPIT_Type *base)

Gets the enabled LPIT interrupts.

Parameters

- base – LPIT peripheral base address.

Returns

The enabled interrupts. This is the logical OR of members of the enumeration lpit_interrupt_enable_t

static inline uint32_t LPIT_GetStatusFlags(LPIT_Type *base)

Gets the LPIT status flags.

Parameters

- base – LPIT peripheral base address.

Returns

The status flags. This is the logical OR of members of the enumeration lpit_status_flags_t

static inline void LPIT_ClearStatusFlags(LPIT_Type *base, uint32_t mask)

Clears the LPIT status flags.

Parameters

- base – LPIT peripheral base address.
- mask – The status flags to clear. This is a logical OR of members of the enumeration lpit_status_flags_t

static inline void LPIT_SetTimerPeriod(LPIT_Type *base, *lpit_chnl_t* channel, uint32_t ticks)

Sets the timer period in units of count.

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

- base – LPIT peripheral base address.

- channel – Timer channel number.
- ticks – Timer period in units of ticks.

static inline void LPIT_SetTimerValue(LPIT_Type *base, *lpit_chnl_t* channel, uint32_t ticks)

Sets the timer period in units of count.

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

Note: Set TVAL register to 0 or 1 is invalid in compare mode.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.
- ticks – Timer period in units of ticks.

static inline uint32_t LPIT_GetCurrentTimerCount(LPIT_Type *base, *lpit_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in fsl_common.h to convert ticks to microseconds or milliseconds.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

Returns

Current timer counting value in ticks.

static inline void LPIT_StartTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Starts the timer counting.

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

static inline void LPIT_StopTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Stops the timer counting.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

FSL_LPIT_DRIVER_VERSION

Version 2.1.3

enum _lpit_chnl

List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

Values:

enumerator kLPIT_Chnl_0

LPIT channel number 0

enumerator kLPIT_Chnl_1

LPIT channel number 1

enumerator kLPIT_Chnl_2

LPIT channel number 2

enumerator kLPIT_Chnl_3

LPIT channel number 3

enum _lpit_timer_modes

Mode options available for the LPIT timer.

Values:

enumerator kLPIT_PeriodicCounter

Use the all 32-bits, counter loads and decrements to zero

enumerator kLPIT_DualPeriodicCounter

Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement

enumerator kLPIT_TriggerAccumulator

Counter loads on first trigger and decrements on each trigger

enumerator kLPIT_InputCapture

Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

enum _lpit_trigger_select

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Values:

enumerator kLPIT_Trigger_TimerChn0

Channel 0 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn1

Channel 1 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn2

Channel 2 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn3

Channel 3 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn4

Channel 4 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn5

Channel 5 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn6
Channel 6 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn7
Channel 7 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn8
Channel 8 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn9
Channel 9 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn10
Channel 10 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn11
Channel 11 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn12
Channel 12 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn13
Channel 13 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn14
Channel 14 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn15
Channel 15 is selected as a trigger source

enum _lpit_trigger_source
Trigger source options available.

Values:

enumerator kLPIT_TriggerSource_External
Use external trigger input

enumerator kLPIT_TriggerSource_Internal
Use internal trigger

enum _lpit_interrupt_enable
List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerInterruptEnable
Channel 0 Timer interrupt

enumerator kLPIT_Channel1TimerInterruptEnable
Channel 1 Timer interrupt

enumerator kLPIT_Channel2TimerInterruptEnable
Channel 2 Timer interrupt

enumerator kLPIT_Channel3TimerInterruptEnable
Channel 3 Timer interrupt

enum _lpit_status_flags
List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerFlag
Channel 0 Timer interrupt flag

enumerator kLPIT_Channel1TimerFlag
Channel 1 Timer interrupt flag

enumerator kLPIT_Channel2TimerFlag
Channel 2 Timer interrupt flag

enumerator kLPIT_Channel3TimerFlag
Channel 3 Timer interrupt flag

typedef enum *_lpit_chnl* lpit_chnl_t
List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

typedef enum *_lpit_timer_modes* lpit_timer_modes_t
Mode options available for the LPIT timer.

typedef enum *_lpit_trigger_select* lpit_trigger_select_t
Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

typedef enum *_lpit_trigger_source* lpit_trigger_source_t
Trigger source options available.

typedef enum *_lpit_interrupt_enable* lpit_interrupt_enable_t
List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

typedef enum *_lpit_status_flags* lpit_status_flags_t
List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

typedef struct *_lpit_chnl_params* lpit_chnl_params_t
Structure to configure the channel timer.

typedef struct *_lpit_config* lpit_config_t
LPIT configuration structure.

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

The configuration structure can be made constant so as to reside in flash.

static void LPIT_ResetStateDelay(void)

Short wait for LPIT state reset.

After clear or set LPIT_EN, there should be delay longer than 4 LPIT functional clock.

static inline void LPIT_Reset(LPIT_Type *base)

Performs a software reset on the LPIT module.

This resets all channels and registers except the Module Control Register.

Parameters

- base – LPIT peripheral base address.

LPIT_RESET_STATE_DELAY

Delay used in LPIT_Reset.

The macro value should be larger than 4 * core clock / LPIT peripheral clock.

struct _lpit_chnl_params

#include <fsl_lpit.h> Structure to configure the channel timer.

Public Members

bool chainChannel

true: Timer chained to previous timer; false: Timer not chained

lpit_timer_modes_t timerMode

Timers mode of operation.

lpit_trigger_select_t triggerSelect

Trigger selection for the timer

lpit_trigger_source_t triggerSource

Decides if we use external or internal trigger.

bool enableReloadOnTrigger

true: Timer reloads when a trigger is detected; false: No effect

bool enableStopOnTimeout

true: Timer will stop after timeout; false: does not stop after timeout

bool enableStartOnTrigger

true: Timer starts when a trigger is detected; false: decrement immediately

struct _lpit_config

#include <fsl_lpit.h> LPIT configuration structure.

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

The configuration structure can be made constant so as to reside in flash.

Public Members

bool enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

bool enableRunInDoze

true: Timers run in doze mode; false: Timers stop in doze mode

2.30 LPSPI: Low Power Serial Peripheral Interface

2.31 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure lpspi_master_config_t.
- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the lpspi_master_config_t structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to lpspi_master_config_t structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi_slave_config_t.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the lpspi_slave_config_t structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to lpspi_slave_config_t structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns
The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters
- base – LPSPI peripheral address.
- statusFlags – The status flag used from type _lpspi_flags.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters
- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters
- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters
- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

bool LPSPI_CheckTransferArgument(LPSPI_Type *base, *lpspi_transfer_t* *transfer, bool isEdma)

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, *lpspi_master_slave_mode_t* mode)

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, *lpspi_which_pcs_t* select)

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.

- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI_SetPCSContinous(LPSPI_Type *base, bool IsContinous)

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, *lpspi_delay_type_t* whichDelay)

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.

uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, *lpspi_delay_type_t* whichDelay, uint32_t srcClock_Hz)

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function configures the SCK to PCS delay, PCS to SCK delay, or the delay between transfers. The delay names are available in type lpspi_delay_type_t.

The function calculates the value needed for the desired delay parameter and returns the actual calculated delay. An exact delay match may not be possible, in which case the closest match is calculated without going below the desired delay value. If the input exceeds the maximum capability, the maximum supported delay is returned.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nanoseconds.
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns

Actual calculated delay value in nanoseconds.

static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.

- data – The data word to be sent.

static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The data read from the data buffer.

void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_master_transfer_callback_t* callback, void *userData)

Initializes the LPSPI master handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_master_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns

true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag
Word Complete flag

enumerator kLPSPI_FrameCompleteFlag
Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag
Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag
Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag
Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag
Data Match flag

enumerator kLPSPI_ModuleBusyFlag
Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable
LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable
Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0
PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity
LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity

LPSPI clock polarity configuration.

Values:

enumerator kLPSPI_ClockPolarityActiveHigh

CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow

CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase

LPSPI clock phase configuration.

Values:

enumerator kLPSPI_ClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction

LPSPI data shifter direction options.

Values:

enumerator kLPSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kLPSPI_LsbFirst

Data transfers start with least significant bit.

enum _lpspi_host_request_select

LPSPI Host Request select configuration.

Values:

enumerator kLPSPI_HostReqExtPin

Host Request is an ext pin.

enumerator kLPSPI_HostReqInternalTrigger

Host Request is an internal trigger.

enum _lpspi_match_config

LPSPI Match configuration options.

Values:

enumerator kLPSI_MatchDisabled

LPSPI Match Disabled.

enumerator kLPSI_1stWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordEqualsM0and2ndWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0andNxtWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordAndM1EqualsM0andM1
LPSPI Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1
LPSPI Match Enabled.

enum _lpspi_pin_config
LPSPI pin (SDO and SDI) configuration.

Values:

enumerator kLPSPI_SdiInSdoOut
LPSPI SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut
LPSPI SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut
LPSPI SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut
LPSPI SDO input, SDI output.

enum _lpspi_data_out_config
LPSPI data output configuration.

Values:

enumerator kLpspiDataOutRetained
Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate
Data out is tristated when chip select is de-asserted

enum _lpspi_transfer_width
LPSPI transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPI delay type selection.

Values:

enumerator kLPSPI_PcsToSck
PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
Delay between transfers.

enum _lpspi_transfer_config_flag_for_master

Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPI slave transfer configFlags.

Values:

enumerator kLPSPI_SlavePcs0

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS0 signal

enumerator kLPSPI_SlavePcs1

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS1 signal

enumerator kLPSPI_SlavePcs2

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS2 signal

enumerator kLPSPI_SlavePcs3

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS3 signal

enumerator kLPSPI_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

enum _lpspi_transfer_state

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator kLPSPI_Idle

Nothing in the transmitter/receiver.

enumerator kLPSPI_Busy

Transfer queue is not finished.

enumerator kLPSPI_Error

Transfer error.

typedef enum *_lpspi_master_slave_mode* lpspi_master_slave_mode_t

LPSPI master or slave mode configuration.

typedef enum *_lpspi_which_pcs_config* lpspi_which_pcs_t

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum *_lpspi_pcs_polarity_config* lpspi_pcs_polarity_config_t

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *_lpspi_clock_polarity* lpspi_clock_polarity_t

LPSPI clock polarity configuration.

typedef enum *_lpspi_clock_phase* lpspi_clock_phase_t

LPSPI clock phase configuration.

typedef enum *_lpspi_shift_direction* lpspi_shift_direction_t

LPSPI data shifter direction options.

typedef enum *_lpspi_host_request_select* lpspi_host_request_select_t

LPSPI Host Request select configuration.

typedef enum *_lpspi_match_config* lpspi_match_config_t

LPSPI Match configuration options.

typedef enum *_lpspi_pin_config* lpspi_pin_config_t

LPSPI pin (SDO and SDI) configuration.

typedef enum *_lpspi_data_out_config* lpspi_data_out_config_t

LPSPI data output configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t

LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t

LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t

LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t

LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t

Forward declaration of the _lpspi_master_handle typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t

Forward declaration of the _lpspi_slave_handle typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)

Master completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)

Slave completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_lpspi_transfer* lpspi_transfer_t

LPSPI master/slave transfer structure.

volatile uint8_t g_lpspiDummyData[]

Global variable for dummy data value setting.

LPSPI_DUMMY_DATA

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

SPI_RETRY_TIMES

Retry times for waiting flag.

LPSPI_MASTER_PCS_SHIFT

LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_PCS_MASK

LPSPI master PCS shift macro , internal used.

LPSPI_SLAVE_PCS_SHIFT

LPSPI slave PCS shift macro , internal used.

LPSPI_SLAVE_PCS_MASK

LPSPI slave PCS shift macro , internal used.

struct lpspi_master_config

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate

Baud Rate for LPSPI.

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi__slave__config

#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs
: Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
: Desired PCS active high or low

lpspi_pin_config_t pinCfg
: Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
: Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct __lpspi__transfer

#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData
: Send buffer.

uint8_t *rxData
: Receive buffer.

volatile size_t dataSize
: Transfer bytes.

uint32_t configFlags
: Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

struct __lpspi__master__handle

#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
: Is PCS continuous in transfer.

volatile bool writeTcrInIsr
: A flag that whether should write TCR in ISR.

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile bool isTxMask
: A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame
: Number of bytes in each frame

volatile uint16_t frameSize
: Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

uint32_t txBuffIfNull
Used if the txData is NULL.

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

struct __lpspi_slave_handle
#include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
A flag that whether should byte swap.

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
: Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

2.32 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION
: LPSPI EDMA driver version.

DMA_MAX_TRANSFER_COUNT
: DMA max transfer size.

typedef struct *_lpspi_master_edma_handle* lpspi_master_edma_handle_t
: Forward declaration of the _lpspi_master_edma_handle typedefs.

typedef struct *_lpspi_slave_edma_handle* lpspi_slave_edma_handle_t
: Forward declaration of the _lpspi_slave_edma_handle typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)
: Completion callback function pointer type.

  Param base
  : LPSPI peripheral base address.

  Param handle
  : Pointer to the handle for the LPSPI master.

  Param status
  : Success or error code describing whether the transfer completed.

  Param userData
  : Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- configFlags – transfer configuration flags. _lpspi_transfer_config_flag_for_master.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

Returns
Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure, config field is not uesed.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.
- kStatus_InvalidArgument – The transfer structure is invalid.

Returns
Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, size_t *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns

status of status_t.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_slave_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size

should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle)

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.

status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, size_t *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns
status of status_t.

struct __lpspi_master_edma_handle

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

volatile uint8_t bytesLastRead
: Bytes for last read RDR.

volatile bool isThereExtraRxBytes
: Is there extra RX byte.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

edma_tcd_t *lastTimeTCD
: Pointer to the lastTime TCD

bool isMultiDMATransmit
: Is there multi DMA transmit

volatile uint8_t dmaTransmitTime
: DMA Transfer times.

uint32_t lastTimeDataBytes
: DMA transmit last Time data Bytes

uint32_t dataBytesEveryTime
: Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

edma_transfer_config_t transferConfigRx
: Config of DMA rx channel.

edma_transfer_config_t transferConfigTx
: Config of DMA tx channel.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

uint32_t transmitCommand
: Used to write TCR for DMA purpose.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
: SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
: *#include <fsl_lpspi_edma.h>* LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

volatile uint8_t bytesLastRead
: Bytes for last read RDR.

volatile bool isThereExtraRxBytes
: Is there extra RX byte.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

uint32_t txBuffIfNull
Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
Used if there is not rxData for DMA purpose.

volatile uint8_t state
LPSPI transfer state.

uint32_t errorCount
Error count for slave transfer.

lpspi_slave_edma_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg

edma_tcd_t lpspiSoftwareTCD[2]
SoftwareTCD, internal used

2.33 LPTMR: Low-Power Timer

void LPTMR_Init(LPTMR_Type *base, const *lptmr_config_t* *config)
Ungates the LPTMR clock and configures the peripheral for a basic operation.

Note: This API should be called at the beginning of the application using the LPTMR driver.

Parameters

- base – LPTMR peripheral base address
- config – A pointer to the LPTMR configuration structure.

void LPTMR_Deinit(LPTMR_Type *base)
Gates the LPTMR clock.

Parameters

- base – LPTMR peripheral base address

void LPTMR_GetDefaultConfig(*lptmr_config_t* *config)

Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

Parameters

- config – A pointer to the LPTMR configuration structure.

static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)

Enables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration lptmr_interrupt_enable_t

static inline void LPTMR_DisableInterrupts(LPTMR_Type *base, uint32_t mask)

Disables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration lptmr_interrupt_enable_t.

static inline uint32_t LPTMR_GetEnabledInterrupts(LPTMR_Type *base)

Gets the enabled LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration lptmr_interrupt_enable_t

static inline uint32_t LPTMR_GetStatusFlags(LPTMR_Type *base)

Gets the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration lptmr_status_flags_t

static inline void LPTMR_ClearStatusFlags(LPTMR_Type *base, uint32_t mask)

Clears the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration lptmr_status_flags_t.

static inline void LPTMR_SetTimerPeriod(LPTMR_Type *base, uint32_t ticks)

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note:

a. The TCF flag is set with the CNR equals the count provided here and then increments.

b. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

- base – LPTMR peripheral base address
- ticks – A timer period in units of ticks

static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – LPTMR peripheral base address

Returns

The current counter value in ticks

static inline void LPTMR_StartTimer(LPTMR_Type *base)

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

- base – LPTMR peripheral base address

static inline void LPTMR_StopTimer(LPTMR_Type *base)

Stops the timer.

This function stops the timer and resets the timer's counter register.

Parameters

- base – LPTMR peripheral base address

FSL_LPTMR_DRIVER_VERSION

Driver Version

enum _lptmr_pin_select

LPTMR pin selection used in pulse counter mode.

Values:

enumerator kLPTMR_PinSelectInput_0

Pulse counter input 0 is selected

enumerator kLPTMR_PinSelectInput_1
Pulse counter input 1 is selected

enumerator kLPTMR_PinSelectInput_2
Pulse counter input 2 is selected

enumerator kLPTMR_PinSelectInput_3
Pulse counter input 3 is selected

enum _lptmr_pin_polarity
LPTMR pin polarity used in pulse counter mode.

Values:

enumerator kLPTMR_PinPolarityActiveHigh
Pulse Counter input source is active-high

enumerator kLPTMR_PinPolarityActiveLow
Pulse Counter input source is active-low

enum _lptmr_timer_mode
LPTMR timer mode selection.

Values:

enumerator kLPTMR_TimerModeTimeCounter
Time Counter mode

enumerator kLPTMR_TimerModePulseCounter
Pulse Counter mode

enum _lptmr_prescaler_glitch_value
LPTMR prescaler/glitch filter values.

Values:

enumerator kLPTMR_Prescale_Glitch_0
Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR_Prescale_Glitch_1
Prescaler divide 4, glitch filter 2

enumerator kLPTMR_Prescale_Glitch_2
Prescaler divide 8, glitch filter 4

enumerator kLPTMR_Prescale_Glitch_3
Prescaler divide 16, glitch filter 8

enumerator kLPTMR_Prescale_Glitch_4
Prescaler divide 32, glitch filter 16

enumerator kLPTMR_Prescale_Glitch_5
Prescaler divide 64, glitch filter 32

enumerator kLPTMR_Prescale_Glitch_6
Prescaler divide 128, glitch filter 64

enumerator kLPTMR_Prescale_Glitch_7
Prescaler divide 256, glitch filter 128

enumerator kLPTMR_Prescale_Glitch_8
Prescaler divide 512, glitch filter 256

enumerator kLPTMR_Prescale_Glitch_9
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR_Prescale_Glitch_10
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR_Prescale_Glitch_11
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR_Prescale_Glitch_12
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR_Prescale_Glitch_13
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR_Prescale_Glitch_14
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR_Prescale_Glitch_15
Prescaler divide 65536, glitch filter 32768

enum _lptmr_prescaler_clock_select
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

Values:

enum _lptmr_interrupt_enable
List of the LPTMR interrupts.

Values:

enumerator kLPTMR_TimerInterruptEnable
Timer interrupt enable

enum _lptmr_status_flags
List of the LPTMR status flags.

Values:

enumerator kLPTMR_TimerCompareFlag
Timer compare flag

typedef enum *_lptmr_pin_select* lptmr_pin_select_t
LPTMR pin selection used in pulse counter mode.

typedef enum *_lptmr_pin_polarity* lptmr_pin_polarity_t
LPTMR pin polarity used in pulse counter mode.

typedef enum *_lptmr_timer_mode* lptmr_timer_mode_t
LPTMR timer mode selection.

typedef enum *_lptmr_prescaler_glitch_value* lptmr_prescaler_glitch_value_t
LPTMR prescaler/glitch filter values.

typedef enum *_lptmr_prescaler_clock_select* lptmr_prescaler_clock_select_t
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

typedef enum *_lptmr_interrupt_enable* lptmr_interrupt_enable_t

List of the LPTMR interrupts.

typedef enum *_lptmr_status_flags* lptmr_status_flags_t

List of the LPTMR status flags.

typedef struct *_lptmr_config* lptmr_config_t

LPTMR config structure.

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

The configuration struct can be made constant so it resides in flash.

static inline void LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)

Enable or disable timer DMA request.

Toggles CSR[TDRE] on the LPTMR side only: when enabled, every compare event (CSR[TCF]=1) raises an LPTMR DMA request line which is auto-cleared after the DMA controller services it.

Note: LPTMR runs on a low-power clock that is asynchronous to the EDMA bus clock. On some EDMA IP variants the channel has TWO gates that must BOTH be opened for an LPTMR request to be accepted: the base hardware request gate (ERQ, opened by EDMA_EnableChannelRequest()) and an extra asynchronous-request gate (opened by EDMA_EnableAsyncRequest() on EDMA4; the classic EDMA + DMAMUX combination does not need this).

Parameters

- base – base LPTMR peripheral base address
- enable – Switcher of timer DMA feature. “true” means to enable, “false” means to disable.

struct __lptmr_config

#include <fsl_lptmr.h> LPTMR config structure.

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

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode

Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler
: True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource
: LPTMR clock source

lptmr_prescaler_glitch_value_t value
: Prescaler or glitch filter value

2.34 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.35 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

status_t LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

Return values

- kStatus_Success – Deinit is success.
- kStatus_LPUART_Timeout – Timeout during deinit.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, flase to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in lpuart_handle_t.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators _lpuart_flags. To check for a specific status, compare the return value with enumerators in the _lpuart_flags. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of _lpuart_interrupt_enable.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

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

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferStartRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART_TransferStopRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, *lpuart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_LPUART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_DriverIRQHandler(uint32_t instance)

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- instance – LPUART instance.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually

Some flag can't manually clear

enumerator kStatus_LPUART_Error

Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun

LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_transmit_rts_polarity
LPUART transmitter RTS polarity.

Values:

enumerator kLPUART_RtsPolarityLow
Transmitter RTS is active low.

enumerator kLPUART_RtsPolarityHigh
Transmitter RTS is active high.

enum _lpuart_idle_type_select
LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag
Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
LPUART transmit CTS configure.

typedef enum *_lpuart_transmit_rts_polarity* lpuart_transmit_rts_polarity_t
LPUART transmitter RTS polarity.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *_lpuart_config* lpuart_config_t
LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_config
#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
RX RTS enable

bool enableTxRTS
TX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

lpuart_transmit_rts_polarity_t txRtsPolarity
TX RTS polarity

uint8_t rtsWatermark
RTS watermark

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

bool enableTx
Enable TX

bool enableRx
Enable RX

bool swapTxdRxd
Swap TXD and RXD pins

bool inverseTxd
Transmit Data Inversion - Setting true reverses the polarity of the transmitted data output

struct __lpuart_transfer
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __lpuart_handle
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

volatile size_t txDataSize
Size of the remaining data to send.

size_t txDataSizeAll
: Size of the data to send out.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t rxDataSizeAll
: Size of the data to receive.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
: Callback function.

void *userData
: LPUART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state.

bool isSevenDataBits
: Seven data bits flag.

bool is16bitData
: 16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed11___

Public Members

uint8_t *data
: The buffer of data to be transfer.

uint8_t *rxData
: The buffer to receive data.

uint16_t *rxData16
: The buffer to receive data.

const uint8_t *txData
: The buffer of data to be sent.

const uint16_t *txData16
: The buffer of data to be sent.

union ___unnamed13___

Public Members

const uint8_t *volatile txData
: Address of remaining data to send.

const uint16_t *volatile txData16
: Address of remaining data to send.

union ___unnamed15___

Public Members

uint8_t *volatile rxData
: Address of remaining data to receive.

uint16_t *volatile rxData16
: Address of remaining data to receive.

union ___unnamed17___

Public Members

uint8_t *rxRingBuffer
: Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16
: Start address of the receiver ring buffer.

2.36 LPUART eDMA Driver

void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

status_t LPUART_SendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- base – LPUART peripheral base address.
- lpuartEdmaHandle – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

typedef struct *_lpuart_edma_handle* lpuart_edma_handle_t

typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

struct _lpuart_edma_handle

#include <fsl_lpuart_edma.h> LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback

Callback function.

void *userData
: LPUART callback function parameter.

size_t rxDataSizeAll
: Size of the data to receive.

size_t txDataSizeAll
: Size of the data to send out.

edma_handle_t *txEdmaHandle
: The eDMA TX channel used.

edma_handle_t *rxEdmaHandle
: The eDMA RX channel used.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state

2.37 LTC: LP Trusted Cryptography

FSL_LTC_DRIVER_VERSION
: LTC driver version. Version 2.0.18.

  Current version: 2.0.18

  Change log:

  - Version 2.0.1
    - fixed warning during g++ compilation
  - Version 2.0.2
    - fixed [KPSDK-10932][LTC][SHA] LTC_HASH() blocks indefinitely when message size exceeds 4080 bytes
  - Version 2.0.3
    - fixed LTC_PKHA_CompareBigNum() in case an integer argument is an array of all zeroes
  - Version 2.0.4
    - constant LTC_PKHA_CompareBigNum() processing time
  - Version 2.0.5
    - Fix MISRA issues
  - Version 2.0.6
    - fixed [KPSDK-23603][LTC] AES Decrypt in ECB and CBC modes fail when ciphertext size > 0xff0 bytes
  - Version 2.0.7
    - Fix MISRA-2012 issues
  - Version 2.0.8

  - Fix Coverity issues
- Version 2.0.9
  - Fix sign-compare warning in ltc_set_context and in ltc_get_context
- Version 2.0.10
  - Fix MISRA-2012 issues
- Version 2.0.11
  - Fix MISRA-2012 issues
- Version 2.0.12
  - Fix AES Decrypt in CBC modes fail when used kLTC_DecryptKey.
- Version 2.0.13
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Init function.
- Version 2.0.14
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Deinit function.
- Version 2.0.15
  - Fix MISRA-2012 issues
- Version 2.0.16
  - Fix unitialized GCC warning in LTC_AES_GenerateDecryptKey()
- Version 2.0.17
  - Fix CMAC for payloads over one block, and if BRIC is present on the device, remove XCBC and "decrypt key" functionality
- Version 2.0.18
  - Fix CERT INT30-C and INT31-C compliance

void LTC_Init(LTC_Type *base)

Initializes the LTC driver. This function initializes the LTC driver.

Parameters

- base – LTC peripheral base address

void LTC_Deinit(LTC_Type *base)

Deinitializes the LTC driver. This function deinitializes the LTC driver.

Parameters

- base – LTC peripheral base address

void LTC_SetDpaMaskSeed(LTC_Type *base, uint32_t mask)

Sets the DPA Mask Seed register.

The DPA Mask Seed register reseeds the mask that provides resistance against DPA (differential power analysis) attacks on AES or DES keys.

Differential Power Analysis Mask (DPA) resistance uses a randomly changing mask that introduces "noise" into the power consumed by the AES or DES. This reduces the signal-to-noise ratio that differential power analysis attacks use to "guess" bits of the key. This randomly changing mask should be seeded at POR, and continues to provide DPA resistance from that point on. However, to provide even more DPA protection it is recommended that the DPA mask be reseeded after every 50,000 blocks have been processed. At that time, software can opt to write a new seed (preferably obtained from an RNG) into the DPA Mask

Seed register (DPAMS), or software can opt to provide the new seed earlier or later, or not at all. DPA resistance continues even if the DPA mask is never reseeded.

Parameters

- base – LTC peripheral base address
- mask – The DPA mask seed.

2.38 LTC AES driver

enum _ltc_aes_key_t

Type of AES key for ECB and CBC decrypt operations.

Values:

enumerator kLTC_EncryptKey

Input key is an encrypt key

enumerator kLTC_DecryptKey

Input key is a decrypt key

typedef enum *_ltc_aes_key_t* ltc_aes_key_t

Type of AES key for ECB and CBC decrypt operations.

status_t LTC_AES_GenerateDecryptKey(LTC_Type *base, const uint8_t *encryptKey, uint8_t *decryptKey, uint32_t keySize)

Transforms an AES encrypt key (forward AES) into the decrypt key (inverse AES).

Transforms the AES encrypt key (forward AES) into the decrypt key (inverse AES). The key derived by this function can be used as a direct load decrypt key for AES ECB and CBC decryption operations (keyType argument).

Parameters

- base – LTC peripheral base address
- encryptKey – Input key for decrypt key transformation
- decryptKey – **[out]** Output key, the decrypt form of the AES key.
- keySize – Size of the input key and output key in bytes. Must be 16, 24, or 32.

Returns

Status from key generation operation

status_t LTC_AES_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *key, uint32_t keySize)

Encrypts AES using the ECB block mode.

Encrypts AES using the ECB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns
Status from encrypt operation

status_t LTC_AES_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *key, uint32_t keySize, *ltc_aes_key_t* keyType)

Decrypts AES using ECB block mode.

Decrypts AES using ECB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key.
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- keyType – Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns
Status from decrypt operation

status_t LTC_AES_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t keySize)

Encrypts AES using CBC block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns
Status from encrypt operation

status_t LTC_AES_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t keySize, *ltc_aes_key_t* keyType)

Decrypts AES using CBC block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for decryption

- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- keyType – Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

Returns

Status from decrypt operation

status_t LTC_AES_CryptCtr(LTC_Type *base, const uint8_t *input, uint8_t *output, uint32_t size, uint8_t counter[16U], const uint8_t *key, uint32_t keySize, uint8_t counterlast[16U], uint32_t *szLeft)

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- base – LTC peripheral base address
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)
- key – Input key to use for forward AES cipher
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- counterlast – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- szLeft – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

status_t LTC_AES_EncryptTagGcm(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, uint8_t *tag, uint32_t tagSize)

Encrypts AES and tags using GCM block mode.

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD

- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Output hash tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from encrypt operation

status_t LTC_AES_DecryptTagGcm(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, const uint8_t *tag, uint32_t tagSize)

Decrypts AES and authenticates using GCM block mode.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Input hash tag to compare. Set to NULL to skip tag processing.
- tagSize – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from decrypt operation

status_t LTC_AES_EncryptTagCcm(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, uint8_t *tag, uint32_t tagSize)

Encrypts AES and tags using CCM block mode.

Encrypts AES and optionally tags using CCM block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes. Zero means authentication only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.

- aad – Input additional authentication data. Can be NULL if aadSize is zero.
- aadSize – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Generated output tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from encrypt operation

status_t LTC_AES_DecryptTagCcm(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, const uint8_t *tag, uint32_t tagSize)

Decrypts AES and authenticates using CCM block mode.

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes. Zero means authentication only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- aad – Input additional authentication data. Can be NULL if aadSize is zero.
- aadSize – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Received tag. Set to NULL to skip tag processing.
- tagSize – Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from decrypt operation

LTC_AES_BLOCK_SIZE

AES block size in bytes

LTC_AES_IV_SIZE

AES Input Vector size in bytes

LTC_KEY_REGISTER_READABLE

LTC_AES_DecryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR decrypt is mapped to the AES CTR generic operation

LTC_AES_EncryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR encrypt is mapped to the AES CTR generic operation

2.39 LTC DES driver

status_t LTC_DES_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key[8])

Encrypts DES using ECB block mode.

Encrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key[8])

Decrypts DES using ECB block mode.

Decrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using CBC block mode.

Encrypts DES using CBC block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Ouput ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using CBC block mode.

Decrypts DES using CBC block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using CFB block mode.

Encrypts DES using CFB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- size – Size of input data in bytes
- iv – Input initial block.
- key – Input key to use for encryption
- ciphertext – **[out]** Output ciphertext

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using CFB block mode.

Decrypts DES using CFB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using OFB block mode.

Encrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using OFB block mode.

Decrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using ECB block mode with two keys.

Encrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using ECB block mode with two keys.

Decrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using CBC block mode with two keys.

Encrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using CBC block mode with two keys.

Decrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle

- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using CFB block mode with two keys.

Encrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using CFB block mode with two keys.

Decrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using OFB block mode with two keys.

Encrypts triple DES using OFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext

- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using OFB block mode with two keys.

Decrypts triple DES using OFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using ECB block mode with three keys.

Encrypts triple DES using ECB block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using ECB block mode with three keys.

Decrypts triple DES using ECB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using CBC block mode with three keys.

Encrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using CBC block mode with three keys.

Decrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using CFB block mode with three keys.

Encrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and ouput data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using CFB block mode with three keys.

Decrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using OFB block mode with three keys.

Encrypts triple DES using OFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt

- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using OFB block mode with three keys.

Decrypts triple DES using OFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

LTC_DES_KEY_SIZE
LTC DES key size - 64 bits.

LTC_DES_IV_SIZE
LTC DES IV size - 8 bytes.

2.40 LTC HASH driver

enum _ltc_hash_algo_t
Supported cryptographic block cipher functions for HASH creation

Values:

enumerator kLTC_XcbcMac
XCBC-MAC (AES engine)

enumerator kLTC_Cmac
CMAC (AES engine)

enumerator kLTC_Sha1
SHA_1 (MDHA engine)

enumerator kLTC_Sha224
SHA_224 (MDHA engine)

enumerator kLTC_Sha256
SHA_256 (MDHA engine)

typedef enum *_ltc_hash_algo_t* ltc_hash_algo_t
Supported cryptographic block cipher functions for HASH creation

typedef struct *_ltc_hash_ctx_t* ltc_hash_ctx_t
Storage type used to save hash context.

status_t LTC_HASH_Init(LTC_Type *base, *ltc_hash_ctx_t* *ctx, *ltc_hash_algo_t* algo, const uint8_t *key, uint32_t keySize)
Initialize HASH context.

This function initialize the HASH. Key shall be supplied if the underlaying algoritm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlaying algoritm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

Parameters

- base – LTC peripheral base address
- ctx – **[out]** Output hash context
- algo – Underlaying algorithm to use for hash computation.
- key – Input key (NULL if underlaying algorithm is SHA)
- keySize – Size of input key in bytes

Returns
Status of initialization

status_t LTC_HASH_Update(*ltc_hash_ctx_t* *ctx, const uint8_t *input, uint32_t inputSize)
Add data to current HASH.

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

Parameters

- ctx – **[inout]** HASH context
- input – Input data
- inputSize – Size of input data in bytes

Returns
Status of the hash update operation

status_t LTC_HASH_Finish(*ltc_hash_ctx_t* *ctx, uint8_t *output, uint32_t *outputSize)
Finalize hashing.

Outputs the final hash and erases the context.

Parameters

- ctx – **[inout]** Input hash context
- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

Returns
Status of the hash finish operation

status_t LTC_HASH(LTC_Type *base, *ltc_hash_algo_t* algo, const uint8_t *input, uint32_t inputSize, const uint8_t *key, uint32_t keySize, uint8_t *output, uint32_t *outputSize)

Create HASH on given data.

Perform the full keyed HASH in one function call.

Parameters

- base – LTC peripheral base address
- algo – Block cipher algorithm to use for CMAC creation
- input – Input data
- inputSize – Size of input data in bytes
- key – Input key
- keySize – Size of input key in bytes
- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

LTC_HASH_CTX_SIZE

LTC HASH Context size.

struct _ltc_hash_ctx_t

#include <fsl_ltc.h> Storage type used to save hash context.

2.41 LTC PKHA driver

enum _ltc_pkha_timing_t

Use of timing equalized version of a PKHA function.

Values:

enumerator kLTC_PKHA_NoTimingEqualized

Normal version of a PKHA operation

enumerator kLTC_PKHA_TimingEqualized

Timing-equalized version of a PKHA operation

enum _ltc_pkha_f2m_t

Integer vs binary polynomial arithmetic selection.

Values:

enumerator kLTC_PKHA_IntegerArith

Use integer arithmetic

enumerator kLTC_PKHA_F2mArith

Use binary polynomial arithmetic

enum _ltc_pkha_montgomery_form_t

Montgomery or normal PKHA input format.

Values:

enumerator kLTC_PKHA_NormalValue

PKHA number is normal integer

enumerator kLTC_PKHA_MontgomeryFormat

PKHA number is in montgomery format

typedef struct *_ltc_pkha_ecc_point_t* ltc_pkha_ecc_point_t

PKHA ECC point structure

typedef enum *_ltc_pkha_timing_t* ltc_pkha_timing_t

Use of timing equalized version of a PKHA function.

typedef enum *_ltc_pkha_f2m_t* ltc_pkha_f2m_t

Integer vs binary polynomial arithmetic selection.

typedef enum *_ltc_pkha_montgomery_form_t* ltc_pkha_montgomery_form_t

Montgomery or normal PKHA input format.

int LTC_PKHA_CompareBigNum(const uint8_t *a, size_t sizeA, const uint8_t *b, size_t sizeB)

Compare two PKHA big numbers.

Compare two PKHA big numbers. Return 1 for a > b, -1 for a < b and 0 if they are same. PKHA big number is lsbyte first. Thus the comparison starts at msbyte which is the last member of tested arrays.

Parameters

- a – First integer represented as an array of bytes, lsbyte first.
- sizeA – Size in bytes of the first integer.
- b – Second integer represented as an array of bytes, lsbyte first.
- sizeB – Size in bytes of the second integer.

Returns

1 if a > b.

Returns

-1 if a < b.

Returns

0 if a = b.

status_t LTC_PKHA_NormalToMontgomery(LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t *sizeB, uint8_t *R2, uint16_t *sizeR2, *ltc_pkha_timing_t* equalTime, *ltc_pkha_f2m_t* arithType)

Converts from integer to Montgomery format.

This function computes R2 mod N and optionally converts A or B into Montgomery format of A or B.

Parameters

- base – LTC peripheral base address
- N – modulus
- sizeN – size of N in bytes
- A – **[inout]** The first input in non-Montgomery format. Output Montgomery format of the first input.
- sizeA – **[inout]** pointer to size variable. On input it holds size of input A in bytes. On output it holds size of Montgomery format of A in bytes.

- B – **[inout]** Second input in non-Montgomery format. Output Montgomery format of the second input.
- sizeB – **[inout]** pointer to size variable. On input it holds size of input B in bytes. On output it holds size of Montgomery format of B in bytes.
- R2 – **[out]** Output Montgomery factor R2 mod N.
- sizeR2 – **[out]** pointer to size variable. On output it holds size of Montgomery factor R2 mod N in bytes.
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_MontgomeryToNormal(LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t *sizeB, *ltc_pkha_timing_t* equalTime, *ltc_pkha_f2m_t* arithType)

Converts from Montgomery format to int.

This function converts Montgomery format of A or B into int A or B.

Parameters

- base – LTC peripheral base address
- N – modulus.
- sizeN – size of N modulus in bytes.
- A – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeA – **[inout]** pointer to size variable. On input it holds size of the input A in bytes. On output it holds size of non-Montgomery A in bytes.
- B – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeB – **[inout]** pointer to size variable. On input it holds size of the input B in bytes. On output it holds size of non-Montgomery B in bytes.
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_ModAdd(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Performs modular addition - (A + B) mod N.

This function performs modular addition of (A + B) mod N, with either integer or binary polynomial (F2m) inputs. In the F2m form, this function is equivalent to a bitwise XOR and it is functionally the same as subtraction.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes

- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus. For F2m operation this can be NULL, as N is ignored during F2m polynomial addition.
- sizeN – Size of N in bytes. This must be given for both integer and F2m polynomial additions.
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_ModSub1(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize)

Performs modular subtraction - (A - B) mod N.

This function performs modular subtraction of (A - B) mod N with integer inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes

Returns
Operation status.

status_t LTC_PKHA_ModSub2(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize)

Performs modular subtraction - (B - A) mod N.

This function performs modular subtraction of (B - A) mod N, with integer inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation

- resultSize – **[out]** Output size of operation in bytes

Returns
Operation status.

status_t LTC_PKHA_ModMul(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType, *ltc_pkha_montgomery_form_t* montIn, *ltc_pkha_montgomery_form_t* montOut, *ltc_pkha_timing_t* equalTime)

Performs modular multiplication - (A x B) mod N.

This function performs modular multiplication with either integer or binary polynomial (F2m) inputs. It can optionally specify whether inputs and/or outputs will be in Montgomery form or not.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus.
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)
- montIn – Format of inputs
- montOut – Format of output
- equalTime – Run the function time equalized or no timing equalization. This argument is ignored for F2m modular multiplication.

Returns
Operation status.

status_t LTC_PKHA_ModExp(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, const uint8_t *E, uint16_t sizeE, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType, *ltc_pkha_montgomery_form_t* montIn, *ltc_pkha_timing_t* equalTime)

Performs modular exponentiation - (A^E) mod N.

This function performs modular exponentiation with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes

- E – exponent
- sizeE – Size of E in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- montIn – Format of A input (normal or Montgomery)
- arithType – Type of arithmetic to perform (integer or F2m)
- equalTime – Run the function time equalized or no timing equalization.

Returns
Operation status.

status_t LTC_PKHA_ModRed(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Performs modular reduction - (A) mod N.

This function performs modular reduction with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_ModInv(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Performs modular inversion - (A^-1) mod N.

This function performs modular inversion with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_ModR2(LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Computes integer Montgomery factor R^2 mod N.

This function computes a constant to assist in converting operands into the Montgomery residue system representation.

Parameters

- base – LTC peripheral base address
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_GCD(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Calculates the greatest common divisor - GCD (A, N).

This function calculates the greatest common divisor of two inputs with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first value (must be smaller than or equal to N)
- sizeA – Size of A in bytes
- N – second value (must be non-zero)
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_PrimalityTest(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, bool *res)

Executes Miller-Rabin primality test.

This function calculates whether or not a candidate prime number is likely to be a prime.

Parameters

- base – LTC peripheral base address
- A – initial random seed
- sizeA – Size of A in bytes
- B – number of trial runs

- sizeB – Size of B in bytes
- N – candidate prime integer
- sizeN – Size of N in bytes
- res – **[out]** True if the value is likely prime or false otherwise

Returns
Operation status.

status_t LTC_PKHA_ECC_PointAdd(LTC_Type *base, const *ltc_pkha_ecc_point_t* *A, const *ltc_pkha_ecc_point_t* *B, const uint8_t *N, const uint8_t *R2modN, const uint8_t *aCurveParam, const uint8_t *bCurveParam, uint8_t size, *ltc_pkha_f2m_t* arithType, *ltc_pkha_ecc_point_t* *result)

Adds elliptic curve points - A + B.

This function performs ECC point addition over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – LTC peripheral base address
- A – Left-hand point
- B – Right-hand point
- N – Prime modulus of the field
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from LTC_PKHA_ModR2() function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns
Operation status.

status_t LTC_PKHA_ECC_PointDouble(LTC_Type *base, const *ltc_pkha_ecc_point_t* *B, const uint8_t *N, const uint8_t *aCurveParam, const uint8_t *bCurveParam, uint8_t size, *ltc_pkha_f2m_t* arithType, *ltc_pkha_ecc_point_t* *result)

Doubles elliptic curve points - B + B.

This function performs ECC point doubling over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – LTC peripheral base address
- B – Point to double
- N – Prime modulus of the field
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)

- result – **[out]** Result point

Returns
Operation status.

status_t LTC_PKHA_ECC_PointMul(LTC_Type *base, const *ltc_pkha_ecc_point_t* *A, const uint8_t *E, uint8_t sizeE, const uint8_t *N, const uint8_t *R2modN, const uint8_t *aCurveParam, const uint8_t *bCurveParam, uint8_t size, *ltc_pkha_timing_t* equalTime, *ltc_pkha_f2m_t* arithType, *ltc_pkha_ecc_point_t* *result, bool *infinity)

Multiplies an elliptic curve point by a scalar - E x (A0, A1).

This function performs ECC point multiplication to multiply an ECC point by a scalar integer multiplier over a prime field (Fp) or a binary field (F2m).

Parameters

- base – LTC peripheral base address
- A – Point as multiplicand
- E – Scalar multiple
- sizeE – The size of E, in bytes
- N – Modulus, a prime number for the Fp field or Irreducible polynomial for F2m field.
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from LTC_PKHA_ModR2() function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (C parameter for operation over F2m).
- size – Size in bytes of curve points and parameters
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point
- infinity – **[out]** Output true if the result is point of infinity, and false otherwise. Writing of this output will be ignored if the argument is NULL.

Returns
Operation status.

struct _ltc_pkha_ecc_point_t

#include <fsl_ltc.h> PKHA ECC point structure

Public Members

uint8_t *X

X coordinate (affine)

uint8_t *Y

Y coordinate (affine)

2.42 LTC Blocking APIs

2.43 Ltc_edma_driver

FSL_LTC_EDMA_DRIVER_VERSION

LTC EDMA driver version. Version 2.0.16.

typedef struct *_ltc_edma_handle* ltc_edma_handle_t

typedef void (*ltc_edma_callback_t)(LTC_Type *base, *ltc_edma_handle_t* *handle, *status_t* status, void *userData)

LTC eDMA callback function.

typedef *status_t* (*ltc_edma_state_machine_t)(LTC_Type *base, *ltc_edma_handle_t* *handle)

LTC eDMA state machine function. It is defined only for private usage inside LTC eDMA driver.

void LTC_CreateHandleEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, *ltc_edma_callback_t* callback, void *userData, *edma_handle_t* *inputFifoEdmaHandle, *edma_handle_t* *outputFifoEdmaHandle)

Init the LTC eDMA handle which is used in transactional functions.

Parameters

- base – LTC module base address
- handle – Pointer to ltc_edma_handle_t structure
- callback – Callback function, NULL means no callback.
- userData – Callback function parameter.
- inputFifoEdmaHandle – User requested eDMA handle for Input FIFO eDMA.
- outputFifoEdmaHandle – User requested eDMA handle for Output FIFO eDMA.

struct _ltc_edma_handle

#include <fsl_ltc_edma.h> LTC eDMA handle. It is defined only for private usage inside LTC eDMA driver.

Public Members

ltc_edma_callback_t callback

Callback function.

void *userData

LTC callback function parameter.

edma_handle_t *inputFifoEdmaHandle

The eDMA TX channel used.

edma_handle_t *outputFifoEdmaHandle

The eDMA RX channel used.

ltc_edma_state_machine_t state_machine

State machine.

uint32_t state

Internal state.

const uint8_t *inData
: Input data.

uint8_t *outData
: Output data.

uint32_t size
: Size of input and output data in bytes.

uint32_t modeReg
: LTC mode register.

uint8_t *counter
: Input counter (updates on return)

const uint8_t *key
: Input key to use for forward AES cipher

uint32_t keySize
: Size of the input key, in bytes. Must be 16, 24, or 32.

uint8_t *counterlast
: Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.

uint32_t *szLeft
: Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

uint32_t lastSize
: Last size.

2.44 LTC eDMA AES driver

status_t LTC_AES_EncryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *key, uint32_t keySize)

Encrypts AES using the ECB block mode.

Encrypts AES using the ECB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

status_t LTC_AES_DecryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *key, uint32_t keySize, *ltc_aes_key_t* keyType)

Decrypts AES using ECB block mode.

Decrypts AES using ECB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key.
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- keyType – Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns

Status from decrypt operation

status_t LTC_AES_EncryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t keySize)

Encrypts AES using CBC block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

status_t LTC_AES_DecryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t keySize, *ltc_aes_key_t* keyType)

Decrypts AES using CBC block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input cipher text to decrypt

- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- keyType – Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

Returns

Status from decrypt operation

status_t LTC_AES_CryptCtrEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *input, uint8_t *output, uint32_t size, uint8_t counter[16U], const uint8_t *key, uint32_t keySize, uint8_t counterlast[16U], uint32_t *szLeft)

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)
- key – Input key to use for forward AES cipher
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- counterlast – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- szLeft – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

LTC_AES_DecryptCtrEDMA(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR decrypt is mapped to the AES CTR generic operation

LTC_AES_EncryptCtrEDMA(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR encrypt is mapped to the AES CTR generic operation

2.45 Ltc_edma_driver_des

status_t LTC_DES_EncryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key[8])

Encrypts DES using ECB block mode.

Encrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key[8])

Decrypts DES using ECB block mode.

Decrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using CBC block mode.

Encrypts DES using CBC block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Ouput ciphertext
- size – Size of input and output data in bytes

- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for encryption

Returns
Status from encrypt/decrypt operation

status_t LTC_DES_DecryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using CBC block mode.

Decrypts DES using CBC block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for decryption

Returns
Status from encrypt/decrypt operation

status_t LTC_DES_EncryptCfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using CFB block mode.

Encrypts DES using CFB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- size – Size of input data in bytes
- iv – Input initial block.
- key – Input key to use for encryption
- ciphertext – **[out]** Output ciphertext

Returns
Status from encrypt/decrypt operation

status_t LTC_DES_DecryptCfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using CFB block mode.

Decrypts DES using CFB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key – Input key to use for decryption

Returns
Status from encrypt/decrypt operation

status_t LTC_DES_EncryptOfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using OFB block mode.

Encrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for encryption

Returns
Status from encrypt/decrypt operation

status_t LTC_DES_DecryptOfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using OFB block mode.

Decrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for decryption

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using ECB block mode with two keys.

Encrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using ECB block mode with two keys.

Decrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using CBC block mode with two keys.

Encrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.

- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using CBC block mode with two keys.

Decrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptCfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using CFB block mode with two keys.

Encrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle

- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptCfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using CFB block mode with two keys.

Decrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptOfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using OFB block mode with two keys.

Encrypts triple DES using OFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptOfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using OFB block mode with two keys.

Decrypts triple DES using OFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using ECB block mode with three keys.

Encrypts triple DES using ECB block mode with three keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptEcbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using ECB block mode with three keys.

Decrypts triple DES using ECB block mode with three keys.

Parameters

- base – LTC peripheral base address

- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using CBC block mode with three keys.

Encrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptCbcEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using CBC block mode with three keys.

Decrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes

- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptCfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using CFB block mode with three keys.

Encrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and ouput data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptCfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using CFB block mode with three keys.

Decrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptOfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using OFB block mode with three keys.

Encrypts triple DES using OFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptOfbEDMA(LTC_Type *base, *ltc_edma_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using OFB block mode with three keys.

Decrypts triple DES using OFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- handle – pointer to ltc_edma_handle_t structure which stores the transaction state.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

2.46 MCM: Miscellaneous Control Module

FSL_MCM_DRIVER_VERSION
MCM driver version.

Enum _mcm_interrupt_flag. Interrupt status flag mask. .

Values:

enumerator kMCM_CacheWriteBuffer
Cache Write Buffer Error Enable.

enumerator kMCM_ParityError
Cache Parity Error Enable.

enumerator kMCM_FPUInvalidOperation
FPU Invalid Operation Interrupt Enable.

enumerator kMCM_FPUDivideByZero
FPU Divide-by-zero Interrupt Enable.

enumerator kMCM_FPUOverflow
FPU Overflow Interrupt Enable.

enumerator kMCM_FPUUnderflow
FPU Underflow Interrupt Enable.

enumerator kMCM_FPUInexact
FPU Inexact Interrupt Enable.

enumerator kMCM_FPUInputDenormalInterrupt
FPU Input Denormal Interrupt Enable.

typedef union *_mcm_buffer_fault_attribute* mcm_buffer_fault_attribute_t
The union of buffer fault attribute.

typedef union *_mcm_lmem_fault_attribute* mcm_lmem_fault_attribute_t
The union of LMEM fault attribute.

static inline void MCM_EnableCrossbarRoundRobin(MCM_Type *base, bool enable)
Enables/Disables crossbar round robin.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
  - **true** Enable crossbar round robin.
  - **false** disable crossbar round robin.

static inline void MCM_EnableInterruptStatus(MCM_Type *base, uint32_t mask)
Enables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(_mcm_interrupt_flag).

static inline void MCM_DisableInterruptStatus(MCM_Type *base, uint32_t mask)

Disables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(_mcm_interrupt_flag).

static inline uint16_t MCM_GetInterruptStatus(MCM_Type *base)

Gets the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline void MCM_ClearCacheWriteBufferErroStatus(MCM_Type *base)

Clears the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultAddress(MCM_Type *base)

Gets buffer fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetBufferFaultAttribute(MCM_Type *base, *mcm_buffer_fault_attribute_t* *bufferfault)

Gets buffer fault attributes.

Parameters

- base – MCM peripheral base address.
- bufferfault – Structure to store the result.

static inline uint32_t MCM_GetBufferFaultData(MCM_Type *base)

Gets buffer fault data.

Parameters

- base – MCM peripheral base address.

static inline void MCM_LimitCodeCachePeripheralWriteBuffering(MCM_Type *base, bool enable)

Limit code cache peripheral write buffering.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
  - **true** Enable limit code cache peripheral write buffering.
  - **false** disable limit code cache peripheral write buffering.

static inline void MCM_BypassFixedCodeCacheMap(MCM_Type *base, bool enable)

Bypass fixed code cache map.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
  - **true** Enable bypass fixed code cache map.
  - **false** disable bypass fixed code cache map.

static inline void MCM_EnableCodeBusCache(MCM_Type *base, bool enable)

Enables/Disables code bus cache.

Parameters

- base – MCM peripheral base address.
- enable – Used to disable/enable code bus cache.
  - **true** Enable code bus cache.
  - **false** disable code bus cache.

static inline void MCM_ForceCodeCacheToNoAllocation(MCM_Type *base, bool enable)

Force code cache to no allocation.

Parameters

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
  - **true** Force code cache to no allocation.
  - **false** Force code cache to allocation.

static inline void MCM_EnableCodeCacheWriteBuffer(MCM_Type *base, bool enable)

Enables/Disables code cache write buffer.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
  - **true** Enable code cache write buffer.
  - **false** Disable code cache write buffer.

static inline void MCM_ClearCodeBusCache(MCM_Type *base)

Clear code bus cache.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnablePcParityFaultReport(MCM_Type *base, bool enable)

Enables/Disables PC Parity Fault Report.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
  - **true** Enable PC Parity Fault Report.
  - **false** disable PC Parity Fault Report.

static inline void MCM_EnablePcParity(MCM_Type *base, bool enable)

Enables/Disables PC Parity.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
  - **true** Enable PC Parity.
  - **false** disable PC Parity.

static inline void MCM_LockConfigState(MCM_Type *base)

Lock the configuration state.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnableCacheParityReporting(MCM_Type *base, bool enable)

Enables/Disables cache parity reporting.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
  - **true** Enable cache parity reporting.
  - **false** disable cache parity reporting.

static inline uint32_t MCM_GetLmemFaultAddress(MCM_Type *base)

Gets LMEM fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetLmemFaultAttribute(MCM_Type *base, *mcm_lmem_fault_attribute_t* *lmemFault)

Get LMEM fault attributes.

Parameters

- base – MCM peripheral base address.
- lmemFault – Structure to store the result.

static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)

Gets LMEM fault data.

Parameters

- base – MCM peripheral base address.

MCM_LMFATR_TYPE_MASK

MCM_LMFATR_MODE_MASK

MCM_LMFATR_BUFF_MASK

MCM_LMFATR_CACH_MASK

MCM_ISCR_STAT_MASK

FSL_COMPONENT_ID

union _mcm_buffer_fault_attribute

#include <fsl_mcm.h> The union of buffer fault attribute.

Public Members

uint32_t attribute

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

struct *_mcm_buffer_fault_attribute._mcm_buffer_fault_attribut* attribute_memory

struct __mcm__buffer__fault__attribut

#include <fsl_mcm.h>

Public Members

uint32_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32_t busErrorSize

Indicates the size of the cache write buffer access.

uint32_t busErrorAccess

Indicates the type of system bus access.

uint32_t busErrorMasterID

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32_t busErrorOverrun

Indicates if another cache write buffer bus error is detected.

union __mcm__lmem__fault__attribute

#include <fsl_mcm.h> The union of LMEM fault attribute.

Public Members

uint32_t attribute

Indicates the attributes of the LMEM fault detected.

struct *_mcm_lmem_fault_attribute._mcm_lmem_fault_attribut* attribute__memory

struct __mcm__lmem__fault__attribut

#include <fsl_mcm.h>

Public Members

uint32_t parityFaultProtectionSignal

Indicates the features of parity fault protection signal.

uint32_t parityFaultMasterSize

Indicates the parity fault master size.

uint32_t parityFaultWrite

Indicates the parity fault is caused by read or write.

uint32_t backdoorAccess

Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32_t parityFaultSyndrome

Indicates the parity fault syndrome.

uint32_t overrun

Indicates the number of faultss.

2.47 MSMC: Multicore System Mode Controller

static inline void SMC_SetPowerModeProtection(SMC_Type *base, uint8_t allowedModes)

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the smc_power_mode_protection_t. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map, for example, to allow LLS and VLLS, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeLls | kSMC_AllowPowerModeVlls). To allow all modes, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeAll).

Parameters

- base – SMC peripheral base address.
- allowedModes – Bitmap of the allowed power modes.

static inline *smc_power_state_t* SMC_GetPowerModeState(SMC_Type *base)

Gets the current power mode status.

This function returns the current power mode stat. Once application switches the power mode, it should always check the stat to check whether it runs into the specified mode or not. An application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the smc_power_state_t for information about the power stat.

Parameters

- base – SMC peripheral base address.

Returns

Current power mode status.

static inline void SMC_PreEnterStopModes(void)

Prepare to enter stop modes.

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

static inline void SMC_PostExitStopModes(void)

Recovering after wake up from stop modes.

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used together with SMC_PreEnterStopModes.

static inline void SMC_PreEnterWaitModes(void)

Prepare to enter wait modes.

This function should be called before entering WAIT/VLPW modes..

static inline void SMC_PostExitWaitModes(void)

Recovering after wake up from stop modes.

This function should be called after wake up from WAIT/VLPW modes. It is used together with SMC_PreEnterWaitModes.

status_t SMC_SetPowerModeRun(SMC_Type *base)

Configure the system to RUN power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeHsrun(SMC_Type *base)

Configure the system to HSRUN power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeWait(SMC_Type *base)

Configure the system to WAIT power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeStop(SMC_Type *base, *smc_partial_stop_option_t* option)

Configure the system to Stop power mode.

Parameters

- base – SMC peripheral base address.
- option – Partial Stop mode option.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlpr(SMC_Type *base)

Configure the system to VLPR power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlpw(SMC_Type *base)

Configure the system to VLPW power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlps(SMC_Type *base)

Configure the system to VLPS power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeLls(SMC_Type *base)

Configure the system to LLS power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlls0(SMC_Type *base)

Configure the system to VLLS0 power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlls2(SMC_Type *base)

Configure the system to VLLS2 power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

static inline uint32_t SMC_GetPreviousResetSources(SMC_Type *base)

Gets the reset source status which caused a previous reset.

This function gets the current reset source status. Use source masks defined in the smc_reset_source_t to get the desired source status.

Example: To get all reset source statuses.

```
resetStatus = SMC_GetPreviousResetSources(SMC0) & kSMC_SourceAll;
```

Example: To test whether the MCU is reset using Watchdog.

```
uint32_t resetStatus;

resetStatus = SMC_GetPreviousResetSources(SMC0) & kSMC_SourceWdog;
```

Example: To test multiple reset sources.

```
uint32_t resetStatus;

resetStatus = SMC_GetPreviousResetSources(SMC0) & (kSMC_SourceWdog | kSMC_SourcePin);
```

Parameters

- base – SMC peripheral base address.

Returns
All reset source status bit map.

static inline uint32_t SMC_GetStickyResetSources(SMC_Type *base)

Gets the sticky reset source status.

This function gets the current reset source status that has not been cleared by software for some specific source.

Example: To get all reset source statuses.

```
uint32_t resetStatus;

resetStatus = SMC_GetStickyResetSources(SMC0) & kSMC_SourceAll;
```

Example, To test whether the MCU is reset using Watchdog.

```
uint32_t resetStatus;

resetStatus = SMC_GetStickyResetSources(SMC0) & kSMC_SourceWdog;
```

Example To test multiple reset sources.

```
uint32_t resetStatus;

resetStatus = SMC_GetStickyResetSources(SMC0) & (kSMC_SourceWdog | kSMC_SourcePin);
```

Parameters

- base – SMC peripheral base address.

Returns

All reset source status bit map.

static inline void SMC_ClearStickyResetSources(SMC_Type *base, uint32_t sourceMasks)

Clears the sticky reset source status.

This function clears the sticky system reset flags indicated by source masks.

Example: Clears multiple reset sources.

```
SMC_ClearStickyResetSources(SMC0, (kSMC_SourceWdog | kSMC_SourcePin));
```

Parameters

- base – SMC peripheral base address.
- sourceMasks – reset source status bit map

void SMC_ConfigureResetPinFilter(SMC_Type *base, const *smc_reset_pin_filter_config_t* *config)

Configures the reset pin filter.

This function sets the reset pin filter including the enablement/disablement and filter width.

Parameters

- base – SMC peripheral base address.
- config – Pointer to the configuration structure.

static inline void SMC_SetSystemResetInterruptConfig(SMC_Type *base, uint32_t intMask)

Sets the system reset interrupt configuration.

For a graceful shut down, the MSMC supports delaying the assertion of the system reset for a period of time when the reset interrupt is generated. This function can be used to enable the interrupt. The interrupts are passed in as bit mask. See smc_interrupt_enable_t for details. For example, to delay a reset after the WDOG timeout or PIN reset occurs, configure as follows: SMC_SetSystemResetInterruptConfig(SMC0, (kSMC_IntWdog | kSMC_IntPin));

Parameters

- base – SMC peripheral base address.
- intMask – Bit mask of the system reset interrupts to enable. See smc_interrupt_enable_t for details.

static inline uint32_t SMC_GetResetInterruptSourcesStatus(SMC_Type *base)

Gets the source status of the system reset interrupt.

This function gets the source status of the reset interrupt. Use source masks defined in the smc_interrupt_enable_t to get the desired source status.

Example: To get all reset interrupt source statuses.

```
uint32_t interruptStatus;

interruptStatus = SMC_GetResetInterruptSourcesStatus(SMC0) & kSMC_IntAll;
```

Example: To test whether the reset interrupt of Watchdog is pending.

```
uint32_t interruptStatus;

interruptStatus = SMC_GetResetInterruptSourcesStatus(SMC0) & kSMC_IntWdog;
```

Example: To test multiple reset interrupt sources.

```
uint32_t interruptStatus;

interruptStatus = SMC_GetResetInterruptSourcesStatus(SMC0) & (kSMC_IntWdog | kSMC_IntPin);
```

Parameters

- base – SMC peripheral base address.

Returns

All reset interrupt source status bit map.

static inline void SMC_ClearResetInterruptSourcesStatus(SMC_Type *base, uint32_t intMask)

Clears the source status of the system reset interrupt.

This function clears the source status of the reset interrupt. Use source masks defined in the smc_interrupt_enable_t to get the desired source status.

Example: To clear all reset interrupt source statuses.

```
uint32_t interruptStatus;

MMC_ClearResetInterruptSourcesStatus(SMC0, kSMC_IntAll);
```

Example, To clear the reset interrupt of Watchdog.

```
uint32_t interruptStatus;

SMC_ClearResetInterruptSourcesStatus(SMC0, kSMC_IntWdog);
```

Example, To clear multiple reset interrupt sources status.

```
uint32_t interruptStatus;

SMC_ClearResetInterruptSourcesStatus(SMC0, (kSMC_IntWdog | kSMC_IntPin));
```

Parameters

- base – SMC peripheral base address.
- intMask – All reset interrupt source status bit map to clear.

static inline void SMC_SetCoreSoftwareResetConfig(SMC_Type *base, uint32_t intMask)

Sets the core software reset feature configuration.

The MSMC supports delaying the assertion of the system reset for a period of time while a core software reset is generated. This allows software to recover without reseting the entire system. This function can be used to enable/disable the core software reset feature. The interrupts are passed in as bit mask. See smc_interrupt_enable_t for details. For example, to delay a system after the WDOG timeout or PIN core software reset occurs, configure as follows: SMC_SetCoreSoftwareResetConfig(SMC0, (kSMC_IntWdog | kSMC_IntPin));

Parameters

- base – SMC peripheral base address.
- intMask – Bit mask of the core software reset to enable. See smc_interrupt_enable_t for details.

static inline uint32_t SMC_GetBootOptionConfig(SMC_Type *base)

Gets the boot option configuration.

This function gets the boot option configuration of MSMC.

Parameters

- base – SMC peripheral base address.

Returns

The boot option configuration. 1 means boot option enabled. 0 means not.

FSL_MSMC_DRIVER_VERSION

MSMC driver version.

enum _smc_power_mode_protection

Power Modes Protection.

Values:

enumerator kSMC_AllowPowerModeVlls

Allow Very-Low-Leakage Stop Mode.

enumerator kSMC_AllowPowerModeLls

Allow Low-Leakage Stop Mode.

enumerator kSMC_AllowPowerModeVlp

Allow Very-Low-Power Mode.

enumerator kSMC_AllowPowerModeHsrun

Allow High Speed Run mode.

enumerator kSMC_AllowPowerModeAll

Allow all power mode.

enum _smc_power_state

Power Modes in PMSTAT.

Values:

enumerator kSMC_PowerStateRun

0000_0001 - Current power mode is RUN

enumerator kSMC_PowerStateStop

0000_0010 - Current power mode is any STOP mode

enumerator kSMC_PowerStateVlpr

0000_0100 - Current power mode is VLPR

enumerator kSMC_PowerStateHsrun

1000_0000 - Current power mode is HSRUN

enum _smc_power_stop_entry_status

Power Stop Entry Status in PMSTAT.

Values:

enumerator kSMC_PowerStopEntryAlt0

Indicates a Stop mode entry since this field was last cleared.

enumerator kSMC_PowerStopEntryAlt1
Indicates the system bus masters acknowledged the Stop mode entry.

enumerator kSMC_PowerStopEntryAlt2
Indicates the system clock peripherals acknowledged the Stop mode entry.

enumerator kSMC_PowerStopEntryAlt3
Indicates the bus clock peripherals acknowledged the Stop mode entry.

enumerator kSMC_PowerStopEntryAlt4
Indicates the slow clock peripherals acknowledged the Stop mode entry.

enumerator kSMC_PowerStopEntryAlt5
Indicates Stop mode entry completed.

enum _smc_run_mode
Run mode definition.

Values:

enumerator kSMC_RunNormal
normal RUN mode.

enumerator kSMC_RunVlpr
Very-Low-Power RUN mode.

enumerator kSMC_Hsrun
High Speed Run mode (HSRUN).

enum _smc_stop_mode
Stop mode definition.

Values:

enumerator kSMC_StopNormal
Normal STOP mode.

enumerator kSMC_StopVlps
Very-Low-Power STOP mode.

enumerator kSMC_StopLls
Low-Leakage Stop mode.

enumerator kSMC_StopVlls2
Very-Low-Leakage Stop mode, VLPS2/3.

enumerator kSMC_StopVlls0
Very-Low-Leakage Stop mode, VLPS0/1.

enum _smc_partial_stop_mode
Partial STOP option.

Values:

enumerator kSMC_PartialStop
STOP - Normal Stop mode

enumerator kSMC_PartialStop1
Partial Stop with both system and bus clocks disabled

enumerator kSMC_PartialStop2
Partial Stop with system clock disabled and bus clock enabled

enumerator kSMC_PartialStop3
Partial Stop with system clock enabled and bus clock disabled

SMC configuration status.

Values:

enumerator kStatus_SMC_StopAbort
Entering Stop mode is abort

enum _smc_reset_source
System Reset Source Name definitions.

Values:

enumerator kSMC_SourceWakeup
Very low-leakage wakeup reset

enumerator kSMC_SourcePor
Power on reset

enumerator kSMC_SourceLvd
Low-voltage detect reset

enumerator kSMC_SourceHvd
High-voltage detect reset

enumerator kSMC_SourceWarm
Warm reset. Warm Reset flag will assert if any of the system reset sources in this register assert (SRS[31:8])

enumerator kSMC_SourceFatal
Fatal reset

enumerator kSMC_SourceCore
Software reset that only reset the core, NOT a sticky system reset source.

enumerator kSMC_SourcePin
RESET_B pin reset.

enumerator kSMC_SourceMdm
MDM reset.

enumerator kSMC_SourceRstAck
Reset Controller timeout reset.

enumerator kSMC_SourceStopAck
Stop timeout reset

enumerator kSMC_SourceScg
SCG loss of lock or loss of clock

enumerator kSMC_SourceWdog
Watchdog reset

enumerator kSMC_SourceSoftware
Software reset

enumerator kSMC_SourceLockup
Lockup reset. Core lockup or exception.

enumerator kSMC_SourceJtag
JTAG system reset

enumerator kSMC_SourceVbat

enumerator kSMC_SourceSecVio
Security violation reset

enumerator kSMC_SourceTamper
Tamper reset

enumerator kSMC_SourceCore0
Core0 System Reset.

enumerator kSMC_SourceCore1
Core1 System Reset.

enumerator kSMC_SourceAll

enum _smc_interrupt_enable
System reset interrupt enable bit definitions.

Values:

enumerator kSMC_IntNone
No interrupt enabled.

enumerator kSMC_IntPin
Pin reset interrupt.

enumerator kSMC_IntMdm
MDM reset interrupt.

enumerator kSMC_IntStopAck
Stop timeout reset interrupt.

enumerator kSMC_IntWdog
Watchdog interrupt.

enumerator kSMC_IntSoftware
Software reset interrupts.

enumerator kSMC_IntLockup
Lock up interrupt.

enumerator kSMC_IntVbat

enumerator kSMC_IntCore0

enumerator kSMC_IntCore1
Core 0 interrupts.

enumerator kSMC_IntAll
Core 1 interrupts. All system reset interrupts.

typedef enum *_smc_power_mode_protection* smc_power_mode_protection_t
Power Modes Protection.

typedef enum *_smc_power_state* smc_power_state_t
Power Modes in PMSTAT.

typedef enum *_smc_power_stop_entry_status* smc_power_stop_entry_status_t
Power Stop Entry Status in PMSTAT.

typedef enum *_smc_run_mode* smc_run_mode_t
Run mode definition.

typedef enum *_smc_stop_mode* smc_stop_mode_t
Stop mode definition.

typedef enum *_smc_partial_stop_mode* smc_partial_stop_option_t
Partial STOP option.

typedef enum *_smc_reset_source* smc_reset_source_t
System Reset Source Name definitions.

typedef enum *_smc_interrupt_enable* smc_interrupt_enable_t
System reset interrupt enable bit definitions.

typedef struct *_smc_reset_pin_filter_config* smc_reset_pin_filter_config_t
Reset pin filter configuration.

struct _smc_reset_pin_filter_config
#include <fsl_msmc.h> Reset pin filter configuration.

Public Members

uint8_t slowClockFilterCount
Reset pin bus clock filter width from 1 to 32 slow clock cycles.

bool enableFilter
Reset pin filter enable/disable.

2.48 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)

2.49 PMC0: Power Management Controller

static inline void PMC0_ConfigureHsrunMode(const *pmc0_hsrun_mode_config_t* *config)

Configure the HSRUN power mode.

This function configures the HSRUN power mode, including the core regulator voltage Level setting, enable forward bias or not.

Parameters

- config – Low-Voltage detect configuration structure.

static inline void PMC0_ConfigureRunMode(const *pmc0_run_mode_config_t* *config)

Configure the RUN power mode.

This function configures the RUN power mode, including the core regulator voltage Level setting.

Parameters

- config – Low-Voltage detect configuration structure.

static inline void PMC0_ConfigureVlprMode(const *pmc0_vlpr_mode_config_t* *config)

Configure the VLPR power mode.

This function configures the VLPR power mode, including the core regulator voltage Level setting, enable reverse back bias or not, turn on force HP band-gap or not, select of HVD/LVD monitor and select of core/array regulator.

Parameters

- config – Low-Voltage detect configuration structure.

static inline void PMC0_ConfigureStopMode(const *pmc0_stop_mode_config_t* *config)

Configure the STOP power mode.

This function configures the STOP power mode, including the core regulator voltage Level setting.

Parameters

- config – Low-Voltage detect configuration structure.

static inline void PMC0_ConfigureVlpsMode(const *pmc0_vlps_mode_config_t* *config)

Configure the VLPS power mode.

This function configures the VLPS power mode, including the core regulator voltage Level setting, enable reverse back bias or not, turn on force HP band-gap or not, select of HVD/LVD monitor and select of core/array regulator.

Parameters

- config – Low-Voltage detect configuration structure.

static inline void PMC0_ConfigureLlsMode(const *pmc0_lls_mode_config_t* *config)

Configure the LLS power mode.

This function configures the LLS power mode, including the core regulator voltage Level setting, enable reverse back bias or not, turn on force HP band-gap or not, select of HVD/LVD monitor and select of core/array regulator.

Parameters

- config – Low-Voltage detect configuration structure.

static inline void PMC0_ConfigureVllsMode(const *pmc0_vlls_mode_config_t* *config)

Configure the VLLS power mode.

This function configures the VLLS power mode, including turn on force HP band-gap or not, select of HVD/LVD monitor and select of core/array regulator.

The select of array regulator is different from the other mode configurations. PMC 0 VLLS config has three array regulator select options where the others have only the latter two, see pmc0_vlls_array_regulator_select_t. Three array regulator select options in PMC 0 VLLS config are shown below:

- Regulator is off (diffrentiator)
- LP Regulator is on
- HP Regulator is on

Parameters

- config – Low-Voltage detect configuration structure.

static inline uint32_t PMC0_GetPMC0PowerModeStatusFlags(void)

Get current power mode of PMC 0.

```
if(kPMC0_HSRUNModeStatusFlags == PMC0_GetPMC0PowerModeStatusFlags(void))
{
    ...
}
```

Returns

PMC 0 current power mode status flags in the _pmc0_power_mode_status_flags.

static inline bool PMC0_GetPMC0PowerTransitionStatus(void)

Get the status of PMC 0 power mode transition.

Returns

If return 'true', which means PMC 0 is in a power mode transition. If return 'false', which means PMC 0 is not in a power mode transition.

static inline uint32_t PMC0_GetPMC1PowerModeStatusFlags(void)

Get current power mode of PMC 1.

```
if(kPMC0_HSRUNModeStatusFlags == PMC0_GetPMC1PowerModeStatusFlags(void))
{
    ...
}
```

Returns

PMC 1 current power mode status flags in the _pmc0_power_mode_status_flags.

static inline bool PMC0_GetPMC1PowerTransitionStatus(void)

Get the status of PMC 1 power mode transition.

Returns

If return 'true', which means PMC 1 is in a power mode transition. If return 'false', which means PMC 1 is not in a power mode transition.

static inline uint32_t PMC0_GetStatusFlags(void)

Gets PMC 0 status flags.

This function gets all PMC 0 status flags. The flags are returned as the logical OR value of the enumerators _pmc0_status_flags. To check for a specific status, compare the return value with enumerators in the _pmc0_status_flags. For example, to check whether core regulator voltage level is changing:

```
if (kPMC0_CoreRegulatorVoltLevelFlag & PMC0_GetStatusFlags(void))
{
    ...
}
```

Returns

PMC 0 status flags which are ORed by the enumerators in the _pmc0_status_flags.

static inline void PMC0_EnableLowVoltDetectInterrupt(void)

Enables the 1.2V Low-Voltage Detector interrupt.

This function enables the 1.2V Low-Voltage Detector interrupt.

static inline void PMC0_DisableLowVoltDetectInterrupt(void)

Disables the 1.2V Low-Voltage Detector interrupt.

This function disables the 1.2V Low-Voltage Detector interrupt.

static inline void PMC0_ClearLowVoltDetectFlag(void)

Clears the 1.2V Low-Voltage Detector flag.

This function enables the 1.2V Low-Voltage Detector flag.

static inline void PMC0_EnableHighVoltDetectInterrupt(void)

Enables the 1.8V High-Voltage Detector interrupt.

This function enables the 1.8V High-Voltage Detector interrupt.

static inline void PMC0_DisableHighVoltDetectInterrupt(void)

Disables the 1.8V High-Voltage Detector interrupt.

This function disables the 1.8V High-Voltage Detector interrupt.

static inline void PMC0_ClearHighVoltDetectFlag(void)

Clears the 1.8V High-Voltage Detector flag.

This function enables the 1.8V High-Voltage Detector flag.

static inline void PMC0_EnableLowVoltDetectReset(bool enable)

Enables the 1.2V Low-Voltage Detector reset.

This function enables 1.2V Low-Voltage Detector reset.

Parameters

- enable – Switcher of 1.2V Low-Voltage Detector reset feature. “true” means to enable, “false” means not.

static inline void PMC0_EnableHighVoltDetectReset(bool enable)

Enables the 1.8V High-Voltage Detector reset.

This function enables 1.8V High-Voltage Detector reset.

Parameters

- enable – Switcher of 1.8V High-Voltage Detector reset feature. “true” means to enable, “false” means not.

static inline void PMC0_ClearPadsIsolation(void)

Releases/clears the isolation in the PADS.

This function releases/clears the isolation in the PADS.

The isolation in the pads only will be asserted during LLS/VLLS. On LLS exit, the isolation will release automatically. ISOACK must be set after a VLLS to RUN mode transition has completed.

static inline void PMC0_PowerOnPmc1(void)

Powers on PMC 1.

This function powers on PMC 1.

When this bit field is asserted the PMC 1 is powered on. This bit would take action only once. This bit will be rearmed after a POR event only. NOTE: USB PHY-related interrupt (NVIC/GIC) and wake-up channels (AWIC/WKPU) must be disabled before turning PMC1 on.

static inline void PMC0_EnableWaitLdoOkSignal(bool enable)

Enables to wait LDO OK signal.

This function enables to wait LDO OK signal.

Parameters

- enable – Switcher of wait LDO OK signal feature. “true” means to enable, “false” means not.

static inline void PMC0_EnablePmc1LdoRegulator(bool enable)

Enables PMC 1 LDO Regulator.

This function enables PMC 1 LDO Regulator.

Parameters

- enable – Switcher of PMC 1 LDO Regulator. “true” means to enable, “false” means not.

static inline void PMC0_EnablePmc1RBBMode(bool enable)

Enable the PMC 1 RBB(reverse back bias) mode.

This function enables PMC1 RBB mode. Since this circuit when enabled has current consumption. It is recommended to use it just in high temperatures when the leakage reduction is much higher than the current consumption.

Parameters

- enable – Switcher of PMC1 RBB mode. “true” means to enable, “false” means disable.

static inline void PMC0_SetBiasConfig(const *pmc0_bias_config_t* *config)

Configure the PMC 0 bias voltage level and enable/disable pull-down.

This function change the RBB&FBB voltage level and RBB pull-down.

Parameters

- config – PMC 0 bias configuration structure.

static inline void PMC0_ConfigureSramBankPowerDown(uint32_t bankMask)

Configures PMC 0 SRAM bank power down.

This function configures PMC 0 SRAM bank power down.

The bit i controls the power mode of the PMC 0 SRAM bank i. PMC0_SRAM_PD[i] = 1’b0 - PMC 0 SRAM bank i is not affected. PMC0_SRAM_PD[i] = 1’b1 - PMC 0 SRAM bank i is in ASD or ARRAY_SHUTDOWN during all modes, except VLLS. During VLLS is in POWER_DOWN mode.

Example: Enable band 0 and 1 in ASD or ARRAY_SHUTDOWN during all modes except VLLS

```
PMC0_ConfigureSramBankPowerDown(0x3U);
```

Parameters

- bankMask – The bands to enable. Logical OR of all bits of band index to enbale.

static inline void PMC0_ConfigureSramBankPowerDownStopMode(uint32_t bankMask)

Configures PMC 0 SRAM bank power down in stop modes.

This function configures PMC 0 SRAM bank power down in stop modes.

The bit i controls the PMC 0 SRAM bank i. PMC0_SRAM_PDS[i] = 1'b0 - PMC 0 SRAM bank i is not affected. PMC0_SRAM_PDS[i] = 1'b1 - PMC 0 SRAM bank i is in ASD or ARRAY_SHUTDOWN mode during STOP, VLPS and LLS modes. During VLLS is in POWER_DOWN mode.

Example: Enable band 0 and 1 in ASD or ARRAY_SHUTDOWN during STOP, VLPS and LLS modes

```
PMC0_ConfigureSramBankPowerDownStopMode(0x3U);
```

Parameters

- bankMask – The bands to enable. Logical OR of all bits of band index to enbale.

static inline void PMC0_ConfigureSramBankPowerDownStandbyMode(uint32_t bankMask)

Configures PMC 0 SRAM bank power down in Standby Mode.

This function configures PMC 0 SRAM bank power down in Standby Mode.

The bit i controls the PMC 0 SRAM bank i. PMC0_SRAM_STDY[i] = 1'b0 - PMC 0 SRAM bank i is not affected. PMC0_SRAM_STDY[i] = 1'b1 - PMC 0 SRAM bank i is in STANDBY mode during all modes (except VLLS and LLS).

Example: Enable band 0 and 1 in STANDBY mode except VLLS and LLS

```
PMC0_ConfigureSramBankPowerDownStandbyMode(0x3U);
```

Parameters

- bankMask – The bands to enable. Logical OR of all bits of band index to enbale.

static inline void PMC0_EnableTemperatureSensor(bool enable)

Enable/disable internal temperature sensor.

Parameters

- enable – Used to enable/disable internal temperature sensor.
  - **true** Enable internal temperature sensor.
  - **false** Disable internal temperature sensor.

static inline void PMC0_SetTemperatureSensorMode(uint8_t mode)

Set temperature sensor mode.

Parameters

- mode – The temperature sensor mode to set.

FSL_PMC0_DRIVER_VERSION

PMC 0 driver version.

enum _pmc0_high_volt_detect_monitor_select

High Voltage Detect Monitor Select.

Values:

enumerator kPMC0_HighVoltDetectLowPowerMonitor
LP monitor is selected.

enumerator kPMC0_HighVoltDetectHighPowerMonitor
HP monitor is selected.

enum _pmc0_low_volt_detect_monitor_select
Low Voltage Detect Monitor Select.

Values:

enumerator kPMC0_LowVoltDetectLowPowerMonitor
LP monitor is selected.

enumerator kPMC0_LowVoltDetectHighPowerMonitor
HP monitor is selected.

enum _pmc0_core_regulator_select
Core Regulator Select.

Values:

enumerator kPMC0_CoreLowPowerRegulator
Core LP regulator is selected.

enumerator kPMC0_CoreHighPowerRegulator
Core HP regulator is selected.

enum _pmc0_array_regulator_select
Array Regulator Select.

Values:

enumerator kPMC0_ArrayLowPowerRegulator
Array LP regulator is selected.

enumerator kPMC0_ArrayHighPowerRegulator
Array HP regulator is selected.

enum _pmc0_vlls_array_regulator_select
VLLS mode array Regulator Select.

Values:

enumerator kPMC0_VllsArrayRegulatorOff
Array regulator is selected OFF. This is selectable only for VLLS mode.

enumerator kPMC0_VllsArrayLowPowerRegulator
Array LP regulator is selected.

enumerator kPMC0_VllsArrayHighPowerRegulator
Array HP regulator is selected.

enum _pmc0_fbb_p_well_voltage_level_select
FBB P-Well voltage level select.

Values:

enumerator kPMC0_FbbPWellNoBiasCondition
No BIAS condition is selected.

enumerator kPMC0_FbbPWellVoltageLevelAt50Mv
Voltage level at 50mV is selected.

enumerator kPMC0_FbbPWellVoltageLevelAt150Mv
Voltage level at 150mV is selected.

enumerator kPMC0_FbbPWellVoltageLevelAt100Mv
Voltage level at 100mV is selected.

enumerator kPMC0_FbbPWellVoltageLevelAt350Mv
Voltage level at 350mV is selected.

enumerator kPMC0_FbbPWellVoltageLevelAt300Mv
Voltage level at 300mV is selected.

enumerator kPMC0_FbbPWellVoltageLevelAt200Mv
Voltage level at 200mV is selected.

enumerator kPMC0_FbbPWellVoltageLevelAt250Mv
Voltage level at 250mV is selected.

enum _pmc0_fbb_n_well_voltage_level_select
FBB N-Well voltage level select.

Values:

enumerator kPMC0_FbbNWellNoBiasCondition
No BIAS condition is selected.

enumerator kPMC0_FbbNWellVoltageLevelAtMinus50Mv
Voltage level at -50mV is selected.

enumerator kPMC0_FbbNWellVoltageLevelAtMinus150Mv
Voltage level at -150mV is selected.

enumerator kPMC0_FbbNWellVoltageLevelAtMinus100Mv
Voltage level at -100mV is selected.

enumerator kPMC0_FbbNWellVoltageLevelAtMinus350Mv
Voltage level at -350mV is selected.

enumerator kPMC0_FbbNWellVoltageLevelAtMinus300Mv
Voltage level at -300mV is selected.

enumerator kPMC0_FbbNWellVoltageLevelAtMinus200Mv
Voltage level at -200mV is selected.

enumerator kPMC0_FbbNWellVoltageLevelAtMinus250Mv
Voltage level at -250mV is selected.

enum _pmc0_rbb_p_well_voltage_level_select
RBB P-Well voltage level select.

Values:

enumerator kPMC0_RBBPWellVoltageLevelAtMinus0_5V
Voltage level at -0.5V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus0_6V
Voltage level at -0.6V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus0_7V
Voltage level at -0.7V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus0_8V
Voltage level at -0.8V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus0_9V
Voltage level at -0.9V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus1_0V
Voltage level at -1.0V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus1_1V
Voltage level at -1.1V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus1_2V
Voltage level at -1.2V is selected.

enumerator kPMC0_RBBPWellVoltageLevelAtMinus1_3V
Voltage level at -1.3V is selected.

enum _pmc0_rbb_n_well_voltage_level_select
RBB N-Well voltage level select.

Values:

enumerator kPMC0_RBBNWellVoltageLevelAt0_5V
Voltage level at 0.5V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt0_6V
Voltage level at 0.6V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt0_7V
Voltage level at 0.7V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt0_8V
Voltage level at 0.8V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt0_9V
Voltage level at 0.9V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt1_0V
Voltage level at 1.0V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt1_1V
Voltage level at 1.1V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt1_2V
Voltage level at 1.2V is selected.

enumerator kPMC0_RBBNWellVoltageLevelAt1_3V
Voltage level at 1.3V is selected.

enum _pmc0_status_flags
PMC 0 status flags.

Values:

enumerator kPMC0_LowVoltDetectEventFlag
1.2V Low-Voltage Detector Flag, sets when low-voltage event was detected.

enumerator kPMC0_LowVoltDetectValueFlag
1.2V Low-Voltage Detector Value, sets when current value of the 1.2V LVD monitor output is 1.

enumerator kPMC0_HighVoltDetectEventFlag
1.8V High-Voltage Detector Flag, sets when high-voltage event was detected.

enumerator kPMC0_HighVoltDetectValueFlag

1.8V High-Voltage Detector Value, sets when current value of the 1.8V HVD monitor output is 1.

enumerator kPMC0_CoreRegulatorVoltLevelFlag

Core Regulator Voltage Level Flag, sets when core regulator voltage level is changing (not stable).

enumerator kPMC0_SramFlag

SRAM Flag, sets when a change mode request is being processed in the SRAMs.

enumerator kPMC0_PMC1VoltageSourceFlag

This flag indicates what is the voltage source selected to supply the PMC 1 and where the sense point of the PMC 1's LVD/HVD is placed. '0' means internal LDO supplies the PMC 1. '1' means external PMIC supplies the PMC 1.

enum _pmc0_power_mode_status_flags

PMC 0 power mode status flags.

Values:

enumerator kPMC0_HSRUNModeStatusFlags

The PMC 0 is in HSRUN mode.

enumerator kPMC0_RUNModeStatusFlags

The PMC 0 is in RUN mode.

enumerator kPMC0_STOPModeStatusFlags

The PMC 0 is in STOP mode.

enumerator kPMC0_VLPRModeStatusFlags

The PMC 0 is in VLPR mode.

enumerator kPMC0_VLPSModeStatusFlags

The PMC 0 is in VLPS mode.

enumerator kPMC0_LLSModeStatusFlags

The PMC 0 is in LLSM mode.

enumerator kPMC0_VLLSModeStatusFlags

The PMC 0 is in VLLS mode.

typedef enum *_pmc0_high_volt_detect_monitor_select* pmc0_high_volt_detect_monitor_select_t

High Voltage Detect Monitor Select.

typedef enum *_pmc0_low_volt_detect_monitor_select* pmc0_low_volt_detect_monitor_select_t

Low Voltage Detect Monitor Select.

typedef enum *_pmc0_core_regulator_select* pmc0_core_regulator_select_t

Core Regulator Select.

typedef enum *_pmc0_array_regulator_select* pmc0_array_regulator_select_t

Array Regulator Select.

typedef enum *_pmc0_vlls_array_regulator_select* pmc0_vlls_array_regulator_select_t

VLLS mode array Regulator Select.

typedef enum *_pmc0_fbb_p_well_voltage_level_select* pmc0_fbb_p_well_voltage_level_select_t

FBB P-Well voltage level select.

typedef enum *_pmc0_fbb_n_well_voltage_level_select* pmc0_fbb_n_well_voltage_level_select_t

FBB N-Well voltage level select.

typedef enum *_pmc0_rbb_p_well_voltage_level_select* pmc0_rbb_p_well_voltage_level_select_t
RBB P-Well voltage level select.

typedef enum *_pmc0_rbb_n_well_voltage_level_select* pmc0_rbb_n_well_voltage_level_select_t
RBB N-Well voltage level select.

typedef struct *_pmc0_hsrun_mode_config* pmc0_hsrun_mode_config_t
PMC 0 HSRUN mode configuration.

typedef struct *_pmc0_run_mode_config* pmc0_run_mode_config_t
PMC 0 RUN mode configuration.

typedef struct *_pmc0_vlpr_mode_config* pmc0_vlpr_mode_config_t
PMC 0 VLPR mode configuration.

typedef struct *_pmc0_stop_mode_config* pmc0_stop_mode_config_t
PMC 0 STOP mode configuration.

typedef struct *_pmc0_vlps_mode_config* pmc0_vlps_mode_config_t
PMC 0 VLPS mode configuration.

typedef struct *_pmc0_lls_mode_config* pmc0_lls_mode_config_t
PMC 0 LLS mode configuration.

typedef struct *_pmc0_vlls_mode_config* pmc0_vlls_mode_config_t
PMC 0 VLLS mode configuration.

typedef struct *_pmc0_bias_config* pmc0_bias_config_t
PMC 0 bias configuration.

FSL_COMPONENT_ID

CORE_REGULATOR_VOLT_LEVEL_MAX
MAX valid values of Core Regulator Voltage Level.

struct _pmc0_hsrun_mode_config
#include <fsl_pmc0.h> PMC 0 HSRUN mode configuration.

Public Members

uint32_t ___pad0___
Reserved.

uint32_t coreRegulatorVoltLevel
Core Regulator Voltage Level.

uint32_t ___pad1___
Reserved.

uint32_t enableForwardBias
Enable forward bias.

uint32_t ___pad2___
Reserved.

struct _pmc0_run_mode_config
#include <fsl_pmc0.h> PMC 0 RUN mode configuration.

Public Members

uint32_t ___pad0___
Reserved.

uint32_t coreRegulatorVoltLevel
Core Regulator Voltage Level.

uint32_t ___pad1___
Reserved.

struct __pmc0__vlpr__mode__config
#include <fsl_pmc0.h> PMC 0 VLPR mode configuration.

Public Members

uint32_t arrayRegulatorSelect
Array Regulator Select. pmc0_array_regulator_select_t

uint32_t ___pad0___
Reserved.

uint32_t coreRegulatorSelect
Core Regulator Select. pmc0_core_regulator_select_t

uint32_t ___pad1___
Reserved.

uint32_t lvdMonitorSelect
1.2V LVD Monitor Select. pmc0_low_volt_detect_monitor_select_t

uint32_t hvdMonitorSelect
1.2V HVD Monitor Select. pmc0_high_volt_detect_monitor_select_t

uint32_t ___pad2___
Reserved.

uint32_t enableForceHpBandgap
Enable force HP band-gap.

uint32_t ___pad3___
Reserved.

uint32_t coreRegulatorVoltLevel
Core Regulator Voltage Level.

uint32_t ___pad4___
Reserved.

uint32_t enableReverseBackBias
Enable reverse back bias.

uint32_t ___pad5___
Reserved.

struct __pmc0__stop__mode__config
#include <fsl_pmc0.h> PMC 0 STOP mode configuration.

Public Members

uint32_t ___pad0___
Reserved.

uint32_t coreRegulatorVoltLevel
Core Regulator Voltage Level.

uint32_t ___pad1___
Reserved.

struct __pmc0__vlps__mode__config
#include <fsl_pmc0.h> PMC 0 VLPS mode configuration.

Public Members

uint32_t arrayRegulatorSelect
Array Regulator Select. pmc0_array_regulator_select_t

uint32_t ___pad0___
Reserved.

uint32_t coreRegulatorSelect
Core Regulator Select. pmc0_core_regulator_select_t

uint32_t ___pad1___
Reserved.

uint32_t lvdMonitorSelect
1.2V LVD Monitor Select. pmc0_low_volt_detect_monitor_select_t

uint32_t hvdMonitorSelect
1.2V HVD Monitor Select. pmc0_high_volt_detect_monitor_select_t

uint32_t ___pad2___
Reserved.

uint32_t enableForceHpBandgap
Enable force HP band-gap.

uint32_t ___pad3___
Reserved.

uint32_t coreRegulatorVoltLevel
Core Regulator Voltage Level.

uint32_t ___pad4___
Reserved.

uint32_t enableReverseBackBias
Enable reverse back bias.

uint32_t ___pad5___
Reserved.

struct __pmc0__lls__mode__config
#include <fsl_pmc0.h> PMC 0 LLS mode configuration.

Public Members

uint32_t arrayRegulatorSelect
: Array Regulator Select. pmc0_array_regulator_select_t

uint32_t ___pad0___
: Reserved.

uint32_t coreRegulatorSelect
: Core Regulator Select. pmc0_core_regulator_select_t

uint32_t ___pad1___
: Reserved.

uint32_t lvdMonitorSelect
: 1.2V LVD Monitor Select. pmc0_low_volt_detect_monitor_select_t

uint32_t hvdMonitorSelect
: 1.2V HVD Monitor Select. pmc0_high_volt_detect_monitor_select_t

uint32_t ___pad2___
: Reserved.

uint32_t enableForceHpBandgap
: Enable force HP band-gap.

uint32_t ___pad3___
: Reserved.

uint32_t coreRegulatorVoltLevel
: Core Regulator Voltage Level.

uint32_t ___pad4___
: Reserved.

uint32_t enableReverseBackBias
: Enable reverse back bias.

uint32_t ___pad5___
: Reserved.

struct __pmc0__vlls__mode__config
: *#include <fsl_pmc0.h>* PMC 0 VLLS mode configuration.

Public Members

uint32_t arrayRegulatorSelect
: Array Regulator Select. pmc0_vlls_array_regulator_select_t

uint32_t ___pad0___
: Reserved.

uint32_t lvdMonitorSelect
: 1.2V LVD Monitor Select. pmc0_low_volt_detect_monitor_select_t

uint32_t hvdMonitorSelect
: 1.2V HVD Monitor Select. pmc0_high_volt_detect_monitor_select_t

uint32_t ___pad1___
: Reserved.

uint32_t enableForceHpBandgap
: Enable force HP band-gap.

uint32_t ___pad2___
: Reserved.

struct _pmc0_bias_config
: *#include <fsl_pmc0.h>* PMC 0 bias configuration.

Public Members

uint32_t ___pad0___
: Reserved.

uint32_t RBBPWellVoltageLevelSelect
: Select PMC0 RBB P-Well voltage level. pmc0_rbb_p_well_voltage_level_select_t

uint32_t ___pad1___
: Reserved.

uint32_t DisableRBBPullDown
: Disable RBB pull-down. '1' means to disable pull-down. '0' means to enable pull-down.

uint32_t FBBNWellVoltageLevelSelect
: Select PMC0 FBB N-Well voltage level. pmc0_fbb_n_well_voltage_level_select_t

uint32_t ___pad2___
: Reserved.

uint32_t FBBPWellVoltageLevelSelect
: Select PMC0 FBB P-Well voltage level. pmc0_fbb_p_well_voltage_level_select_t

uint32_t ___pad3___
: Reserved.

2.50 PORT: Port Control and Interrupts

static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const *port_pin_config_t* *config)

Sets the port PCR register.

This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.

- config – PORT PCR register configuration structure.

static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const *port_pin_config_t* *config)

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```
Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

static inline void PORT_SetMultipleInterruptPinsConfig(PORT_Type *base, uint32_t mask, *port_interrupt_t* config)

Sets the port interrupt configuration in PCR register for multiple pins.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT pin interrupt configuration.
  - kPORT_InterruptOrDMADisabled: Interrupt/DMA request disabled.
  - kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kPORT_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kPORT_InterruptLogicZero : Interrupt when logic zero.
  - kPORT_InterruptRisingEdge : Interrupt on rising edge.
  - kPORT_InterruptFallingEdge: Interrupt on falling edge.
  - kPORT_InterruptEitherEdge : Interrupt on either edge.
  - kPORT_InterruptLogicOne : Interrupt when logic one.
  - kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).

– kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

static inline void PORT_SetPinMux(PORT_Type *base, uint32_t pin, *port_mux_t* mux)

Configures the pin muxing.

Note: : This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

---

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- mux – pin muxing slot selection.
  - kPORT_PinDisabledOrAnalog: Pin disabled or work in analog function.
  - kPORT_MuxAsGpio : Set as GPIO.
  - kPORT_MuxAlt2 : chip-specific.
  - kPORT_MuxAlt3 : chip-specific.
  - kPORT_MuxAlt4 : chip-specific.
  - kPORT_MuxAlt5 : chip-specific.
  - kPORT_MuxAlt6 : chip-specific.
  - kPORT_MuxAlt7 : chip-specific.

static inline void PORT_EnablePinsDigitalFilter(PORT_Type *base, uint32_t mask, bool enable)

Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

static inline void PORT_SetDigitalFilterConfig(PORT_Type *base, const *port_digital_filter_config_t* *config)

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

static inline void PORT_SetPinInterruptConfig(PORT_Type *base, uint32_t pin, *port_interrupt_t* config)

Configures the port pin interrupt/DMA request.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT pin interrupt configuration.
  - kPORT_InterruptOrDMADisabled: Interrupt/DMA request disabled.

- kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
- kPORT_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
- kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
- kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
- kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
- kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
- kPORT_InterruptLogicZero : Interrupt when logic zero.
- kPORT_InterruptRisingEdge : Interrupt on rising edge.
- kPORT_InterruptFallingEdge: Interrupt on falling edge.
- kPORT_InterruptEitherEdge : Interrupt on either edge.
- kPORT_InterruptLogicOne : Interrupt when logic one.
- kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
- kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

static inline void PORT_SetPinDriveStrength(PORT_Type *base, uint32_t pin, uint8_t strength)

Configures the port pin drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- strength – PORT pin drive strength
  - kPORT_LowDriveStrength = 0U - Low-drive strength is configured.
  - kPORT_HighDriveStrength = 1U - High-drive strength is configured.

static inline uint32_t PORT_GetPinsInterruptFlags(PORT_Type *base)

Reads the whole port status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

- base – PORT peripheral base pointer.

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

static inline void PORT_ClearPinsInterruptFlags(PORT_Type *base, uint32_t mask)

Clears the multiple pin interrupt status flag.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.

FSL_PORT_DRIVER_VERSION
PORT driver version.

enum _port_pull
Internal resistor pull feature selection.

Values:

enumerator kPORT_PullDisable
Internal pull-up/down resistor is disabled.

enumerator kPORT_PullDown
Internal pull-down resistor is enabled.

enumerator kPORT_PullUp
Internal pull-up resistor is enabled.

enum _port_slew_rate
Slew rate selection.

Values:

enumerator kPORT_FastSlewRate
Fast slew rate is configured.

enumerator kPORT_SlowSlewRate
Slow slew rate is configured.

enum _port_open_drain_enable
Open Drain feature enable/disable.

Values:

enumerator kPORT_OpenDrainDisable
Open drain output is disabled.

enumerator kPORT_OpenDrainEnable
Open drain output is enabled.

enum _port_passive_filter_enable
Passive filter feature enable/disable.

Values:

enumerator kPORT_PassiveFilterDisable
Passive input filter is disabled.

enumerator kPORT_PassiveFilterEnable
Passive input filter is enabled.

enum _port_drive_strength
Configures the drive strength.

Values:

enumerator kPORT_LowDriveStrength
Low-drive strength is configured.

enumerator kPORT_HighDriveStrength
High-drive strength is configured.

enum _port_lock_register
Unlock/lock the pin control register field[15:0].

Values:

enumerator kPORT_UnlockRegister
Pin Control Register fields [15:0] are not locked.

enumerator kPORT_LockRegister
Pin Control Register fields [15:0] are locked.

enum _port_mux
Pin mux selection.

Values:

enumerator kPORT_PinDisabledOrAnalog
Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT_MuxAsGpio
Corresponding pin is configured as GPIO.

enumerator kPORT_MuxAlt0
Chip-specific

enumerator kPORT_MuxAlt1
Chip-specific

enumerator kPORT_MuxAlt2
Chip-specific

enumerator kPORT_MuxAlt3
Chip-specific

enumerator kPORT_MuxAlt4
Chip-specific

enumerator kPORT_MuxAlt5
Chip-specific

enumerator kPORT_MuxAlt6
Chip-specific

enumerator kPORT_MuxAlt7
Chip-specific

enumerator kPORT_MuxAlt8
Chip-specific

enumerator kPORT_MuxAlt9
Chip-specific

enumerator kPORT_MuxAlt10
Chip-specific

enumerator kPORT_MuxAlt11
Chip-specific

enumerator kPORT_MuxAlt12
Chip-specific

enumerator kPORT_MuxAlt13
Chip-specific

enumerator kPORT_MuxAlt14
Chip-specific

enumerator kPORT_MuxAlt15
Chip-specific

enum _port_interrupt
Configures the interrupt generation condition.

Values:

enumerator kPORT_InterruptOrDMADisabled
Interrupt/DMA request is disabled.

enumerator kPORT_DMARisingEdge
DMA request on rising edge.

enumerator kPORT_DMAFallingEdge
DMA request on falling edge.

enumerator kPORT_DMAEitherEdge
DMA request on either edge.

enumerator kPORT_FlagRisingEdge
Flag sets on rising edge.

enumerator kPORT_FlagFallingEdge
Flag sets on falling edge.

enumerator kPORT_FlagEitherEdge
Flag sets on either edge.

enumerator kPORT_InterruptLogicZero
Interrupt when logic zero.

enumerator kPORT_InterruptRisingEdge
Interrupt on rising edge.

enumerator kPORT_InterruptFallingEdge
Interrupt on falling edge.

enumerator kPORT_InterruptEitherEdge
Interrupt on either edge.

enumerator kPORT_InterruptLogicOne
Interrupt when logic one.

enumerator kPORT_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kPORT_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

enum _port_digital_filter_clock_source
Digital filter clock source selection.

Values:

enumerator kPORT_BusClock
Digital filters are clocked by the bus clock.

enumerator kPORT_LpoClock
Digital filters are clocked by the 1 kHz LPO clock.

typedef enum *_port_mux* port_mux_t
Pin mux selection.

typedef enum *_port_interrupt* port_interrupt_t
: Configures the interrupt generation condition.

typedef enum *_port_digital_filter_clock_source* port_digital_filter_clock_source_t
: Digital filter clock source selection.

typedef struct *_port_digital_filter_config* port_digital_filter_config_t
: PORT digital filter feature configuration definition.

typedef struct *_port_pin_config* port_pin_config_t
: PORT pin configuration structure.

FSL_COMPONENT_ID

struct _port_digital_filter_config
: *#include <fsl_port.h>* PORT digital filter feature configuration definition.

Public Members

uint32_t digitalFilterWidth
: Set digital filter width

port_digital_filter_clock_source_t clockSource
: Set digital filter clockSource

struct _port_pin_config
: *#include <fsl_port.h>* PORT pin configuration structure.

Public Members

uint16_t pullSelect
: No-pull/pull-down/pull-up select

uint16_t slewRate
: Fast/slow slew rate Configure

uint16_t passiveFilterEnable
: Passive filter enable/disable

uint16_t openDrainEnable
: Open drain enable/disable

uint16_t driveStrength
: Fast/slow drive strength configure

uint16_t lockRegister
: Lock/unlock the PCR field[15:0]

2.51 QSPI: Quad Serial Peripheral Interface

2.52 Quad Serial Peripheral Interface Driver

uint32_t QSPI_GetInstance(QuadSPI_Type *base)

Get the instance number for QSPI.

Parameters

- base – QSPI base pointer.

void QSPI_Init(QuadSPI_Type *base, *qspi_config_t* *config, uint32_t srcClock_Hz)

Initializes the QSPI module and internal state.

This function enables the clock for QSPI and also configures the QSPI with the input configure parameters. Users should call this function before any QSPI operations.

Parameters

- base – Pointer to QuadSPI Type.
- config – QSPI configure structure.
- srcClock_Hz – QSPI source clock frequency in Hz.

void QSPI_GetDefaultQspiConfig(*qspi_config_t* *config)

Gets default settings for QSPI.

Parameters

- config – QSPI configuration structure.

void QSPI_Deinit(QuadSPI_Type *base)

Deinitializes the QSPI module.

Clears the QSPI state and QSPI module registers.

Parameters

- base – Pointer to QuadSPI Type.

void QSPI_SetFlashConfig(QuadSPI_Type *base, *qspi_flash_config_t* *config)

Configures the serial flash parameter.

This function configures the serial flash relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the QSPI features.

Parameters

- base – Pointer to QuadSPI Type.
- config – Flash configuration parameters.

void QSPI_SetDelayChainConfig(QuadSPI_Type *base, *qspi_delay_chain_config_t* *config)

Configures the delay chain parameter.

This function configures the slave delay chain.

Parameters

- base – Pointer to QuadSPI Type.
- config – Delay chain configuration parameters.

void QSPI_SoftwareReset(QuadSPI_Type *base)

Software reset for the QSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – Pointer to QuadSPI Type.

static inline void QSPI_Enable(QuadSPI_Type *base, bool enable)

Enables or disables the QSPI module.

Parameters

- base – Pointer to QuadSPI Type.
- enable – True means enable QSPI, false means disable.

static inline uint32_t QSPI_GetStatusFlags(QuadSPI_Type *base)

Gets the state value of QSPI.

Parameters

- base – Pointer to QuadSPI Type.

Returns

status flag, use status flag to AND _qspi_flags could get the related status.

static inline uint32_t QSPI_GetErrorStatusFlags(QuadSPI_Type *base)

Gets QSPI error status flags.

Parameters

- base – Pointer to QuadSPI Type.

Returns

status flag, use status flag to AND _qspi_error_flags could get the related status.

static inline void QSPI_ClearErrorFlag(QuadSPI_Type *base, uint32_t mask)

Clears the QSPI error flags.

Parameters

- base – Pointer to QuadSPI Type.
- mask – Which kind of QSPI flags to be cleared, a combination of _qspi_error_flags.

static inline void QSPI_EnableInterrupts(QuadSPI_Type *base, uint32_t mask)

Enables the QSPI interrupts.

Parameters

- base – Pointer to QuadSPI Type.
- mask – QSPI interrupt source.

static inline void QSPI_DisableInterrupts(QuadSPI_Type *base, uint32_t mask)

Disables the QSPI interrupts.

Parameters

- base – Pointer to QuadSPI Type.
- mask – QSPI interrupt source.

static inline void QSPI_EnableDMA(QuadSPI_Type *base, uint32_t mask, bool enable)

Enables the QSPI DMA source.

Parameters

- base – Pointer to QuadSPI Type.
- mask – QSPI DMA source.
- enable – True means enable DMA, false means disable.

static inline uint32_t QSPI_GetTxDataRegisterAddress(QuadSPI_Type *base)

Gets the Tx data register address. It is used for DMA operation.

Parameters

- base – Pointer to QuadSPI Type.

Returns

QSPI Tx data register address.

uint32_t QSPI_GetRxDataRegisterAddress(QuadSPI_Type *base)

Gets the Rx data register address used for DMA operation.

This function returns the Rx data register address or Rx buffer address according to the Rx read area settings.

Parameters

- base – Pointer to QuadSPI Type.

Returns

QSPI Rx data register address.

static inline void QSPI_SetIPCommandAddress(QuadSPI_Type *base, uint32_t addr)

Sets the IP command address.

Parameters

- base – Pointer to QuadSPI Type.
- addr – IP command address.

static inline void QSPI_SetIPCommandSize(QuadSPI_Type *base, uint32_t size)

Sets the IP command size.

Parameters

- base – Pointer to QuadSPI Type.
- size – IP command size.

void QSPI_ExecuteIPCommand(QuadSPI_Type *base, uint32_t index)

Executes IP commands located in LUT table.

Parameters

- base – Pointer to QuadSPI Type.
- index – IP command located in which LUT table index.

void QSPI_ExecuteAHBCommand(QuadSPI_Type *base, uint32_t index)

Executes AHB commands located in LUT table.

Parameters

- base – Pointer to QuadSPI Type.
- index – AHB command located in which LUT table index.

static inline void QSPI_EnableIPParallelMode(QuadSPI_Type *base, bool enable)

Enables/disables the QSPI IP command parallel mode.

Parameters

- base – Pointer to QuadSPI Type.
- enable – True means enable parallel mode, false means disable parallel mode.

static inline void QSPI_EnableAHBParallelMode(QuadSPI_Type *base, bool enable)

Enables/disables the QSPI AHB command parallel mode.

Parameters

- base – Pointer to QuadSPI Type.
- enable – True means enable parallel mode, false means disable parallel mode.

void QSPI_UpdateLUT(QuadSPI_Type *base, uint32_t index, uint32_t *cmd)

Updates the LUT table.

Parameters

- base – Pointer to QuadSPI Type.
- index – Which LUT index needs to be located. It should be an integer divided by 4.
- cmd – Command sequence array.

static inline void QSPI_ClearFifo(QuadSPI_Type *base, uint32_t mask)

Clears the QSPI FIFO logic.

Parameters

- base – Pointer to QuadSPI Type.
- mask – Which kind of QSPI FIFO to be cleared.

static inline void QSPI_ClearCommandSequence(QuadSPI_Type *base, *qspi_command_seq_t* seq)

@ brief Clears the command sequence for the IP/buffer command.

This function can reset the command sequence.

Parameters

- base – QSPI base address.
- seq – Which command sequence need to reset, IP command, buffer command or both.

static inline void QSPI_EnableDDRMode(QuadSPI_Type *base, bool enable)

Enable or disable DDR mode.

Parameters

- base – QSPI base pointer
- enable – True means enable DDR mode, false means disable DDR mode.

void QSPI_SetReadDataArea(QuadSPI_Type *base, *qspi_read_area_t* area)

@ brief Set the RX buffer readout area.

This function can set the RX buffer readout, from AHB bus or IP Bus.

Parameters

- base – QSPI base address.
- area – QSPI Rx buffer readout area. AHB bus buffer or IP bus buffer.

void QSPI_WriteBlocking(QuadSPI_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 – QSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to send

static inline void QSPI_WriteData(QuadSPI_Type *base, uint32_t data)

Writes data into FIFO.

Parameters

- base – QSPI base pointer
- data – The data bytes to send

void QSPI_ReadBlocking(QuadSPI_Type *base, uint32_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

- base – QSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to receive

uint32_t QSPI_ReadData(QuadSPI_Type *base)

Receives data from data FIFO.

Parameters

- base – QSPI base pointer

Returns

The data in the FIFO.

static inline void QSPI_TransferSendBlocking(QuadSPI_Type *base, *qspi_transfer_t* *xfer)

Writes data to the QSPI transmit buffer.

This function writes a continuous data to the QSPI transmit FIFO. This function is a block function and can return only when finished. This function uses polling methods.

Parameters

- base – Pointer to QuadSPI Type.
- xfer – QSPI transfer structure.

static inline void QSPI_TransferReceiveBlocking(QuadSPI_Type *base, *qspi_transfer_t* *xfer)

Reads data from the QSPI receive buffer in polling way.

This function reads continuous data from the QSPI receive buffer/FIFO. This function is a blocking function and can return only when finished. This function uses polling methods. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

- base – Pointer to QuadSPI Type.
- xfer – QSPI transfer structure.

FSL_QSPI_DRIVER_VERSION
QSPI driver version.

Status structure of QSPI.

Values:

enumerator kStatus_QSPI_Idle
QSPI is in idle state

enumerator kStatus_QSPI_Busy
QSPI is busy

enumerator kStatus_QSPI_Error
Error occurred during QSPI transfer

enum _qspi_read_area
QSPI read data area, from IP FIFO or AHB buffer.

Values:

enumerator kQSPI_ReadAHB
QSPI read from AHB buffer.

enumerator kQSPI_ReadIP
QSPI read from IP FIFO.

enum _qspi_command_seq
QSPI command sequence type.

Values:

enumerator kQSPI_IPSeq
IP command sequence

enumerator kQSPI_BufferSeq
Buffer command sequence

enumerator kQSPI_AllSeq

enum _qspi_fifo
QSPI buffer type.

Values:

enumerator kQSPI_TxFifo
QSPI Tx FIFO

enumerator kQSPI_RxFifo
QSPI Rx FIFO

enumerator kQSPI_AllFifo
QSPI all FIFO, including Tx and Rx

enum _qspi_error_flags
QSPI error flags.

Values:

enumerator kQSPI_DataLearningFail
Data learning pattern failure flag

enumerator kQSPI_TxBufferFill
Tx buffer fill flag

enumerator kQSPI_TxBufferUnderrun
Tx buffer underrun flag

enumerator kQSPI_IllegalInstruction
Illegal instruction error flag

enumerator kQSPI_RxBufferOverflow
Rx buffer overflow flag

enumerator kQSPI_RxBufferDrain
Rx buffer drain flag

enumerator kQSPI_AHBSequenceError
AHB sequence error flag

enumerator kQSPI_AHBBufferOverflow
AHB buffer overflow flag

enumerator kQSPI_IPCommandUsageError
IP command usage error flag

enumerator kQSPI_IPCommandTriggerDuringAHBAccess
IP command trigger during AHB access error

enumerator kQSPI_IPCommandTriggerDuringIPAccess
IP command trigger cannot be executed

enumerator kQSPI_IPCommandTriggerDuringAHBGrant
IP command trigger during AHB grant error

enumerator kQSPI_IPCommandTransactionFinished
IP command transaction finished flag

enumerator kQSPI_FlagAll
All error flag

enum _qspi_flags
QSPI state bit.

Values:

enumerator kQSPI_DataLearningSamplePoint
Data learning sample point

enumerator kQSPI_TxBufferFull
Tx buffer full flag

enumerator kQSPI_TxBufferEnoughData
Tx buffer enough data available

enumerator kQSPI_RxDMA
Rx DMA is requesting or running

enumerator kQSPI_RxBufferFull
Rx buffer full

enumerator kQSPI_RxWatermark
Rx buffer watermark exceeded

enumerator kQSPI_AHB3BufferFull
AHB buffer 3 full

enumerator kQSPI_AHB2BufferFull
AHB buffer 2 full

enumerator kQSPI_AHB1BufferFull
AHB buffer 1 full

enumerator kQSPI_AHB0BufferFull
AHB buffer 0 full

enumerator kQSPI_AHB3BufferNotEmpty
AHB buffer 3 not empty

enumerator kQSPI_AHB2BufferNotEmpty
AHB buffer 2 not empty

enumerator kQSPI_AHB1BufferNotEmpty
AHB buffer 1 not empty

enumerator kQSPI_AHB0BufferNotEmpty
AHB buffer 0 not empty

enumerator kQSPI_AHBTransactionPending
AHB access transaction pending

enumerator kQSPI_AHBCommandPriorityGranted
AHB command priority granted

enumerator kQSPI_AHBAccess
AHB access

enumerator kQSPI_IPAccess
IP access

enumerator kQSPI_Busy
Module busy

enumerator kQSPI_StateAll
All flags

enum _qspi_interrupt_enable
QSPI interrupt enable.

Values:

enumerator kQSPI_DataLearningFailInterruptEnable
Data learning pattern failure interrupt enable

enumerator kQSPI_TxBufferFillInterruptEnable
Tx buffer fill interrupt enable

enumerator kQSPI_TxBufferUnderrunInterruptEnable
Tx buffer underrun interrupt enable

enumerator kQSPI_IllegalInstructionInterruptEnable
Illegal instruction error interrupt enable

enumerator kQSPI_RxBufferOverflowInterruptEnable
Rx buffer overflow interrupt enable

enumerator kQSPI_RxBufferDrainInterruptEnable
Rx buffer drain interrupt enable

enumerator kQSPI_AHBSequenceErrorInterruptEnable
AHB sequence error interrupt enable

enumerator kQSPI_AHBBufferOverflowInterruptEnable
AHB buffer overflow interrupt enable

enumerator kQSPI_IPCommandUsageErrorInterruptEnable
IP command usage error interrupt enable

enumerator kQSPI_IPCommandTriggerDuringAHBAccessInterruptEnable
IP command trigger during AHB access error

enumerator kQSPI_IPCommandTriggerDuringIPAccessInterruptEnable
IP command trigger cannot be executed

enumerator kQSPI_IPCommandTriggerDuringAHBGrantInterruptEnable
IP command trigger during AHB grant error

enumerator kQSPI_IPCommandTransactionFinishedInterruptEnable
IP command transaction finished interrupt enable

enumerator kQSPI_AllInterruptEnable
All error interrupt enable

enum _qspi_dma_enable
QSPI DMA request flag.

Values:

enumerator kQSPI_RxBufferDrainDMAEnable
Rx buffer drain DMA

enumerator kQSPI_AllDDMAEnable

enum _qspi_dqs_phrase_shift
Phrase shift number for DQS mode.

Values:

enumerator kQSPI_DQSNoPhraseShift
No phase shift

enumerator kQSPI_DQSPhraseShift45Degree
Select 45 degree phase shift

enumerator kQSPI_DQSPhraseShift90Degree
Select 90 degree phase shift

enumerator kQSPI_DQSPhraseShift135Degree
Select 135 degree phase shift

enum _qspi_dqs_read_sample_clock
Qspi read sampling option.

Values:

enumerator kQSPI_ReadSampleClkInternalLoopback
Read sample clock adopts internal loopback mode.

enumerator kQSPI_ReadSampleClkLoopbackFromDqsPad
Dummy Read strobe generated by QSPI Controller and loopback from DQS pad.

enumerator kQSPI_ReadSampleClkExternalInputFromDqsPad
Flash provided Read strobe and input from DQS pad.

typedef enum *_qspi_read_area* qspi_read_area_t
: QSPI read data area, from IP FIFO or AHB buffer.

typedef enum *_qspi_command_seq* qspi_command_seq_t
: QSPI command sequence type.

typedef enum *_qspi_fifo* qspi_fifo_t
: QSPI buffer type.

typedef enum *_qspi_dqs_phrase_shift* qspi_dqs_phrase_shift_t
: Phrase shift number for DQS mode.

typedef enum *_qspi_dqs_read_sample_clock* qspi_dqs_read_sample_clock_t
: Qspi read sampling option.

typedef struct *QspiDQSConfig* qspi_dqs_config_t
: DQS configure features.

typedef struct *QspiFlashTiming* qspi_flash_timing_t
: Flash timing configuration.

typedef struct *QspiConfig* qspi_config_t
: QSPI configuration structure.

typedef struct *_qspi_flash_config* qspi_flash_config_t
: External flash configuration items.

typedef struct *_qspi_transfer* qspi_transfer_t
: Transfer structure for QSPI.

typedef struct *_ip_command_config* ip_command_config_t
: 16-bit access reg for IPCR register

typedef struct *_qspi_delay_chain_config* qspi_delay_chain_config_t
: Slave delay chain configuration items.

QSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
: Macro functions for LUT table.

QSPI_CMD
: Macro for QSPI LUT command.

QSPI_ADDR

QSPI_DUMMY

QSPI_MODE

QSPI_MODE2

QSPI_MODE4

QSPI_READ

QSPI_WRITE

QSPI_JMP_ON_CS

QSPI_ADDR_DDR

QSPI_MODE_DDR

QSPI_MODE2_DDR

QSPI_MODE4_DDR

QSPI_READ_DDR

QSPI_WRITE_DDR

QSPI_DATA_LEARN

QSPI_CMD_DDR

QSPI_CADDR

QSPI_CADDR_DDR

QSPI_STOP

QSPI_PAD_1
: Macro for QSPI PAD.

QSPI_PAD_2

QSPI_PAD_4

QSPI_PAD_8

struct QspiDQSConfig
: *#include <fsl_qspi.h>* DQS configure features.

Public Members

uint32_t portADelayTapNum
: Delay chain tap number selection for QSPI port A DQS

qspi_dqs_phrase_shift_t shift
: Phase shift for internal DQS generation

qspi_dqs_read_sample_clock_t rxSampleClock
: Read sample clock for Dqs.

bool enableDQSClkInverse
: Enable inverse clock for internal DQS generation

struct QspiFlashTiming
: *#include <fsl_qspi.h>* Flash timing configuration.

Public Members

uint32_t dataHoldTime
: Serial flash data in hold time

uint32_t CSHoldTime
: Serial flash CS hold time in terms of serial flash clock cycles

uint32_t CSSetupTime
: Serial flash CS setup time in terms of serial flash clock cycles

struct QspiConfig
: *#include <fsl_qspi.h>* QSPI configuration structure.

Public Members

uint8_t txWatermark
QSPI transmit watermark value

uint8_t rxWatermark
QSPI receive watermark value.

uint32_t AHBbufferSize[1]
AHB buffer size.

uint8_t AHBbufferMaster[1]
AHB buffer master.

bool enableAHBbuffer3AllMaster
Is AHB buffer3 for all master.

qspi_read_area_t area
Which area Rx data readout

bool enableQspi
Enable QSPI after initialization

struct __qspi__flash__config
#include <fsl_qspi.h> External flash configuration items.

Public Members

uint32_t flashA1Size
Flash A1 size

uint32_t flashA2Size
Flash A2 size

uint32_t flashB1Size
Flash B1 size

uint32_t flashB2Size
Flash B2 size

uint32_t lookuptable[1]
Flash command in LUT

uint32_t CSHoldTime
CS line hold time

uint32_t CSSetupTime
CS line setup time

uint32_t cloumnspace
Column space size

uint32_t dataLearnValue
Data Learn value if enable data learn

bool enableWordAddress
If enable word address.

struct __qspi__transfer
#include <fsl_qspi.h> Transfer structure for QSPI.

Public Members

uint32_t *data
: Pointer to data to transmit

size_t dataSize
: Bytes to be transmit

struct _ip_command_config
: *#include <fsl_qspi.h>* 16-bit access reg for IPCR register

struct _qspi_delay_chain_config
: *#include <fsl_qspi.h>* Slave delay chain configuration items.

Public Members

bool highFreqDelay
: Selects delay chain for low/high frequency of operation.

union IPCR_REG

Public Members

___IO uint32_t IPCR
: IP Configuration Register

struct *_ip_command_config* BITFIELD

struct BITFIELD

Public Members

___IO uint16_t IDATZ
: 16-bit access for IDATZ field in IPCR register

___IO uint8_t RESERVED_0
: 8-bit access for RESERVED_0 field in IPCR register

___IO uint8_t SEQID
: 8-bit access for SEQID field in IPCR register

2.53 Quad Serial Peripheral Interface EDMA Driver

void QSPI_TransferTxCreateHandleEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the QSPI handle for send which is used in transactional functions and set the callback.

Parameters

- base – QSPI peripheral base address
- handle – Pointer to qspi_edma_handle_t structure
- callback – QSPI callback, NULL means no callback.
- userData – User callback function data.

- dmaHandle – User requested eDMA handle for eDMA transfer

void QSPI_TransferRxCreateHandleEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the QSPI handle for receive which is used in transactional functions and set the callback.

Parameters

- base – QSPI peripheral base address
- handle – Pointer to qspi_edma_handle_t structure
- callback – QSPI callback, NULL means no callback.
- userData – User callback function data.
- dmaHandle – User requested eDMA handle for eDMA transfer

status_t QSPI_TransferSendEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_transfer_t* *xfer)

Transfers QSPI data using an eDMA non-blocking method.

This function writes data to the QSPI transmit FIFO. This function is non-blocking.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure
- xfer – QSPI transfer structure.

status_t QSPI_TransferReceiveEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_transfer_t* *xfer)

Receives data using an eDMA non-blocking method.

This function receive data from the QSPI receive buffer/FIFO. This function is non-blocking. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure
- xfer – QSPI transfer structure.

void QSPI_TransferAbortSendEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – QSPI peripheral base address.
- handle – Pointer to qspi_edma_handle_t structure

void QSPI_TransferAbortReceiveEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle)

Aborts the receive data using eDMA.

This function abort receive data which using eDMA.

Parameters

- base – QSPI peripheral base address.

- handle – Pointer to qspi_edma_handle_t structure

status_t QSPI_TransferGetSendCountEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, size_t *count)

Gets the transferred counts of send.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure.
- count – Bytes sent.

Return values

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

status_t QSPI_TransferGetReceiveCountEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, size_t *count)

Gets the status of the receive transfer.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure
- count – Bytes received.

Return values

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

FSL_QSPI_EDMA_DRIVER_VERSION

QSPI EDMA driver version.

typedef struct *_qspi_edma_handle* qspi_edma_handle_t

typedef void (*qspi_edma_callback_t)(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *status_t* status, void *userData)

QSPI eDMA transfer callback function for finish and error.

struct _qspi_edma_handle

#include <fsl_qspi_edma.h> QSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

eDMA handler for QSPI send

size_t transferSize

Bytes need to transfer.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t count

The transfer data count in a DMA request

uint32_t state
: Internal state for QSPI eDMA transfer

qspi_edma_callback_t callback
: Callback for users while transfer finish or error occurred

void *userData
: User callback parameter

2.54 SAI: Serial Audio Interface

2.55 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 2.4.11

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy
SAI Tx is busy.

enumerator kStatus_SAI_RxBusy
SAI Rx is busy.

enumerator kStatus_SAI_TxError
SAI Tx FIFO error.

enumerator kStatus_SAI_RxError
SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull
SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle
SAI Tx is idle

enumerator kStatus_SAI_RxIdle
SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.

Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.

Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.

Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx
Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx
Synchronous with another SAI receiver

enum _sai_bclk_source
Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk
Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1
Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2
Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3
Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv
Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0
Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1
Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable
FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable
FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing
The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate

Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz
Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz
Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz
Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz
Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz
Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz
Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz
Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz
Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz
Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz
Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width

Audio word width.

Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_data_pin_state

sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine
sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled
sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead
sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite
sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite
sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead
sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite
sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type
sai transceiver type

Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

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

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol

Audio bus protocol in SAI

sai_sync_mode_t syncMode

SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource

Bit Clock source

sai_master_slave_t masterSlave

Master or slave

struct _sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz

Sample rate of audio data

uint32_t bitWidth

Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo

Mono or stereo

uint32_t masterClockHz
: Master clock frequency in Hz

uint8_t watermark
: Watermark value

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct __sai__master__clock
: *#include <fsl_sai.h>* master clock configurations

Public Members

bool mclkOutputEnable
: master clock output enable

uint32_t mclkHz
: target mclk frequency

uint32_t mclkSourceClkHz
: mclk source frequency

struct __sai__fifo
: *#include <fsl_sai.h>* sai fifo configurations

Public Members

bool fifoContinueOneError
: fifo continues when error occur

sai_fifo_combine_t fifoCombine
: fifo combine mode

sai_fifo_packing_t fifoPacking
: fifo packing mode

uint8_t fifoWatermark
: fifo watermark

struct __sai__bit__clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkSrcSwap
: bit clock source swap

bool bclkInputDelay
: bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
: bit clock polarity

sai_bclk_source_t bclkSource
: bit Clock source

struct __sai__frame__sync
: *#include <fsl_sai.h>* sai frame sync configurations

Public Members

uint8_t frameSyncWidth
: frame sync width in number of bit clocks

bool frameSyncEarly
: TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
: internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity
: frame sync polarity

struct __sai__serial__data
: *#include <fsl_sai.h>* sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
: sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
: configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
: configure the number of bits in the first word in each frame

uint8_t dataWordNLength
: configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
: used to record the data length for dma transfer

uint8_t dataFirstBitShifted
: Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
: configure the number of words in each frame

uint32_t dataMaskedWord
configure whether the transmit word is masked

struct __sai__transceiver
#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct __sai__transfer
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct __sai__handle
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
Transfer status

sai_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

uint8_t bitWidth
: Bit width for transfer, 8/16/24/32 bits

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

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

uint8_t watermark
: Watermark value

2.56 SAI EDMA Driver

void SAI_TransferTxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle)

Initializes the SAI eDMA handle.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- txDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferRxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *rxDmaHandle)

Initializes the SAI Rx eDMA handle.

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- rxDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferSetInterleaveType(*sai_edma_handle_t* *handle, *sai_edma_interleave_t* interleaveType)

Initializes the SAI interleave type.

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI_EDMAInterleavePerChannelSample always

Parameters

- handle – SAI eDMA handle pointer.
- interleaveType – Multi channel interleave type.

void SAI_TransferTxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Tx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnWrite to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

void SAI_TransferRxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Rx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;
SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

status_t SAI_TransferSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

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

Return values

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

status_t SAI_TransferReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using eDMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers

b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

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

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SAI is busy receiving data.

status_t SAI_TransferSendLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.

status_t SAI_TransferReceiveLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.

void SAI_TransferTerminateSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI send.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI receive.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI transfer using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

status_t SAI_TransferGetSendCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count sent by SAI.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- count – Bytes count sent by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

status_t SAI_TransferGetReceiveCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count received by SAI.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- count – Bytes count received by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

uint32_t SAI_TransferGetValidTransferSlotsEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Gets valid transfer slot.

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

Return values

valid – slot count that application submit.

FSL_SAI_EDMA_DRIVER_VERSION

Version 2.7.4

enum _sai_edma_interleave

sai interleave type

Values:

enumerator kSAI_EDMAInterleavePerChannelSample

enumerator kSAI_EDMAInterleavePerChannelBlock

typedef struct *sai_edma_handle* sai_edma_handle_t

typedef void (*sai_edma_callback_t)(I2S_Type *base, *sai_edma_handle_t* *handle, *status_t* status, void *userData)

SAI eDMA transfer callback function for finish and error.

typedef enum *_sai_edma_interleave* sai_edma_interleave_t

sai interleave type

MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferReceiveEDMA

MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferSendEDMA

struct sai_edma_handle

#include <fsl_sai_edma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for SAI send

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t bytesPerFrame

Bytes in a frame

uint8_t channelMask

Enabled channel mask value, reference _sai_channel_mask

uint8_t channelNums

total enabled channel nums

uint8_t channel

Which data channel

uint8_t count

The transfer data count in a DMA request

uint32_t state

Internal state for SAI eDMA transfer

sai_edma_callback_t callback

Callback for users while transfer finish or error occurs

void *userData

User callback parameter

uint8_t tcd[((4U) + 1U) * sizeof(*edma_tcd_t*)]

TCD pool for eDMA transfer.

sai_transfer_t saiQueue[(4U)]
Transfer queue storing queued transfer.

size_t transferSize[(4U)]
Data bytes need to transfer

sai_edma_interleave_t interleaveType
Transfer interleave type

volatile uint8_t queueUser
Index for user to queue transfer.

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

2.57 SEMA42: Hardware Semaphores Driver

FSL_SEMA42_DRIVER_VERSION
SEMA42 driver version.

SEMA42 status return codes.

Values:

enumerator kStatus_SEMA42_Busy
SEMA42 gate has been locked by other processor.

enumerator kStatus_SEMA42_Reseting
SEMA42 gate reseting is ongoing.

enum _sema42_gate_status
SEMA42 gate lock status.

Values:

enumerator kSEMA42_Unlocked
The gate is unlocked.

enumerator kSEMA42_LockedByProc0
The gate is locked by processor 0.

enumerator kSEMA42_LockedByProc1
The gate is locked by processor 1.

enumerator kSEMA42_LockedByProc2
The gate is locked by processor 2.

enumerator kSEMA42_LockedByProc3
The gate is locked by processor 3.

enumerator kSEMA42_LockedByProc4
The gate is locked by processor 4.

enumerator kSEMA42_LockedByProc5
The gate is locked by processor 5.

enumerator kSEMA42_LockedByProc6
The gate is locked by processor 6.

enumerator kSEMA42_LockedByProc7
The gate is locked by processor 7.

enumerator kSEMA42_LockedByProc8
The gate is locked by processor 8.

enumerator kSEMA42_LockedByProc9
The gate is locked by processor 9.

enumerator kSEMA42_LockedByProc10
The gate is locked by processor 10.

enumerator kSEMA42_LockedByProc11
The gate is locked by processor 11.

enumerator kSEMA42_LockedByProc12
The gate is locked by processor 12.

enumerator kSEMA42_LockedByProc13
The gate is locked by processor 13.

enumerator kSEMA42_LockedByProc14
The gate is locked by processor 14.

typedef enum *_sema42_gate_status* sema42_gate_status_t
SEMA42 gate lock status.

void SEMA42_Init(SEMA42_Type *base)
Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42_ResetGate or SEMA42_ResetAllGates function.

Parameters

- base – SEMA42 peripheral base address.

void SEMA42_Deinit(SEMA42_Type *base)
De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

- base – SEMA42 peripheral base address.

status_t SEMA42_TryLock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)
Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_SEMA42_Busy – Sema42 gate has been locked by another processor.

status_t SEMA42_Lock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns

status_t

static inline void SEMA42_Unlock(SEMA42_Type *base, uint8_t gateNum)

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline *sema42_gate_status_t* SEMA42_GetGateStatus(SEMA42_Type *base, uint8_t gateNum)

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

status Current status.

status_t SEMA42_ResetGate(SEMA42_Type *base, uint8_t gateNum)

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – SEMA42 gate is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

static inline *status_t* SEMA42_ResetAllGates(SEMA42_Type *base)

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.

Return values

- kStatus_Success – SEMA42 is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

SEMA42_GATE_NUM_RESET_ALL

The number to reset all SEMA42 gates.

SEMA42_GATEn(base, n)

SEMA42 gate n register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42_GATEn gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U: 0 ^ 3 = 3 1 ^ 3 = 2 2 ^ 3 = 1 3 ^ 3 = 0 4 ^ 3 = 7 5 ^ 3 = 6 6 ^ 3 = 5 7 ^ 3 = 4 ...

SEMA42_BUSY_POLL_COUNT

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.58 SNVS: Secure Non-Volatile Storage

2.59 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.

void SNVS_HP_RTC_TimeSynchronize(SNVS_Type *base)

The function synchronizes RTC counter value with SRTC.

Parameters

- base – SNVS peripheral base address

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.

void SNVS_HP_SetLocks(SNVS_Type *base)

brief Set SNVS HP Set locks.

param base SNVS peripheral base address

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

2.60 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_EnablePassiveTamper(SNVS_Type *base, *snvs_lp_external_tamper_t* pin, *snvs_lp_passive_tamper_t* config)

Enables the specified SNVS external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin
- config – Configuration structure of external passive tamper

status_t SNVS_LP_EnableTxActiveTamper(SNVS_Type *base, *snvs_lp_active_tx_tamper_t* pin, *tamper_active_tx_config_t* config)

Enable active tamper tx external pad.

Parameters

- base – SNVS peripheral base address
- pin – SNVS active tamper pin
- config – Configuration structure of external active tamper

status_t SNVS_LP_EnableRxActiveTamper(SNVS_Type *base, *snvs_lp_external_tamper_t* rx, *tamper_active_rx_config_t* config)

Enable active tamper rx external pad.

Parameters

- base – SNVS peripheral base address
- rx – SNVS external RX tamper pin
- config – SNVS RX tamper config structure

status_t SNVS_LP_SetVoltageTamper(SNVS_Type *base, bool enable)

Sets voltage tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

status_t SNVS_LP_SetTemperatureTamper(SNVS_Type *base, bool enable)

Sets temperature tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

status_t SNVS_LP_SetClockTamper(SNVS_Type *base, bool enable)

Sets clock tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

snvs_lp_external_tamper_status_t SNVS_LP_CheckVoltageTamper(SNVS_Type *base)

brief Check voltage tamper

param base SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_CheckTemperatureTamper(SNVS_Type *base)

Check temperature tamper.

Parameters

- base – SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_CheckClockTamper(SNVS_Type *base)

brief Check clock tamper

param base SNVS peripheral base address

void SNVS_LP_TamperPinTx_GetDefaultConfig(*tamper_active_tx_config_t* *config)

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->clock = kSNVS_ActiveTamper16HZ; config->seed = 0U; config->polynomial = 0U; endcode

Parameters

- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_TamperPinRx_GetDefaultConfig(*tamper_active_rx_config_t* *config)

brief Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->filterenable = 0U; config->filter = 0U; config->tx = kSNVS_ActiveTamper1; endcode param config Pointer to the user's SNVS configuration structure.

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.

void SNVS_LP_DisableExternalTamper(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Disables the specified SNVS external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

void SNVS_LP_DisableAllExternalTamper(SNVS_Type *base)

Disable all external tamper.

Parameters

- base – SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_GetExternalTamperStatus(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Returns status of the specified external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

Returns

The status flag. This is the enumeration :: _snvs_lp_external_tamper_status

void SNVS_LP_ClearExternalTamperStatus(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Clears status of the specified external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

void SNVS_LP_ClearAllExternalTamperStatus(SNVS_Type *base)

Clears status of the all external tamper.

Parameters

- base – SNVS peripheral base address

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.

status_t SNVS_LP_SSM_State_Transition(SNVS_Type *base)

brief Transition SNVS SSM state to Trusted/Non-secure from Check state

param base SNVS peripheral base address

return kStatus_Success: Success in transitioning SSM State kStatus_Fail: SSM State transition failed

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

List of SNVS_LP external tampers.

Values:

enumerator kSNVS_ExternalTamper1

enum _snvs_lp_active_tamper

List of SNVS_LP active tampers.

Values:

enumerator kSNVS_ActiveTamper1

enumerator kSNVS_ActiveTamper2

enumerator kSNVS_ActiveTamper3

enumerator kSNVS_ActiveTamper4

enumerator kSNVS_ActiveTamper5

enum _snvs_lp_active_clock
List of SNVS_LP external tampers.

Values:

enumerator kSNVS_ActiveTamper16HZ

enumerator kSNVS_ActiveTamper8HZ

enumerator kSNVS_ActiveTamper4HZ

enumerator kSNVS_ActiveTamper2HZ

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* snvs_lp_external_tamper_t
List of SNVS_LP external tampers.

typedef enum *_snvs_lp_active_tamper* snvs_lp_active_tx_tamper_t
List of SNVS_LP active tampers.

typedef enum *_snvs_lp_active_clock* snvs_lp_active_clock_t
List of SNVS_LP external tampers.

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 tamper_active_tx_config_t
#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP active TX tamper pins.

struct tamper_active_rx_config_t
#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP active RX tamper pins.

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

2.61 TPM: Timer PWM Module

uint32_t TPM_GetInstance(TPM_Type *base)

Gets the instance from the base address.

Parameters

- base – TPM peripheral base address

Returns

The TPM instance

void TPM_Init(TPM_Type *base, const *tpm_config_t* *config)

Ungates the TPM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the TPM driver.

Parameters

- base – TPM peripheral base address
- config – Pointer to user's TPM config structure.

void TPM_Deinit(TPM_Type *base)

Stops the counter and gates the TPM clock.

Parameters

- base – TPM peripheral base address

void TPM_GetDefaultConfig(*tpm_config_t* *config)

Fill in the TPM config struct with the default settings.

The default values are:

```
    config->prescale = kTPM_Prescale_Divide_1;
    config->useGlobalTimeBase = false;
    config->syncGlobalTimeBase = false;
    config->dozeEnable = false;
    config->dbgMode = false;
    config->enableReloadOnTrigger = false;
    config->enableStopOnOverflow = false;
    config->enableStartOnTrigger = false;
#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
    config->enablePauseOnTrigger = false;
#endif
    config->triggerSelect = kTPM_Trigger_Select_0;
#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
    config->triggerSource = kTPM_TriggerSource_External;
    config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
#endif
#if defined(FSL_FEATURE_TPM_HAS_POL) && FSL_FEATURE_TPM_HAS_POL
    config->chnlPolarity = 0U;
#endif
```

Parameters

- config – Pointer to user's TPM config structure.

tpm_clock_prescale_t TPM_CalculateCounterClkDiv(TPM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

Calculates the counter clock prescaler.

This function calculates the values for SC[PS].

return Calculated clock prescaler value.

Parameters

- base – TPM peripheral base address
- counterPeriod_Hz – The desired frequency in Hz which corresponding to the time when the counter reaches the mod value
- srcClock_Hz – TPM counter clock in Hz

static inline void TPM_Reset(TPM_Type *base)

Performs a software reset on the TPM module.

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

Note: TPM software reset is available on certain SoC's only

Parameters

- base – TPM peripheral base address

status_t TPM_SetupPwm(TPM_Type *base, const *tpm_chnl_pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *tpm_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

- base – TPM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)
- numOfChnls – Number of channels to configure, this should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration tpm_pwm_mode_t
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – TPM counter clock in Hz

Returns

kStatus_Success PWM setup successful kStatus_Error PWM setup failed kStatus_Timeout PWM setup timeout when write register CnV or MOD

status_t TPM_UpdatePwmDutycycle(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_pwm_mode_t* currentPwmMode, uint8_t dutyCyclePercent)

Update the duty cycle of an active PWM signal.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number. In combined mode, this represents the channel pair number
- currentPwmMode – The current PWM mode set during PWM setup

- dutyCyclePercent – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns
kStatus_Success if the PWM setup was successful, kStatus_Error on failure

void TPM_UpdateChnlEdgeLevelSelect(TPM_Type *base, *tpm_chnl_t* chnlNumber, uint8_t level)

Update the edge level selection for a channel.

Note: When the TPM has PWM pause level select feature (FSL_FEATURE_TPM_HAS_PAUSE_LEVEL_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use TPM_DisableChannel API to close the PWM output.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- level – The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

static inline uint8_t TPM_GetChannelContorlBits(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fields).

Deprecated:

Please use TPM_GetChannelControlBits() instead.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns
The contorl bits value. This is the logical OR of members of the enumeration tpm_chnl_control_bit_mask_t.

static inline uint8_t TPM_GetChannelControlBits(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fields).

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns
The control bits value. This is the logical OR of members of the enumeration tpm_chnl_control_bit_mask_t.

static inline *status_t* TPM_DisableChannel(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Dsiable the channel.

This function disable the channel by clear all mode and level control bits.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

static inline *status_t* TPM_EnableChannel(TPM_Type *base, *tpm_chnl_t* chnlNumber, uint8_t control)

Enable the channel according to mode and level configs.

This function enable the channel output according to input mode/level config parameters.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- control – The contorl bits value. This is the logical OR of members of the enumeration tpm_chnl_control_bit_mask_t.

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

void TPM_SetupInputCapture(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_input_capture_edge_t* captureMode)

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- captureMode – Specifies which edge to capture

status_t TPM_SetupOutputCompare(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_output_compare_mode_t* compareMode, uint32_t compareValue)

Configures the TPM to generate timed pulses.

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- compareMode – Action to take on the channel output when the compare condition is met
- compareValue – Value to be programmed in the CnV register.

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnV

void TPM_SetupDualEdgeCapture(TPM_Type *base, *tpm_chnl_t* chnlPairNumber, const *tpm_dual_edge_capture_param_t* *edgeParam, uint32_t filterValue)

Configures the dual edge capture mode of the TPM.

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

- base – TPM peripheral base address
- chnlPairNumber – The TPM channel pair number; options are 0, 1, 2, 3
- edgeParam – Sets up the dual edge capture function
- filterValue – Filter value, specify 0 to disable filter.

void TPM_SetupQuadDecode(TPM_Type *base, const *tpm_phase_params_t* *phaseAParams, const *tpm_phase_params_t* *phaseBParams, *tpm_quad_decode_mode_t* quadMode)

Configures the parameters and activates the quadrature decode mode.

Parameters

- base – TPM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters
- quadMode – Selects encoding mode used in quadrature decoder mode

static inline void TPM_SetChannelPolarity(TPM_Type *base, *tpm_chnl_t* chnlNumber, bool enable)

Set the input and output polarity of each of the channels.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Set the channel polarity to active high; false: Set the channel polarity to active low;

static inline void TPM_EnableChannelExtTrigger(TPM_Type *base, *tpm_chnl_t* chnlNumber, bool enable)

Enable external trigger input to be used by channel.

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

Note: No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

void TPM_EnableInterrupts(TPM_Type *base, uint32_t mask)

Enables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration tpm_interrupt_enable_t

void TPM_DisableInterrupts(TPM_Type *base, uint32_t mask)

Disables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration tpm_interrupt_enable_t

uint32_t TPM_GetEnabledInterrupts(TPM_Type *base)

Gets the enabled TPM interrupts.

Parameters

- base – TPM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration tpm_interrupt_enable_t

void TPM_RegisterCallBack(TPM_Type *base, *tpm_callback_t* callback)

Register callback.

If channel or overflow interrupt is enabled by the user, then a callback can be registered which will be invoked when the interrupt is triggered.

Parameters

- base – TPM peripheral base address
- callback – Callback function

void TPM_DriverIRQHandler(uint32_t instance)

TPM driver IRQ handler common entry.

This function provides the common IRQ request entry for TPM.

Parameters

- instance – TPM instance.

static inline uint32_t TPM_GetChannelValue(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Gets the TPM channel value.

Note: The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns

The channle CnV regisyer value.

static inline uint32_t TPM_GetStatusFlags(TPM_Type *base)

Gets the TPM status flags.

Parameters

- base – TPM peripheral base address

Returns
The status flags. This is the logical OR of members of the enumeration tpm_status_flags_t

static inline void TPM_ClearStatusFlags(TPM_Type *base, uint32_t mask)

Clears the TPM status flags.

Parameters

- base – TPM peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration tpm_status_flags_t

static inline *status_t* TPM_SetTimerPeriod(TPM_Type *base, uint32_t ticks)

Sets the timer period in units of ticks.

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note:

a. This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.

b. Call the utility macros provided in the fsl_common.h to convert usec or msec to ticks.

Parameters

- base – TPM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

static inline uint32_t TPM_GetCurrentTimerCount(TPM_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – TPM peripheral base address

Returns
The current counter value in ticks

static inline void TPM_StartTimer(TPM_Type *base, *tpm_clock_source_t* clockSource)

Starts the TPM counter.

Parameters

- base – TPM peripheral base address
- clockSource – TPM clock source; once clock source is set the counter will start running

static inline *status_t* TPM_StopTimer(TPM_Type *base)

Stops the TPM counter.

Parameters

- base – TPM peripheral base address

Returns

kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

FSL_TPM_DRIVER_VERSION

TPM driver version 2.5.0.

enum _tpm_chnl

List of TPM channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kTPM_Chnl_0

TPM channel number 0

enumerator kTPM_Chnl_1

TPM channel number 1

enumerator kTPM_Chnl_2

TPM channel number 2

enumerator kTPM_Chnl_3

TPM channel number 3

enumerator kTPM_Chnl_4

TPM channel number 4

enumerator kTPM_Chnl_5

TPM channel number 5

enumerator kTPM_Chnl_6

TPM channel number 6

enumerator kTPM_Chnl_7

TPM channel number 7

enum _tpm_pwm_mode

TPM PWM operation modes.

Values:

enumerator kTPM_EdgeAlignedPwm

Edge aligned PWM

enumerator kTPM_CenterAlignedPwm

Center aligned PWM

enumerator kTPM_CombinedPwm

Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

enum _tpm_pwm_level_select

TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

Values:

enumerator kTPM_HighTrue

High true pulses

enumerator kTPM_LowTrue

Low true pulses

enum _tpm_pwm_pause_level_select

TPM PWM output when first enabled or paused: set or clear.

Values:

enumerator kTPM_ClearOnPause

Clear Output when counter first enabled or paused.

enumerator kTPM_SetOnPause

Set Output when counter first enabled or paused.

enum _tpm_chnl_control_bit_mask

List of TPM channel modes and level control bit mask.

Values:

enumerator kTPM_ChnlELSnAMask

Channel ELSA bit mask.

enumerator kTPM_ChnlELSnBMask

Channel ELSB bit mask.

enumerator kTPM_ChnlMSAMask

Channel MSA bit mask.

enumerator kTPM_ChnlMSBMask

Channel MSB bit mask.

enum _tpm_trigger_select

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

Values:

enumerator kTPM_Trigger_Select_0

enumerator kTPM_Trigger_Select_1

enumerator kTPM_Trigger_Select_2

enumerator kTPM_Trigger_Select_3

enumerator kTPM_Trigger_Select_4

enumerator kTPM_Trigger_Select_5

enumerator kTPM_Trigger_Select_6

enumerator kTPM_Trigger_Select_7

enumerator kTPM_Trigger_Select_8

enumerator kTPM_Trigger_Select_9

enumerator kTPM_Trigger_Select_10

enumerator kTPM_Trigger_Select_11

enumerator kTPM_Trigger_Select_12

enumerator kTPM_Trigger_Select_13

enumerator kTPM_Trigger_Select_14

enumerator kTPM_Trigger_Select_15

enum _tpm_trigger_source

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

Values:

enumerator kTPM_TriggerSource_External

Use external trigger input

enumerator kTPM_TriggerSource_Internal

Use internal trigger (channel pin input capture)

enum _tpm_ext_trigger_polarity

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

Values:

enumerator kTPM_ExtTrigger_Active_High

External trigger input is active high

enumerator kTPM_ExtTrigger_Active_Low

External trigger input is active low

enum _tpm_output_compare_mode

TPM output compare modes.

Values:

enumerator kTPM_NoOutputSignal

No channel output when counter reaches CnV

enumerator kTPM_ToggleOnMatch

Toggle output

enumerator kTPM_ClearOnMatch
Clear output

enumerator kTPM_SetOnMatch
Set output

enumerator kTPM_HighPulseOutput
Pulse output high

enumerator kTPM_LowPulseOutput
Pulse output low

enum _tpm_input_capture_edge
TPM input capture edge.

Values:

enumerator kTPM_RisingEdge
Capture on rising edge only

enumerator kTPM_FallingEdge
Capture on falling edge only

enumerator kTPM_RiseAndFallEdge
Capture on rising or falling edge

enum _tpm_quad_decode_mode
TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

Values:

enumerator kTPM_QuadPhaseEncode
Phase A and Phase B encoding mode

enumerator kTPM_QuadCountAndDir
Count and direction encoding mode

enum _tpm_phase_polarity
TPM quadrature phase polarities.

Values:

enumerator kTPM_QuadPhaseNormal
Phase input signal is not inverted

enumerator kTPM_QuadPhaseInvert
Phase input signal is inverted

enum _tpm_clock_source
TPM clock source selection.

Values:

enumerator kTPM_SystemClock
System clock

enumerator kTPM_ExternalClock
External TPM_EXTCLK pin clock

enumerator kTPM_ExternalInputTriggerClock
Selected external input trigger clock

enum _tpm_clock_prescale
TPM prescale value selection for the clock source.

Values:

enumerator kTPM_Prescale_Divide_1
Divide by 1

enumerator kTPM_Prescale_Divide_2
Divide by 2

enumerator kTPM_Prescale_Divide_4
Divide by 4

enumerator kTPM_Prescale_Divide_8
Divide by 8

enumerator kTPM_Prescale_Divide_16
Divide by 16

enumerator kTPM_Prescale_Divide_32
Divide by 32

enumerator kTPM_Prescale_Divide_64
Divide by 64

enumerator kTPM_Prescale_Divide_128
Divide by 128

enum _tpm_interrupt_enable
List of TPM interrupts.

Values:

enumerator kTPM_Chnl0InterruptEnable
Channel 0 interrupt.

enumerator kTPM_Chnl1InterruptEnable
Channel 1 interrupt.

enumerator kTPM_Chnl2InterruptEnable
Channel 2 interrupt.

enumerator kTPM_Chnl3InterruptEnable
Channel 3 interrupt.

enumerator kTPM_Chnl4InterruptEnable
Channel 4 interrupt.

enumerator kTPM_Chnl5InterruptEnable
Channel 5 interrupt.

enumerator kTPM_Chnl6InterruptEnable
Channel 6 interrupt.

enumerator kTPM_Chnl7InterruptEnable
Channel 7 interrupt.

enumerator kTPM_TimeOverflowInterruptEnable
Time overflow interrupt.

enum _tpm_status_flags

List of TPM flags.

Values:

enumerator kTPM_Chnl0Flag

Channel 0 flag

enumerator kTPM_Chnl1Flag

Channel 1 flag

enumerator kTPM_Chnl2Flag

Channel 2 flag

enumerator kTPM_Chnl3Flag

Channel 3 flag

enumerator kTPM_Chnl4Flag

Channel 4 flag

enumerator kTPM_Chnl5Flag

Channel 5 flag

enumerator kTPM_Chnl6Flag

Channel 6 flag

enumerator kTPM_Chnl7Flag

Channel 7 flag

enumerator kTPM_TimeOverflowFlag

Time overflow flag

typedef enum *_tpm_chnl* tpm_chnl_t

List of TPM channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_tpm_pwm_mode* tpm_pwm_mode_t

TPM PWM operation modes.

typedef enum *_tpm_pwm_level_select* tpm_pwm_level_select_t

TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

typedef enum *_tpm_pwm_pause_level_select* tpm_pwm_pause_level_select_t

TPM PWM output when first enabled or paused: set or clear.

typedef enum *_tpm_chnl_control_bit_mask* tpm_chnl_control_bit_mask_t

List of TPM channel modes and level control bit mask.

typedef struct *_tpm_chnl_pwm_signal_param* tpm_chnl_pwm_signal_param_t

Options to configure a TPM channel's PWM signal.

typedef enum *_tpm_trigger_select* tpm_trigger_select_t

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

typedef enum *_tpm_trigger_source* tpm_trigger_source_t

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

typedef enum *_tpm_ext_trigger_polarity* tpm_ext_trigger_polarity_t

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

typedef enum *_tpm_output_compare_mode* tpm_output_compare_mode_t

TPM output compare modes.

typedef enum *_tpm_input_capture_edge* tpm_input_capture_edge_t

TPM input capture edge.

typedef struct *_tpm_dual_edge_capture_param* tpm_dual_edge_capture_param_t

TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_quad_decode_mode* tpm_quad_decode_mode_t

TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_phase_polarity* tpm_phase_polarity_t

TPM quadrature phase polarities.

typedef struct *_tpm_phase_param* tpm_phase_params_t

TPM quadrature decode phase parameters.

typedef enum *_tpm_clock_source* tpm_clock_source_t

TPM clock source selection.

typedef enum *_tpm_clock_prescale* tpm_clock_prescale_t

TPM prescale value selection for the clock source.

typedef struct *_tpm_config* tpm_config_t

TPM config structure.

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

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

typedef enum *_tpm_interrupt_enable* tpm_interrupt_enable_t

List of TPM interrupts.

typedef enum *_tpm_status_flags* tpm_status_flags_t

List of TPM flags.

typedef void (*tpm_callback_t)(TPM_Type *base)

TPM callback function pointer.

Param base
TPM peripheral base address.

TPM_TIMEOUT

Max loops to wait for writing register.

When writing MOD CnV CnSC and SC register, driver will wait until register is updated. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

TPM_MAX_COUNTER_VALUE(x)

Help macro to get the max counter value.

struct _tpm_chnl_pwm_signal_param

#include <fsl_tpm.h> Options to configure a TPM channel's PWM signal.

Public Members

tpm_chnl_t chnlNumber

TPM channel to configure. In combined mode (available in some SoC's), this represents the channel pair number

tpm_pwm_pause_level_select_t pauseLevel

PWM output level when counter first enabled or paused

tpm_pwm_level_select_t level

PWM output active level select

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

uint8_t firstEdgeDelayPercent

Used only in combined PWM mode to generate asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greate than 100.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

tpm_pwm_pause_level_select_t secPauseLevel

Used only in combined PWM mode. Define the second channel output level when counter first enabled or paused

uint8_t deadTimeValue[2]

The dead time value for channel n and n+1 in combined complementary PWM mode. Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue * 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

struct __tpm__dual__edge__capture__param

#include <fsl_tpm.h> TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

Public Members

bool enableSwap

true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored

tpm_input_capture_edge_t currChanEdgeMode

Input capture edge select for channel n

tpm_input_capture_edge_t nextChanEdgeMode

Input capture edge select for channel n+1

struct __tpm__phase__param

#include <fsl_tpm.h> TPM quadrature decode phase parameters.

Public Members

uint32_t phaseFilterVal

Filter value, filter is disabled when the value is zero

tpm_phase_polarity_t phasePolarity

Phase polarity

struct __tpm__config

#include <fsl_tpm.h> TPM config structure.

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

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

Public Members

tpm_clock_prescale_t prescale

Select TPM clock prescale value

bool useGlobalTimeBase

true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase

bool syncGlobalTimeBase

true: The TPM counter is synchronized to the global time base; false: disabled

tpm_trigger_select_t triggerSelect

Input trigger to use for controlling the counter operation

tpm_trigger_source_t triggerSource

Decides if we use external or internal trigger.

tpm_ext_trigger_polarity_t extTriggerPolarity
when using external trigger source, need selects the polarity of it.

bool enableDoze
true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode

bool enableDebugMode
true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode

bool enableReloadOnTrigger
true: TPM counter is reloaded on trigger; false: TPM counter not reloaded

bool enableStopOnOverflow
true: TPM counter stops after overflow; false: TPM counter continues running after overflow

bool enableStartOnTrigger
true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately

bool enablePauseOnTrigger
true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

uint8_t chnlPolarity
Defines the input/output polarity of the channels in POL register

2.62 TRGMUX: Trigger Mux Driver

static inline void TRGMUX_LockRegister(TRGMUX_Type *base, uint32_t index)
Sets the flag of the register which is used to mark writeable.

The function sets the flag of the register which is used to mark writeable. Example:

```
TRGMUX_LockRegister(TRGMUX0,kTRGMUX_Trgmux0Dmamux0);
```

Parameters

- base – TRGMUX peripheral base address.
- index – The index of the TRGMUX register, see the enum trgmux_device_t defined in <SOC>.h.

status_t TRGMUX_SetTriggerSource(TRGMUX_Type *base, uint32_t index,
trgmux_trigger_input_t input, uint32_t trigger_src)
Configures the trigger source of the appointed peripheral.

The function configures the trigger source of the appointed peripheral. Example:

```
TRGMUX_SetTriggerSource(TRGMUX0, kTRGMUX_Trgmux0Dmamux0, kTRGMUX_TriggerInput0,
↪ kTRGMUX_SourcePortPin);
```

Parameters

- base – TRGMUX peripheral base address.
- index – The index of the TRGMUX register, see the enum trgmux_device_t defined in <SOC>.h.
- input – The MUX select for peripheral trigger input

- trigger_src – The trigger inputs for various peripherals. See the enum trgmux_source_t defined in <SOC>.h.

Return values

- kStatus_Success – Configured successfully.
- kStatus_TRGMUX_Locked – Configuration failed because the register is locked.

FSL_TRGMUX_DRIVER_VERSION

TRGMUX driver version.

TRGMUX configure status.

Values:

enumerator kStatus_TRGMUX_Locked

Configure failed for register is locked

enum _trgmux_trigger_input

Defines the MUX select for peripheral trigger input.

Values:

enumerator kTRGMUX_TriggerInput0

The MUX select for peripheral trigger input 0

enumerator kTRGMUX_TriggerInput1

The MUX select for peripheral trigger input 1

enumerator kTRGMUX_TriggerInput2

The MUX select for peripheral trigger input 2

enumerator kTRGMUX_TriggerInput3

The MUX select for peripheral trigger input 3

typedef enum *_trgmux_trigger_input* trgmux_trigger_input_t

Defines the MUX select for peripheral trigger input.

2.63 TRNG: True Random Number Generator

FSL_TRNG_DRIVER_VERSION

TRNG driver version 2.0.22.

Current version: 2.0.22

Change log:

- version 2.0.22
  - Added support for KW43 and MCXW70 devices.
- version 2.0.21
  - Added support for MCXC devices.
- version 2.0.20
  - Added support for MCXA devices.
- version 2.0.19
  - Added support for MCXA and MCXL.

- version 2.0.18
  - TRNG health checks now done in software on RT5xx and RT6xx.
- version 2.0.17
  - Added support for RT700.
- version 2.0.16
  - Added support for Dual oscillator mode.
- version 2.0.15
  - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest reccomended by design team.
- version 2.0.14
  - add support for RW610 and RW612
- version 2.0.13
  - After deepsleep it might return error, added clearing bits in TRNG_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG_GetRandomData(), for some data length it doesn't reloading entropy correctly.
- version 2.0.12
  - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.11
  - Add clearing pending errors in TRNG_Init().
- version 2.0.10
  - Fixed doxygen issues.
- version 2.0.9
  - Fix HIS_CCM metrics issues.
- version 2.0.8
  - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.
- version 2.0.7
  - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
  - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.5
  - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG_Init to restart entropy generation
- Version 2.0.2

- fix MISRA issues

- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

enum _trng_sample_mode

TRNG sample mode. Used by trng_config_t.

Values:

enumerator kTRNG_SampleModeVonNeumann

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeRaw

Use raw data into both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeVonNeumannRaw

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum _trng_clock_mode

TRNG clock mode. Used by trng_config_t.

Values:

enumerator kTRNG_ClockModeRingOscillator

Ring oscillator is used to operate the TRNG (default).

enumerator kTRNG_ClockModeSystem

System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

enum _trng_ring_osc_div

TRNG ring oscillator divide. Used by trng_config_t.

Values:

enumerator kTRNG_RingOscDiv0

Ring oscillator with no divide

enumerator kTRNG_RingOscDiv2

Ring oscillator divided-by-2.

enumerator kTRNG_RingOscDiv4

Ring oscillator divided-by-4.

enumerator kTRNG_RingOscDiv8

Ring oscillator divided-by-8.

typedef enum *_trng_sample_mode* trng_sample_mode_t

TRNG sample mode. Used by trng_config_t.

typedef enum *_trng_clock_mode* trng_clock_mode_t

TRNG clock mode. Used by trng_config_t.

typedef enum *_trng_ring_osc_div* trng_ring_osc_div_t

TRNG ring oscillator divide. Used by trng_config_t.

typedef struct *_trng_statistical_check_limit* trng_statistical_check_limit_t

Data structure for definition of statistical check limits. Used by trng_config_t.

typedef struct *_trng_user_config* trng_config_t

Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the TRNG_Init() function. It contains all TRNG configurations.

status_t TRNG_GetDefaultConfig(*trng_config_t* *userConfig)

Initializes the user configuration structure to default values.

This function initializes the configuration structure to default values. The default values are platform dependent.

Parameters

- userConfig – User configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

status_t TRNG_Init(TRNG_Type *base, const *trng_config_t* *userConfig)

Initializes the TRNG.

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

- base – TRNG base address
- userConfig – Pointer to the initialization configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

void TRNG_Deinit(TRNG_Type *base)

Shuts down the TRNG.

This function shuts down the TRNG.

Parameters

- base – TRNG base address.

status_t TRNG_GetRandomData(TRNG_Type *base, void *data, size_t dataSize)

Gets random data.

This function gets random data from the TRNG.

Parameters

- base – TRNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

Returns

random data

struct __trng_statistical_check_limit

#include <fsl_trng.h> Data structure for definition of statistical check limits. Used by trng_config_t.

Public Members

uint32_t maximum
: Maximum limit.

int32_t minimum
: Minimum limit.

struct __trng_user_config
: *#include <fsl_trng.h>* Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the TRNG_Init() function. It contains all TRNG configurations.

Public Members

bool lock
: Disable programmability of TRNG registers.

trng_clock_mode_t clockMode
: Clock mode used to operate TRNG.

trng_ring_osc_div_t ringOscDiv
: Ring oscillator divide used by TRNG.

trng_sample_mode_t sampleMode
: Sample mode of the TRNG ring oscillator.

uint16_t entropyDelay
: Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken.

uint16_t sampleSize
: Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation.

uint16_t sparseBitLimit
: Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated. This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

uint8_t retryCount
: Retry count. It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

uint8_t longRunMaxLimit
: Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

trng_statistical_check_limit_t monobitLimit
: Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.

trng_statistical_check_limit_t runBit1Limit
: Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.

trng_statistical_check_limit_t runBit2Limit
: Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.

trng_statistical_check_limit_t runBit3Limit

Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.

trng_statistical_check_limit_t runBit4Limit

Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.

trng_statistical_check_limit_t runBit5Limit

Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.

trng_statistical_check_limit_t runBit6PlusLimit

Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.

trng_statistical_check_limit_t pokerLimit

Maximum and minimum limits for statistical check of "Poker Test".

trng_statistical_check_limit_t frequencyCountLimit

Maximum and minimum limits for statistical check of entropy sample frequency count.

2.64 TSTMR: Timestamp Timer Driver

void TSTMR_Init(TSTMR_Type *base)

Init TSTMR.

This function initializes the TSTMR module.

Parameters

- base – TSTMR peripheral base address.

void TSTMR_Deinit(TSTMR_Type *base)

Deinit TSTMR.

This function deinitializes the TSTMR module.

Parameters

- base – TSTMR peripheral base address.

FSL_TSTMR_DRIVER_VERSION

Version 2.1.0

static inline uint64_t TSTMR_ReadTimeStamp(TSTMR_Type *base)

Reads the time stamp.

This function reads the low and high registers and returns the 56-bit free running counter value. This can be read by software at any time to determine the software ticks. TSTMR registers can be read with 32-bit accesses only. The TSTMR LOW read should occur first, followed by the TSTMR HIGH read.

Parameters

- base – TSTMR peripheral base address.

Returns

The 56-bit time stamp value.

void TSTMR_DelayUs(TSTMR_Type *base, uint64_t delayInUs)

Delays for a specified number of microseconds.

This function repeatedly reads the timestamp register and waits for the user-specified delay value.

Parameters

- base – TSTMR peripheral base address.
- delayInUs – Delay value in microseconds.

2.65 WDOG32: 32-bit Watchdog Timer

void WDOG32_GetDefaultConfig(*wdog32_config_t* *config)

Initializes the WDOG32 configuration structure.

This function initializes the WDOG32 configuration structure to default values. The default values are:

```
wdog32Config->enableWdog32 = true;
wdog32Config->clockSource = kWDOG32_ClockSource1;
wdog32Config->prescaler = kWDOG32_ClockPrescalerDivide1;
wdog32Config->workMode.enableWait = true;
wdog32Config->workMode.enableStop = false;
wdog32Config->workMode.enableDebug = false;
wdog32Config->testMode = kWDOG32_TestModeDisabled;
wdog32Config->enableUpdate = true;
wdog32Config->enableInterrupt = false;
wdog32Config->enableWindowMode = false;
wdog32Config->windowValue = 0U;
wdog32Config->timeoutValue = 0xFFFFU;
```

See also:

wdog32_config_t

Parameters

- config – Pointer to the WDOG32 configuration structure.

status_t WDOG32_Init(WDOG_Type *base, const *wdog32_config_t* *config)

Initializes the WDOG32 module.

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
wdog32_config_t config;
WDOG32_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
WDOG32_Init(wdog_base,&config);
```

Note: If there is errata ERR010536 (FSL_FEATURE_WDOG_HAS_ERRATA_010536 defined as 1), then after calling this function, user need delay at least 4 LPO clock cycles before accessing other WDOG32 registers.

Parameters

- base – WDOG32 peripheral base address.
- config – The configuration of the WDOG32.

Return values

- kStatus_Success – The initialization was successful
- kStatus_Timeout – The initialization timed out

status_t WDOG32_Deinit(WDOG_Type *base)

De-initializes the WDOG32 module.

This function shuts down the WDOG32. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

- base – WDOG32 peripheral base address.

Return values

- kStatus_Success – The de-initialization was successful
- kStatus_Timeout – The de-initialization timed out

status_t WDOG32_Unlock(WDOG_Type *base)

Unlocks the WDOG32 register written.

This function unlocks the WDOG32 register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

- base – WDOG32 peripheral base address

Return values

- kStatus_Success – The unlock sequence was successful
- kStatus_Timeout – The unlock sequence timed out

void WDOG32_Enable(WDOG_Type *base)

Enables the WDOG32 module.

This function writes a value into the WDOG_CS register to enable the WDOG32. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.

void WDOG32_Disable(WDOG_Type *base)

Disables the WDOG32 module.

This function writes a value into the WDOG_CS register to disable the WDOG32. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address

void WDOG32_EnableInterrupts(WDOG_Type *base, uint32_t mask)

Enables the WDOG32 interrupt.

This function writes a value into the WDOG_CS register to enable the WDOG32 interrupt. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.
- mask – The interrupts to enable. The parameter can be a combination of the following source if defined:
  - kWDOG32_InterruptEnable

void WDOG32_DisableInterrupts(WDOG_Type *base, uint32_t mask)

Disables the WDOG32 interrupt.

This function writes a value into the WDOG_CS register to disable the WDOG32 interrupt. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.
- mask – The interrupts to disabled. The parameter can be a combination of the following source if defined:
  - kWDOG32_InterruptEnable

static inline uint32_t WDOG32_GetStatusFlags(WDOG_Type *base)

Gets the WDOG32 all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;
status = WDOG32_GetStatusFlags(wdog_base) & kWDOG32_RunningFlag;
```

See also:

_wdog32_status_flags_t

- true: related status flag has been set.
- false: related status flag is not set.

Parameters

- base – WDOG32 peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void WDOG32_ClearStatusFlags(WDOG_Type *base, uint32_t mask)

Clears the WDOG32 flag.

This function clears the WDOG32 status flag.

Example to clear an interrupt flag:

```
WDOG32_ClearStatusFlags(wdog_base,kWDOG32_InterruptFlag);
```

Parameters

- base – WDOG32 peripheral base address.
- mask – The status flags to clear. The parameter can be any combination of the following values:
  - kWDOG32_InterruptFlag

void WDOG32_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG32 timeout value.

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. To ensure the reconfiguration fits the timing of WCT, unlock function will be called inline.

Parameters

- base – WDOG32 peripheral base address
- timeoutCount – WDOG32 timeout value, count of WDOG32 clock ticks.

void WDOG32_SetWindowValue(WDOG_Type *base, uint16_t windowValue)

Sets the WDOG32 window value.

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.
- windowValue – WDOG32 window value.

static inline void WDOG32_Refresh(WDOG_Type *base)

Refreshes the WDOG32 timer.

This function feeds the WDOG32. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG32 peripheral base address

static inline uint16_t WDOG32_GetCounterValue(WDOG_Type *base)

Gets the WDOG32 counter value.

This function gets the WDOG32 counter value.

Parameters

- base – WDOG32 peripheral base address.

Returns

Current WDOG32 counter value.

WDOG_FIRST_WORD_OF_UNLOCK

First word of unlock sequence

WDOG_SECOND_WORD_OF_UNLOCK

Second word of unlock sequence

WDOG_FIRST_WORD_OF_REFRESH
First word of refresh sequence

WDOG_SECOND_WORD_OF_REFRESH
Second word of refresh sequence

FSL_WDOG32_DRIVER_VERSION
WDOG32 driver version.

enum _wdog32_clock_source
Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

Values:

enumerator kWDOG32_ClockSource0
Clock source 0

enumerator kWDOG32_ClockSource1
Clock source 1

enumerator kWDOG32_ClockSource2
Clock source 2

enumerator kWDOG32_ClockSource3
Clock source 3

enum _wdog32_clock_prescaler
Describes the selection of the clock prescaler.

Values:

enumerator kWDOG32_ClockPrescalerDivide1
Divided by 1

enumerator kWDOG32_ClockPrescalerDivide256
Divided by 256

enum _wdog32_test_mode
Describes WDOG32 test mode.

Values:

enumerator kWDOG32_TestModeDisabled
Test Mode disabled

enumerator kWDOG32_UserModeEnabled
User Mode enabled

enumerator kWDOG32_LowByteTest
Test Mode enabled, only low byte is used

enumerator kWDOG32_HighByteTest
Test Mode enabled, only high byte is used

enum _wdog32_interrupt_enable_t

WDOG32 interrupt configuration structure.

This structure contains the settings for all of the WDOG32 interrupt configurations.

Values:

enumerator kWDOG32_InterruptEnable

Interrupt is generated before forcing a reset

enum _wdog32_status_flags_t

WDOG32 status flags.

This structure contains the WDOG32 status flags for use in the WDOG32 functions.

Values:

enumerator kWDOG32_RunningFlag

Running flag, set when WDOG32 is enabled

enumerator kWDOG32_InterruptFlag

Interrupt flag, set when interrupt occurs

typedef enum *_wdog32_clock_source* wdog32_clock_source_t

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

typedef enum *_wdog32_clock_prescaler* wdog32_clock_prescaler_t

Describes the selection of the clock prescaler.

typedef struct *_wdog32_work_mode* wdog32_work_mode_t

Defines WDOG32 work mode.

typedef enum *_wdog32_test_mode* wdog32_test_mode_t

Describes WDOG32 test mode.

typedef struct *_wdog32_config* wdog32_config_t

Describes WDOG32 configuration structure.

struct _wdog32_work_mode

#include <fsl_wdog32.h> Defines WDOG32 work mode.

Public Members

bool enableWait

Enables or disables WDOG32 in wait mode

bool enableStop

Enables or disables WDOG32 in stop mode

bool enableDebug

Enables or disables WDOG32 in debug mode

struct _wdog32_config

#include <fsl_wdog32.h> Describes WDOG32 configuration structure.

Public Members

bool enableWdog32
: Enables or disables WDOG32

wdog32_clock_source_t clockSource
: Clock source select

wdog32_clock_prescaler_t prescaler
: Clock prescaler value

wdog32_work_mode_t workMode
: Configures WDOG32 work mode in debug stop and wait mode

wdog32_test_mode_t testMode
: Configures WDOG32 test mode

bool enableUpdate
: Update write-once register enable

bool enableInterrupt
: Enables or disables WDOG32 interrupt

bool enableWindowMode
: Enables or disables WDOG32 window mode

uint16_t windowValue
: Window value

uint16_t timeoutValue
: Timeout value

2.66 XRDC: Extended Resource Domain Controller

void XRDC_GetHardwareConfig(XRDC_Type *base, *xrdc_hardware_config_t* *config)
: Gets the XRDC hardware configuration.

  This function gets the XRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

  Parameters

  - base – XRDC peripheral base address.
  - config – Pointer to the structure to get the configuration.

static inline void XRDC_LockGlobalControl(XRDC_Type *base)
: Locks the XRDC global control register XRDC_CR.

  This function locks the XRDC_CR register. After it is locked, the register is read-only until the next reset.

  Parameters

  - base – XRDC peripheral base address.

static inline void XRDC_SetGlobalValid(XRDC_Type *base, bool valid)
: Sets the XRDC global valid.

  This function sets the XRDC global valid or invalid. When the XRDC is global invalid, all accesses from all bus masters to all slaves are allowed.

  Parameters

- base – XRDC peripheral base address.
- valid – True to valid XRDC.

static inline uint8_t XRDC_GetCurrentMasterDomainId(XRDC_Type *base)

Gets the domain ID of the current bus master.

This function returns the domain ID of the current bus master.

Parameters

- base – XRDC peripheral base address.

Returns

Domain ID of current bus master.

status_t XRDC_GetAndClearFirstDomainError(XRDC_Type *base, *xrdc_error_t* *error)

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – XRDC peripheral base address.
- error – Pointer to the error information.

Returns

If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_XRDC_NoError.

status_t XRDC_GetAndClearFirstSpecificDomainError(XRDC_Type *base, *xrdc_error_t* *error, uint8_t domainId)

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – XRDC peripheral base address.
- error – Pointer to the error information.
- domainId – The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_XRDC_NoError.

void XRDC_GetPidDefaultConfig(*xrdc_pid_config_t* *config)

Gets the default PID configuration structure.

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

```
config->pid       = 0U;
config->tsmEnable = 0U;
config->sp4smEnable = 0U;
config->lockMode  = kXRDC_PidLockSecurePrivilegeWritable;
```

Parameters

- config – Pointer to the configuration structure.

void XRDC_SetPidConfig(XRDC_Type *base, xrdc_master_t master, const *xrdc_pid_config_t* *config)

Configures the PID for a specific bus master.

This function configures the PID for a specific bus master. Do not use this function for non-processor bus masters.

Parameters

- base – XRDC peripheral base address.
- master – Which bus master to configure.
- config – Pointer to the configuration structure.

static inline void XRDC_SetPidLockMode(XRDC_Type *base, xrdc_master_t master, *xrdc_pid_lock_t* lockMode)

Sets the PID configuration register lock mode.

This function sets the PID configuration register lock XRDC_PIDn[LK2].

Parameters

- base – XRDC peripheral base address.
- master – Which master's PID to lock.
- lockMode – Lock mode to set.

void XRDC_GetDefaultNonProcessorDomainAssignment(*xrdc_non_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId            = 0U;
assignment->privilegeAttr       = kXRDC_ForceUser;
assignment->privilegeAttr       = kXRDC_ForceSecure;
assignment->bypassDomainId      = 0U;
assignment->blogicPartId        = 0U;
assignment->benableLogicPartId  = 0U;
assignment->lock                = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void XRDC_GetDefaultProcessorDomainAssignment(*xrdc_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId           = 0U;
assignment->domainIdSelect     = kXRDC_DidMda;
assignment->dpidEnable         = kXRDC_PidDisable;
assignment->pidMask            = 0U;
assignment->pid                = 0U;
assignment->logicPartId        = 0U;
```

(continues on next page)

(continued from previous page)

```
assignment->enableLogicPartId  = 0U;
assignment->lock               = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void XRDC_SetNonProcessorDomainAssignment(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex, const *xrdc_non_processor_domain_assignment_t* *domainAssignment)

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
xrdc_non_processor_domain_assignment_t nonProcessorAssignment;

XRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Which assignment register to set.
- domainAssignment – Pointer to the assignment structure.

void XRDC_SetProcessorDomainAssignment(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex, const *xrdc_processor_domain_assignment_t* *domainAssignment)

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for core 0. In this example, there are 3 assignment registers for core 0.

```
xrdc_processor_domain_assignment_t processorAssignment;

XRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 1;
processorAssignment.xxx      = xxx;
XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterCpu0, 0U, &processorAssignment);

processorAssignment.domainId = 2;
processorAssignment.xxx      = xxx;
XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterCpu0, 1U, &processorAssignment);
```

(continues on next page)

(continued from previous page)

```
processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterCpu0, 2U, &processorAssignment);
```

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Which assignment register to set.
- domainAssignment – Pointer to the assignment structure.

static inline void XRDC_LockMasterDomainAssignment(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex)

Locks the bus master domain assignment register.

This function locks the master domain assignment. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to lock. After it is locked, the register can't be changed until next reset.

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Which assignment register to lock.

static inline void XRDC_SetMasterDomainAssignmentValid(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to configure.

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Index for the domain assignment register.
- valid – True to set valid, false to set invalid.

void XRDC_GetMemAccessDefaultConfig(*xrdc_mem_access_config_t* *config)

Gets the default memory region access policy.

This function gets the default memory region access policy. It sets the policy as follows:

```
config->enableSema          = false;
config->semaNum             = 0U;
config->subRegionDisableMask = 0U;
config->size                = kXrdcMemSizeNone;
config->lockMode            = kXRDC_AccessConfigLockWritable;
config->baseAddress         = 0U;
config->policy[0]           = kXRDC_AccessPolicyNone;
config->policy[1]           = kXRDC_AccessPolicyNone;
...
config->policy[15]          = kXRDC_AccessPolicyNone;
```

Parameters

- config – Pointer to the configuration structure.

void XRDC_SetMemAccessConfig(XRDC_Type *base, const *xrdc_mem_access_config_t* *config)

Sets the memory region access policy.

This function sets the memory region access configuration as valid. There are two methods to use it:

Example 1: Set one configuration run time. Set memory region 0x20000000 ~ 0x20000400 accessible by domain 0, use MRC0_1.

```
xrdc_mem_access_config_t config =
{
    .mem         = kXRDC_MemMrc0_1,
    .baseAddress = 0x20000000U,
    .size        = kXRDC_MemSize1K,
    .policy[0]   = kXRDC_AccessPolicyAll
};
XRDC_SetMemAccessConfig(XRDC, &config);
```

Example 2: Set multiple configurations during startup. Set memory region 0x20000000 ~ 0x20000400 accessible by domain 0, use MRC0_1. Set memory region 0x1FFF0000 ~ 0x1FFF0800 accessible by domain 0, use MRC0_2.

```
xrdc_mem_access_config_t configs[] =
{
    {
        .mem         = kXRDC_MemMrc0_1,
        .baseAddress = 0x20000000U,
        .size        = kXRDC_MemSize1K,
        .policy[0]   = kXRDC_AccessPolicyAll
    },
    {
        .mem         = kXRDC_MemMrc0_2,
        .baseAddress = 0x1FFF0000U,
        .size        = kXRDC_MemSize2K,
        .policy[0]   = kXRDC_AccessPolicyAll
    }
};

for (i=0U; i<((sizeof(configs)/sizeof(configs[0]))); i++)
{
    XRDC_SetMemAccessConfig(XRDC, &configs[i]);
}
```

Parameters

- base – XRDC peripheral base address.
- config – Pointer to the access policy configuration structure.

static inline void XRDC_SetMemAccessLockMode(XRDC_Type *base, xrdc_mem_t mem, *xrdc_access_config_lock_t* lockMode)

Sets the memory region descriptor register lock mode.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region descriptor to lock.
- lockMode – The lock mode to set.

static inline void XRDC_SetMemAccessValid(XRDC_Type *base, xrdc_mem_t mem, bool valid)

Sets the memory region descriptor as valid or invalid.

This function sets the memory region access configuration dynamically. For example:

```
xrdc_mem_access_config_t config =
{
    .mem          = kXRDC_MemMrc0_1,
    .baseAddress = 0x20000000U,
    .size         = kXRDC_MemSize1K,
    .policy[0]    = kXRDC_AccessPolicyAll
};
XRDC_SetMemAccessConfig(XRDC, &config);

XRDC_SetMemAccessValid(kXRDC_MemMrc0_1, false);

XRDC_SetMemAccessValid(kXRDC_MemMrc0_1, true);
```

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region descriptor to set.
- valid – True to set valid, false to set invalid.

void XRDC_SetMemExclAccessLockMode(XRDC_Type *base, xrdc_mem_t mem, *xrdc_excl_access_lock_config_t* lockMode)

Sets the memory region exclusive access lock mode configuration.

Note: Any write to MRGD_W[0-3]_n clears the MRGD_W4_n[VLD] indicator so a coherent register state can be supported. It is indispensable to re-assert the valid bit when dynamically changing the EAL in the MRGD, which is done in this API.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region's exclusive access lock mode to configure.
- lockMode – The exclusive access lock mode to set.

void XRDC_ForceMemExclAccessLockRelease(XRDC_Type *base, xrdc_mem_t mem)

Forces the release of the memory region exclusive access lock.

A lock can be forced to the available state (EAL=10) by a domain that does not own the lock through the forced lock release procedure: The procedure to force a exclusive access lock release is as follows:

a. Write 0x02000046 to W1 register (PAC/MSC) or W3 register (MRC)
b. Write 0x02000052 to W1 register (PAC/MSC) or W3 register (MRC)

Note: The two writes must be consecutive, any intervening write to the register resets the sequence.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region's exclusive access lock to force release.

static inline uint8_t XRDC_GetMemExclAccessLockDomainOwner(XRDC_Type *base, xrdc_mem_t mem)

Gets the exclusive access lock domain owner of the memory region.

This function returns the domain ID of the exclusive access lock owner of the memory region.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region's exclusive access lock domain owner to get.

Returns

Domain ID of the memory region exclusive access lock owner.

void XRDC_GetPeriphAccessDefaultConfig(*xrdc_periph_access_config_t* *config)

Gets the default peripheral access configuration.

The default configuration is set as follows:

```
config->enableSema      = false;
config->semaNum         = 0U;
config->lockMode        = kXRDC_AccessConfigLockWritable;
config->policy[0]       = kXRDC_AccessPolicyNone;
config->policy[1]       = kXRDC_AccessPolicyNone;
...
config->policy[15]      = kXRDC_AccessPolicyNone;
```

Parameters

- config – Pointer to the configuration structure.

void XRDC_SetPeriphAccessConfig(XRDC_Type *base, const *xrdc_periph_access_config_t* *config)

Sets the peripheral access configuration.

This function sets the peripheral access configuration as valid. Two methods to use it: Method 1: Set for one peripheral, which is used for runtime settings. Example: set LPTMR0 accessible by domain 0

```
xrdc_periph_access_config_t config;

config.periph    = kXRDC_PeriphLptmr0;
config.policy[0] = kXRDC_AccessPolicyAll;
XRDC_SetPeriphAccessConfig(XRDC, &config);
```

Method 2: Set for multiple peripherals, which is used for initialization settings.

```
xrdc_periph_access_config_t configs[] =
{
    {
        .periph    = kXRDC_PeriphLptmr0,
        .policy[0] = kXRDC_AccessPolicyAll,
        .policy[1] = kXRDC_AccessPolicyAll
    },
    {
        .periph    = kXRDC_PeriphLpuart0,
        .policy[0] = kXRDC_AccessPolicyAll,
        .policy[1] = kXRDC_AccessPolicyAll
    }
};

for (i=0U; i<(sizeof(configs)/sizeof(configs[0])), i++)
{
    XRDC_SetPeriphAccessConfig(XRDC, &config[i]);
}
```

Parameters

- base – XRDC peripheral base address.
- config – Pointer to the configuration structure.

static inline void XRDC_SetPeriphAccessLockMode(XRDC_Type *base, xrdc_periph_t periph, *xrdc_access_config_lock_t* lockMode)

Sets the peripheral access configuration register lock mode.

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral access configuration register to lock.
- lockMode – The lock mode to set.

static inline void XRDC_SetPeriphAccessValid(XRDC_Type *base, xrdc_periph_t periph, bool valid)

Sets the peripheral access as valid or invalid.

This function sets the peripheral access configuration dynamically. For example:

```
xrdc_periph_access_config_t config =
{
    .periph    = kXRDC_PeriphLptmr0;
    .policy[0] = kXRDC_AccessPolicyAll;
};
XRDC_SetPeriphAccessConfig(XRDC, &config);

XRDC_SetPeriphAccessValid(kXrdcPeriLptmr0, false);

XRDC_SetPeriphAccessValid(kXrdcPeriLptmr0, true);
```

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral access configuration to set.
- valid – True to set valid, false to set invalid.

static inline void XRDC_SetPeriphExclAccessLockMode(XRDC_Type *base, xrdc_periph_t periph, *xrdc_excl_access_lock_config_t* lockMode)

Sets the peripheral exclusive access lock mode configuration.

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral's exclusive access lock mode to configure.
- lockMode – The exclusive access lock mode to set.

void XRDC_ForcePeriphExclAccessLockRelease(XRDC_Type *base, xrdc_periph_t periph)

Forces the release of the peripheral exclusive access lock.

A lock can be forced to the available state (EAL=10) by a domain that does not own the lock through the forced lock release procedure: The procedure to force a exclusive access lock release is as follows:

a. Write 0x02000046 to W1 register (PAC/MSC) or W3 register (MRC)
b. Write 0x02000052 to W1 register (PAC/MSC) or W3 register (MRC)

Note: The two writes must be consecutive, any intervening write to the register resets the sequence.

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral's exclusive access lock to force release.

static inline uint8_t XRDC_GetPeriphExclAccessLockDomainOwner(XRDC_Type *base, xrdc_periph_t periph)

Gets the exclusive access lock domain owner of the peripheral.

This function returns the domain ID of the exclusive access lock owner of the peripheral.

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral's exclusive access lock domain owner to get.

Returns

Domain ID of the peripheral exclusive access lock owner.

XRDC status _xrdc_status.

Values:

enumerator kStatus_XRDC_NoError

No error captured.

enum _xrdc_pid_enable

XRDC PID enable mode, the register bit XRDC_MDA_Wx[PE], used for domain hit evaluation.

Values:

enumerator kXRDC_PidDisable

PID is not used in domain hit evalution.

enumerator kXRDC_PidDisable1

PID is not used in domain hit evalution.

enumerator kXRDC_PidExp0

((XRDC_MDA_W[PID] & ~XRDC_MDA_W[PIDM]) == (XRDC_PID[PID] & ~XRDC_MDA_W[PIDM])).

enumerator kXRDC_PidExp1

~((XRDC_MDA_W[PID] & ~XRDC_MDA_W[PIDM]) == (XRDC_PID[PID] & ~XRDC_MDA_W[PIDM])).

enum _xrdc_did_sel

XRDC domain ID select method, the register bit XRDC_MDA_Wx[DIDS], used for domain hit evaluation.

Values:

enumerator kXRDC_DidMda

Use MDAn[3:0] as DID.

enumerator kXRDC_DidInput

Use the input DID (DID_in) as DID.

enumerator kXRDC_DidMdaAndInput

Use MDAn[3:2] concatenated with DID_in[1:0] as DID.

enumerator kXRDC_DidReserved

Reserved.

enum _xrdc_secure_attr

XRDC secure attribute, the register bit XRDC_MDA_Wx[SA], used for non-processor bus master domain assignment.

Values:

enumerator kXRDC_ForceSecure
Force the bus attribute for this master to secure.

enumerator kXRDC_ForceNonSecure
Force the bus attribute for this master to non-secure.

enumerator kXRDC_MasterSecure
Use the bus master's secure/nonsecure attribute directly.

enumerator kXRDC_MasterSecure1
Use the bus master's secure/nonsecure attribute directly.

enum _xrdc_privilege_attr
XRDC privileged attribute, the register bit XRDC_MDA_Wx[PA], used for non-processor bus master domain assignment.

Values:

enumerator kXRDC_ForceUser
Force the bus attribute for this master to user.

enumerator kXRDC_ForcePrivilege
Force the bus attribute for this master to privileged.

enumerator kXRDC_MasterPrivilege
Use the bus master's attribute directly.

enumerator kXRDC_MasterPrivilege1
Use the bus master's attribute directly.

enum _xrdc_pid_lock
XRDC PID LK2 definition XRDC_PIDn[LK2].

Values:

enumerator kXRDC_PidLockSecurePrivilegeWritable
Writable by any secure privileged write.

enumerator kXRDC_PidLockSecurePrivilegeWritable1
Writable by any secure privileged write.

enumerator kXRDC_PidLockMasterXOnly
PIDx is only writable by master x.

enumerator kXRDC_PidLockLocked
Read-only until the next reset.

enum _xrdc_access_policy
XRDC domain access control policy.

Values:

enumerator kXRDC_AccessPolicyNone

enumerator kXRDC_AccessPolicySpuR

enumerator kXRDC_AccessPolicySpRw

enumerator kXRDC_AccessPolicySpuRw

enumerator kXRDC_AccessPolicySpuRwNpR

enumerator kXRDC_AccessPolicySpuRwNpuR

enumerator kXRDC_AccessPolicySpuRwNpRw

enumerator kXRDC_AccessPolicyAll

enum _xrdc_access_config_lock

Access configuration lock mode, the register field PDAC and MRGD LK2.

Values:

enumerator kXRDC_AccessConfigLockWritable

Entire PDACn/MRGDn can be written.

enumerator kXRDC_AccessConfigLockWritable1

Entire PDACn/MRGDn can be written.

enumerator kXRDC_AccessConfigLockDomainXOnly

Domain x only write the DxACP field.

enumerator kXRDC_AccessConfigLockLocked

PDACn is read-only until the next reset.

enum _xrdc_excl_access_lock_config

Exclusive access lock mode configuration, the register field PDAC and MRGD EAL.

Values:

enumerator kXRDC_ExclAccessLockDisabled

Lock disabled.

enumerator kXRDC_ExclAccessLockDisabledUntilNextRst

Lock disabled until next reset.

enumerator kXRDC_ExclAccessLockEnabledStateAvail

Lock enabled, lock state = available.

enumerator kXRDC_ExclAccessLockEnabledStateNotAvail

Lock enabled, lock state = not available.

enum _xrdc_mem_code_region

XRDC memory code region indicator.

Values:

enumerator kXRDC_MemCodeRegion0

Code region indicator 0=data.

enumerator kXRDC_MemCodeRegion1

Code region indicator 1=code.

enum _xrdc_access_flags_select

XRDC domain access flags/policy select.

Policy: {R,W,X} Read, write, execute flags. flag = 0 : inhibits access, flag = 1 : allows access. policy => SecurePriv_NonSecurePriv_SecureUser_NonSecureUsr xxx_xxx_xxx_xxx => PS{R,W,X}_PN{R,W,X}_US{R,W,X}_UN{R,W,X}

```
PS > PN > US > UN        PS > PN > US > UN
```

DxSEL CodeRegion = 0 CodeRegion = 1 000 000_000_000_000 = 0x000 000_000_000_000 = 0x000 001 ACCSET1 010 ACCSET2 011 110_000_000_000 = 0xC00 001_001_001_001 = 0x249 100 110_110_000_000 = 0xD80 111_000_000_000 = 0xE00 101 110_110_100_100 = 0xDA4 110_111_000_000 = 0xDC0 110 110_110_110_000 = 0xDB0 110_110_111_000 = 0xDB8 111 110_110_110_110 = 0xDB6 110_110_111_111 = 0xDBF

Values:

enumerator kXRDC_AccessFlagsNone

enumerator kXRDC_AccessFlagsAlt1

enumerator kXRDC_AccessFlagsAlt2

enumerator kXRDC_AccessFlagsAlt3

enumerator kXRDC_AccessFlagsAlt4

enumerator kXRDC_AccessFlagsAlt5

enumerator kXRDC_AccessFlagsAlt6

enumerator kXRDC_AccessFlagsAlt7

enum _xrdc_controller

XRDC controller definition for domain error check.

Values:

enumerator kXRDC_MemController0
Memory region controller 0.

enumerator kXRDC_MemController1
Memory region controller 1.

enumerator kXRDC_MemController2
Memory region controller 2.

enumerator kXRDC_MemController3
Memory region controller 3.

enumerator kXRDC_MemController4
Memory region controller 4.

enumerator kXRDC_MemController5
Memory region controller 5.

enumerator kXRDC_MemController6
Memory region controller 6.

enumerator kXRDC_MemController7
Memory region controller 7.

enumerator kXRDC_MemController8
Memory region controller 8.

enumerator kXRDC_MemController9
Memory region controller 9.

enumerator kXRDC_MemController10
Memory region controller 10.

enumerator kXRDC_MemController11
Memory region controller 11.

enumerator kXRDC_MemController12
Memory region controller 12.

enumerator kXRDC_MemController13
Memory region controller 13.

enumerator kXRDC_MemController14
Memory region controller 14.

enumerator kXRDC_MemController15
Memory region controller 15.

enumerator kXRDC_PeriphController0
Peripheral access controller 0.

enumerator kXRDC_PeriphController1
Peripheral access controller 1.

enumerator kXRDC_PeriphController2
Peripheral access controller 2.

enumerator kXRDC_PeriphController3
Peripheral access controller 3.

enum _xrdc_error_state
XRDC domain error state definition XRDC_DERR_W1_n[EST].

Values:

enumerator kXRDC_ErrorStateNone
No access violation detected.

enumerator kXRDC_ErrorStateNone1
No access violation detected.

enumerator kXRDC_ErrorStateSingle
Single access violation detected.

enumerator kXRDC_ErrorStateMulti
Multiple access violation detected.

enum _xrdc_error_attr
XRDC domain error attribute definition XRDC_DERR_W1_n[EATR].

Values:

enumerator kXRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kXRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kXRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kXRDC_ErrorSecurePrivilegeData
Secure privileged mode, data access.

enumerator kXRDC_ErrorNonSecureUserInst
NonSecure user mode, instruction fetch access.

enumerator kXRDC_ErrorNonSecureUserData
NonSecure user mode, data access.

enumerator kXRDC_ErrorNonSecurePrivilegeInst
NonSecure privileged mode, instruction fetch access.

enumerator kXRDC_ErrorNonSecurePrivilegeData
NonSecure privileged mode, data access.

enum _xrdc_error_type

XRDC domain error access type definition XRDC_DERR_W1_n[ERW].

Values:

enumerator kXRDC_ErrorTypeRead

Error occurs on read reference.

enumerator kXRDC_ErrorTypeWrite

Error occurs on write reference.

typedef struct *_xrdc_hardware_config* xrdc_hardware_config_t

XRDC hardware configuration.

typedef enum *_xrdc_pid_enable* xrdc_pid_enable_t

XRDC PID enable mode, the register bit XRDC_MDA_Wx[PE], used for domain hit evaluation.

typedef enum *_xrdc_did_sel* xrdc_did_sel_t

XRDC domain ID select method, the register bit XRDC_MDA_Wx[DIDS], used for domain hit evaluation.

typedef enum *_xrdc_secure_attr* xrdc_secure_attr_t

XRDC secure attribute, the register bit XRDC_MDA_Wx[SA], used for non-processor bus master domain assignment.

typedef enum *_xrdc_privilege_attr* xrdc_privilege_attr_t

XRDC privileged attribute, the register bit XRDC_MDA_Wx[PA], used for non-processor bus master domain assignment.

typedef struct *_xrdc_processor_domain_assignment* xrdc_processor_domain_assignment_t

Domain assignment for the processor bus master.

typedef struct *_xrdc_non_processor_domain_assignment*
xrdc_non_processor_domain_assignment_t

Domain assignment for the non-processor bus master.

typedef enum *_xrdc_pid_lock* xrdc_pid_lock_t

XRDC PID LK2 definition XRDC_PIDn[LK2].

typedef struct *_xrdc_pid_config* xrdc_pid_config_t

XRDC process identifier (PID) configuration.

typedef enum *_xrdc_access_policy* xrdc_access_policy_t

XRDC domain access control policy.

typedef enum *_xrdc_access_config_lock* xrdc_access_config_lock_t

Access configuration lock mode, the register field PDAC and MRGD LK2.

typedef enum *_xrdc_excl_access_lock_config* xrdc_excl_access_lock_config_t

Exclusive access lock mode configuration, the register field PDAC and MRGD EAL.

typedef struct *_xrdc_periph_access_config* xrdc_periph_access_config_t

XRDC peripheral domain access control configuration.

typedef enum *_xrdc_mem_code_region* xrdc_mem_code_region_t

XRDC memory code region indicator.

typedef enum *_xrdc_access_flags_select* xrdc_access_flags_select_t

XRDC domain access flags/policy select.

Policy: {R,W,X} Read, write, execute flags. flag = 0 : inhibits access, flag = 1 : allows access. policy => SecurePriv_NonSecurePriv_SecureUser_NonSecureUsr xxx_xxx_xxx_xxx => PS{R,W,X}_PN{R,W,X}_US{R,W,X}_UN{R,W,X}

```
PS > PN > US > UN        PS > PN > US > UN
```

DxSEL CodeRegion = 0 CodeRegion = 1 000 000_000_000_000 = 0x000 000_000_000_000 = 0x000 001 ACCSET1 010 ACCSET2 011 110_000_000_000 = 0xC00 001_001_001_001 = 0x249 100 110_110_000_000 = 0xD80 111_000_000_000 = 0xE00 101 110_110_100_100 = 0xDA4 110_111_000_000 = 0xDC0 110 110_110_110_000 = 0xDB0 110_110_111_000 = 0xDB8 111 110_110_110_110 = 0xDB6 110_110_111_111 = 0xDBF

typedef struct *_xrdc_mem_access_config* xrdc_mem_access_config_t
XRDC memory region domain access control configuration.

typedef enum *_xrdc_controller* xrdc_controller_t
XRDC controller definition for domain error check.

typedef enum *_xrdc_error_state* xrdc_error_state_t
XRDC domain error state definition XRDC_DERR_W1_n[EST].

typedef enum *_xrdc_error_attr* xrdc_error_attr_t
XRDC domain error attribute definition XRDC_DERR_W1_n[EATR].

typedef enum *_xrdc_error_type* xrdc_error_type_t
XRDC domain error access type definition XRDC_DERR_W1_n[ERW].

typedef struct *_xrdc_error* xrdc_error_t
XRDC domain error definition.

void XRDC_Init(XRDC_Type *base)
Initializes the XRDC module.

This function enables the XRDC clock.

Parameters

- base – XRDC peripheral base address.

void XRDC_Deinit(XRDC_Type *base)
De-initializes the XRDC module.

This function disables the XRDC clock.

Parameters

- base – XRDC peripheral base address.

FSL_XRDC_DRIVER_VERSION

struct _xrdc_hardware_config
#include <fsl_xrdc.h> XRDC hardware configuration.

Public Members

uint8_t masterNumber
Number of bus masters.

uint8_t domainNumber
Number of domains.

uint8_t pacNumber
Number of PACs.

uint8_t mrcNumber
Number of MRCs.

struct _xrdc_processor_domain_assignment
#include <fsl_xrdc.h> Domain assignment for the processor bus master.

Public Members

uint32_t domainId
: Domain ID.

uint32_t domainIdSelect
: Domain ID select method, see xrdc_did_sel_t.

uint32_t pidEnable
: PId enable method, see xrdc_pid_enable_t.

uint32_t pidMask
: PId mask.

uint32_t ___pad0___
: Reserved.

uint32_t pid
: PId value.

uint32_t ___pad1___
: Reserved.

uint32_t ___pad2___
: Reserved.

uint32_t ___pad3___
: Reserved.

uint32_t ___pad4___
: Reserved.

uint32_t lock
: Lock the register.

uint32_t ___pad5___
: Reserved.

struct __xrdc__non__processor__domain__assignment
: *#include <fsl_xrdc.h>* Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId
: Domain ID.

uint32_t privilegeAttr
: Privileged attribute, see xrdc_privilege_attr_t.

uint32_t secureAttr
: Secure attribute, see xrdc_secure_attr_t.

uint32_t bypassDomainId
: Bypass domain ID.

uint32_t ___pad0___
: Reserved.

uint32_t ___pad1___
: Reserved.

uint32_t ___pad2___
Reserved.

uint32_t ___pad3___
Reserved.

uint32_t lock
Lock the register.

uint32_t ___pad4___
Reserved.

struct _xrdc_pid_config
#include <fsl_xrdc.h> XRDC process identifier (PID) configuration.

Public Members

uint32_t pid
PID value, PIDn[PID].

uint32_t ___pad0___
Reserved.

uint32_t sp4smEnable
Enable special 4-state model.

uint32_t tsmEnable
Enable three-state model.

uint32_t lockMode
PIDn configuration lock mode, see xrdc_pid_lock_t.

uint32_t ___pad1___
Reserved.

struct _xrdc_periph_access_config
#include <fsl_xrdc.h> XRDC peripheral domain access control configuration.

Public Members

xrdc_periph_t periph
Peripheral name.

xrdc_access_config_lock_t lockMode
PDACn lock configuration.

xrdc_excl_access_lock_config_t exclAccessLockMode
Exclusive access lock configuration.

xrdc_access_policy_t policy[1]
Access policy for each domain.

struct _xrdc_mem_access_config
#include <fsl_xrdc.h> XRDC memory region domain access control configuration.

Public Members

xrdc_mem_t mem
: Memory region descriptor name.

xrdc_access_config_lock_t lockMode
: MRGDn lock configuration.

xrdc_access_flags_select_t policy[1]
: Access policy/flags select for each domain.

xrdc_mem_code_region_t codeRegion
: Code region select. xrdc_mem_code_region_t.

uint32_t baseAddress
: Memory region base/start address.

uint32_t endAddress
: Memory region end address. The 5 LSB of end address is ignored and forced to 0x1F by hardware.

xrdc_excl_access_lock_config_t exclAccessLockMode
: Exclusive access lock configuration.

struct _xrdc_error
: *#include <fsl_xrdc.h>* XRDC domain error definition.

Public Members

xrdc_controller_t controller
: Which controller captured access violation.

uint32_t address
: Access address that generated access violation.

xrdc_error_state_t errorState
: Error state.

xrdc_error_attr_t errorAttr
: Error attribute.

xrdc_error_type_t errorType
: Error type.

uint8_t errorPort
: Error port.

uint8_t domainId
: Domain ID.

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 */

/* 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_TII	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types

The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P(	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types

The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	The smallest memory entity. On the vast majority of platforms, this is an unsigned 8-bit integer. On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	A single character in the communication buffer. Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

Document references

Links

- This document online: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

3.2 Multicore SDK

Multicore Software Development Kit (MCSDK) provides comprehensive software support for NXP multicore devices. The MCSDK is combined with the MCUXpresso SDK to form a framework for easy development of multicore applications.

3.2.1 Quick links

- *Release Notes*

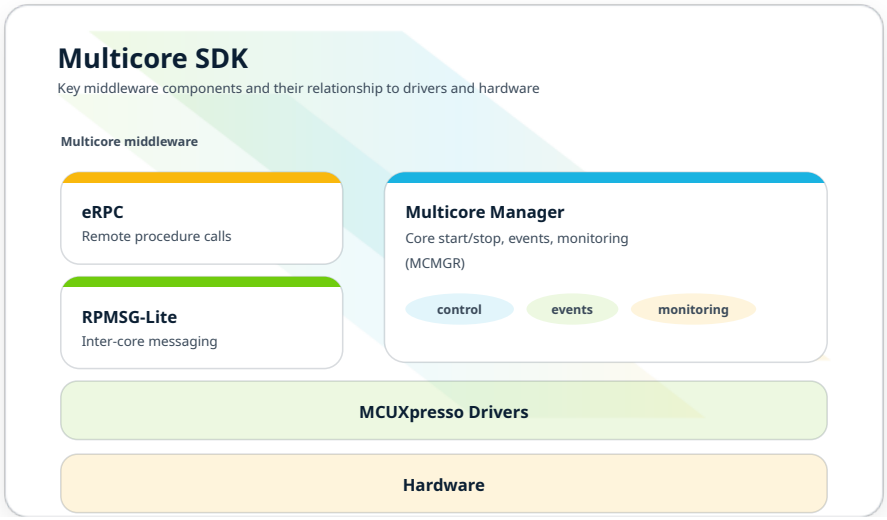


Fig. 1: Multicore SDK software stack (simplified).

- *Getting Started*
- *Multicore SDK Changelog*

3.2.2 Components

Component	Description
RPMSG-Lite	Lightweight implementation of the Remote Processor Messaging (RPMsg) protocol for inter-core messaging.
Multicore Manager	Services for multicore systems (core start/stop, events, and monitoring).
eRPC	Embedded RPC system suitable for multicore and multiprocessor designs.

MCSDK Release Notes

Multicore SDK (MCSDK) Release Notes Overview

These are the release notes for the NXP **Multicore Software Development Kit (MCSDK)**.

Item	Value
MCSDK version	**26.06.00**
Scope	Multicore middleware components and examples
Location in MCUXpresso SDK	<MCUXpressoSDK_install_dir>/middleware/multicore/

The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications on NXP devices.

What's new

Component updates

Review detailed changes per component:

- eRPC: *CHANGELOG*
- RPMSG-Lite: *CHANGELOG*
- Multicore Manager (MCMGR): *CHANGELOG*

Supported evaluation boards

Boards with enabled multicore example coverage in this release:

Multicore examples
LPCXpresso55S69FRDM-K32L3A6MIMXRT1170-EVKBMIMXRT1160-EVKMIMXRT1180-EVKMCX-N5XX-EVKMCX-N9XX-EVKFRDM-MCXN947MIMXRT700-EVKKW47-EVKKW47-LOCFRDM-MCXW72MCX-W72-EVKFRDM-IMXRT1186

Development tools

The MCSDK is compiled and tested with the toolchains described in the MCUXpresso SDK documentation:

- https://mcuxpresso.nxp.com/mcuxsdk/latest/html/gsd/installation.html#toolchain

Release contents

This table describes the release contents. Not all MCUXpresso SDK packages contain the full 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 (eRPC)	<MCSDK_dir>/erpc/
Multicore Manager (MCMGR)	<MCSDK_dir>/mcmgr/
RPMSG-Lite	<MCSDK_dir>/rpmsg-lite/
Multicore demo applications	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK component overview

Together, the MCSDK and the MCUXpresso SDK form a framework for developing software for NXP multicore devices. The MCSDK release includes:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR)
- Remote Processor Messaging – Lite (RPMSG-Lite)

The MCSDK also includes documentation and demo applications for multicore communication.

Demo applications

The demo applications demonstrate MCSDK component usage on supported development boards.

Examples are located under:

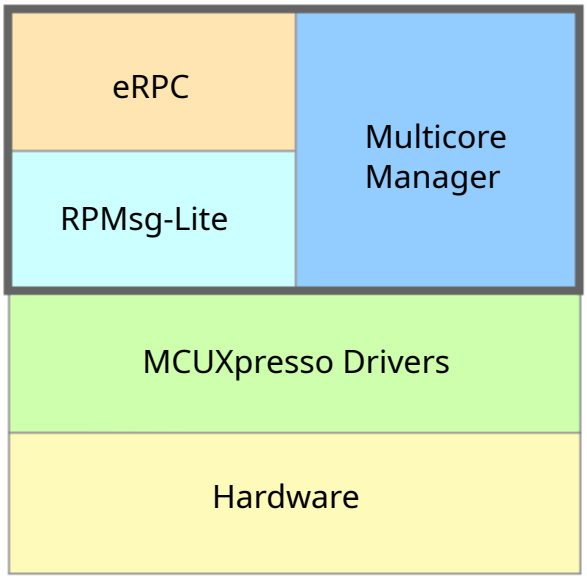
- Multicore: <MCUXpressoSDK_install_dir>/examples/multicore_examples/

Multicore examples
freertos_message_buffershello_worldmulticore_managerrpmsg_lite_pingpongrpmsg_lite_pingpong_rtosrpmsg_lite_

MCSDK Getting Started

Getting Started with Multicore SDK (MCSDK) Overview

The **Multicore Software Development Kit (MCSDK)** provides software components for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications.



All MCSDK-related files are located in:

- <MCUXpressoSDK_install_dir>/middleware/multicore/

For supported toolchain versions, see the release notes.

Quick links

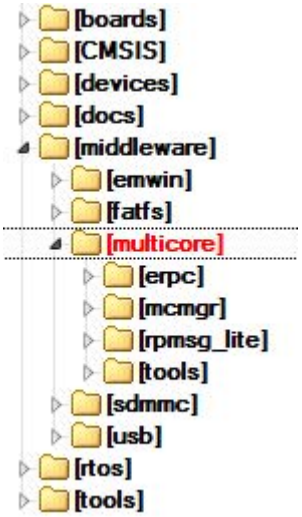
- MCSDK Release Notes
- *MCSDK Changelog*
- Component repositories (within MCSDK):
  - *RPMSG-Lite*
  - *Multicore Manager (MCMGR)*
  - eRPC
- eRPC detailed Getting Started guide:

– eRPC Getting Started

MCSDK components

The MCSDK consists of the following components:

- **Embedded Remote Procedure Call (eRPC):** Library + code generator implementing transparent function calls to services running on another core.
- **Multicore Manager (MCMGR):** Core management services (start/stop secondary cores, events, monitoring).
- **Remote Processor Messaging – Lite (RPMSG-Lite):** Lightweight inter-processor messaging library based on RPMsg.



Where things live

Item	Path
MCSDK root	<MCUXpressoSDK_install_dir>/middleware/multicore/
eRPC	<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/
MCMGR	<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/
RPMSG-Lite	<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite/
MCSDK documentation	<MCUXpressoSDK_install_dir>/middleware/multicore/mcuxsdk-doc/
Multicore examples	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK demo applications

Multicore example applications are stored together with other MCUXpresso SDK examples.

Location	Folder
Multicore example projects	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/

Each example application contains a README describing setup, required images, and runtime behavior.

Inter-Processor Communication (IPC) levels

The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Pick the lightest one that matches your requirements.

IPC using low-level drivers

NXP multicore SoCs commonly provide dedicated IPC peripherals (Mailbox for some LPC parts, Messaging Unit (MU) for some Kinetis and i.MX parts).

This is the most lightweight approach: using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via registers (scalar or pointer to shared memory) and trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific device.

Messaging mechanism (RPMSG-Lite)

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to be sent between endpoints created on each CPU.

RPMSG-Lite provides this capability and is the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for message exchange.

This costs additional code and data memory, but provides a unified messaging API and helps portability across devices/RTOSes.

Remote procedure calls (eRPC)

To facilitate IPC further, eRPC allows remote function invocation via a simple local function call.

Using different transport layers, it can communicate:

- between individual cores of multicore SoCs (commonly via RPMSG-Lite)
- between separate processors (SPI, UART, TCP/IP)

If the communicating peer is a Linux user-space application, the generated code can be in C/C++ or Python.

The IPC levels demonstrate the scalability of the MCSDK. Based on application needs, different IPC techniques can be used depending on complexity, speed, memory, and system design.

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.

[26.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.2.0
  - RPMsg-Lite v5.4.1

[26.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0

  - Multicore Manager (MCMgr) v5.1.0
  - RPMsg-Lite v5.4.0

[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

RPMSG-Lite

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/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:

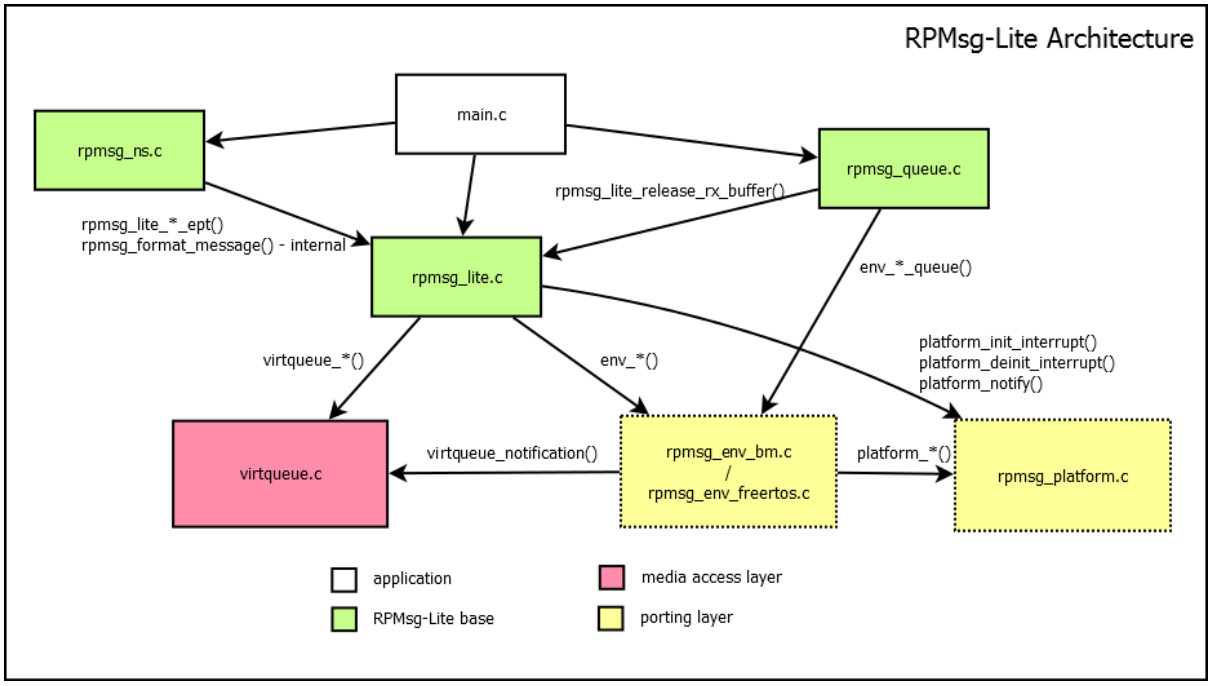
Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

Implementation

The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in rpmsg_lite.c. Two optional components are used to implement a blocking receive API (in rpmsg_queue.c) and dynamic “named” endpoint creation and deletion announcement service (in rpmsg_ns.c).

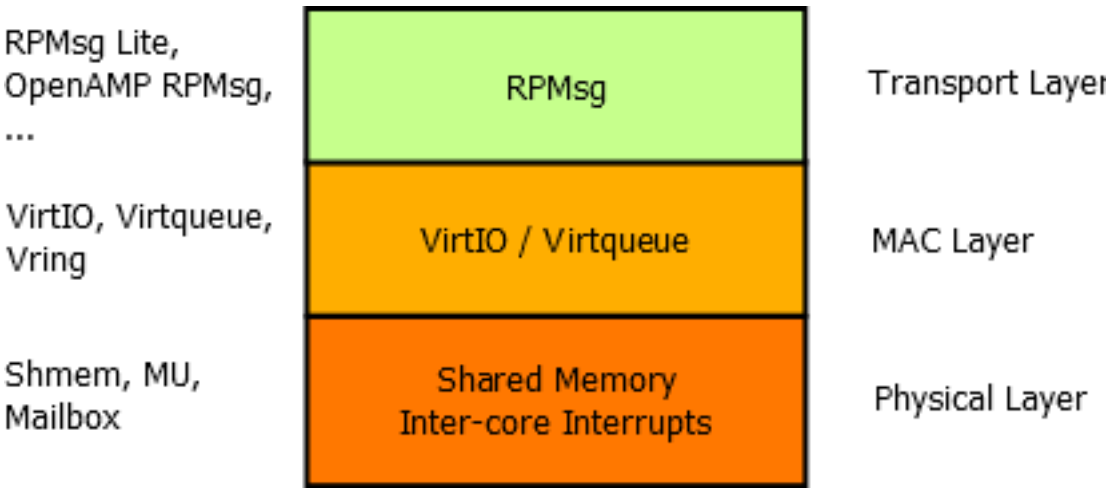
The actual “media access” layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in rpmsg_env_bm.c, and the FreeRTOS environment is implemented in rpmsg_env_freertos.c etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in rpmsg_platform.c and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component

This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional)

This subcomponent is optional and requires implementation of the env_*_queue() functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional)

This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to

receive these announcement taking any user-defined action in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage

The application should put the /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.

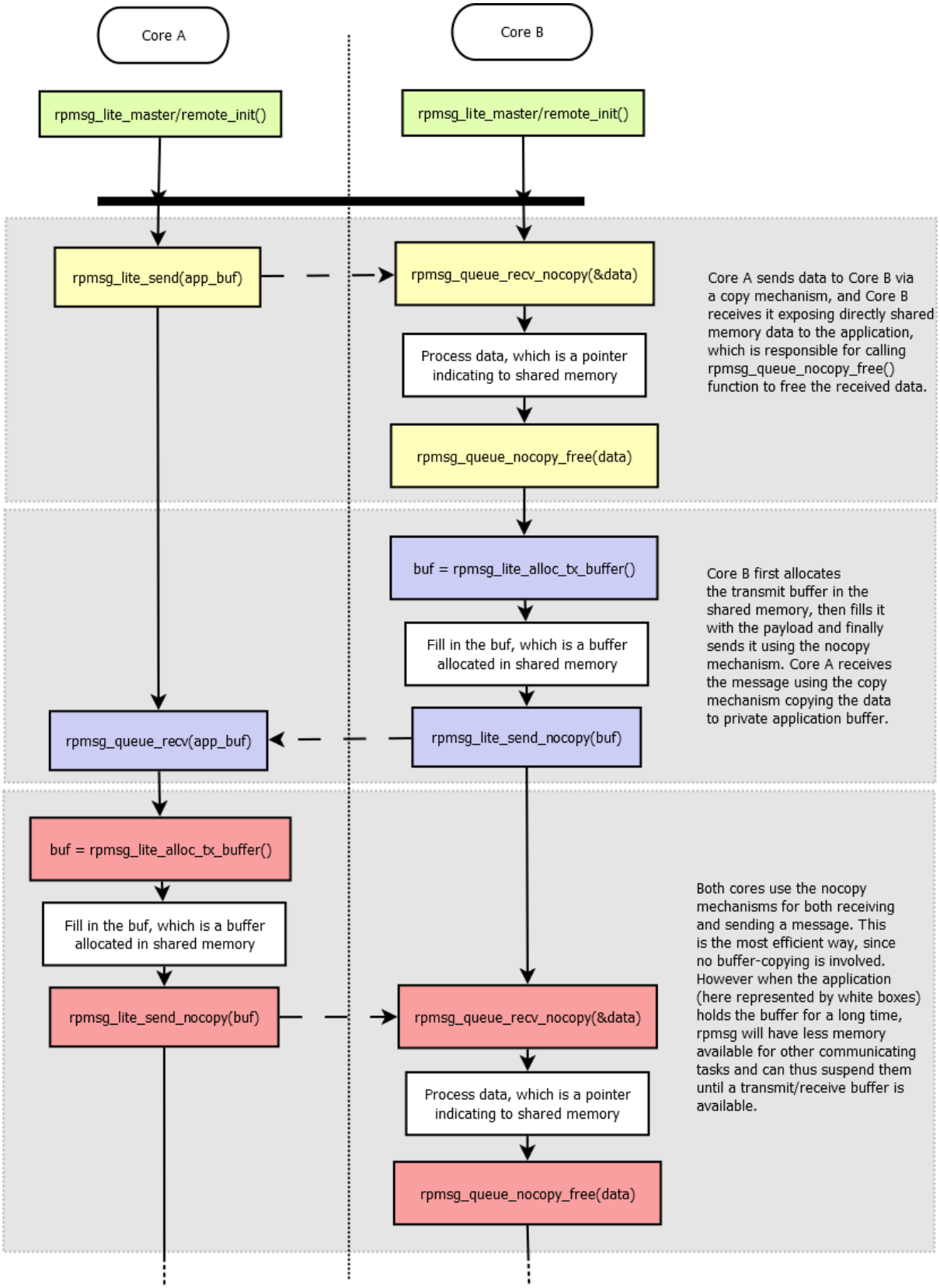
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- 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.

v5.4.1

Fixed

- Corrected isr_counter3 handling in RT700 porting layer platform_deinit_interrupt implementation.

v5.4.0

Added

- Added i.MX93W parts support
- Added i.MX952 parts support
- Added mcxl14x parts support
- Added mcxw70 parts support
- Zephyr rpmsg-lite samples add support for RT600 and RT700 DSP cores

Changed

- Port platform_get_custom_shmem_config change return value from uint32_t to int32_t for error handling consistency.

Fixed

- Updated unit tests code to allow correct GCOV code coverage measurement.
- Added missing dsb instructions into all ISRs for cm4/cm7 based platform layers.
- Fixed CERT EXP34-C by adding __attribute__((noreturn)) to RL_HANG(void) function.
- **MISRA C-2012 Rule 14.3 compliance**: Simplified RL_WORD_ALIGN_UP and RL_WORD_ALIGN_DOWN macros to eliminate invariant controlling expression violations (CID 5011148)
  - Replaced ternary operator implementations with standard bitwise alignment idioms
  - Fixed hardcoded 4U constant in RL_WORD_ALIGN_UP to use RL_WORD_SIZE for better portability
- Fixed several MISRA compliance violations

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 (MCMGR)

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/mcux-mcmgr/

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

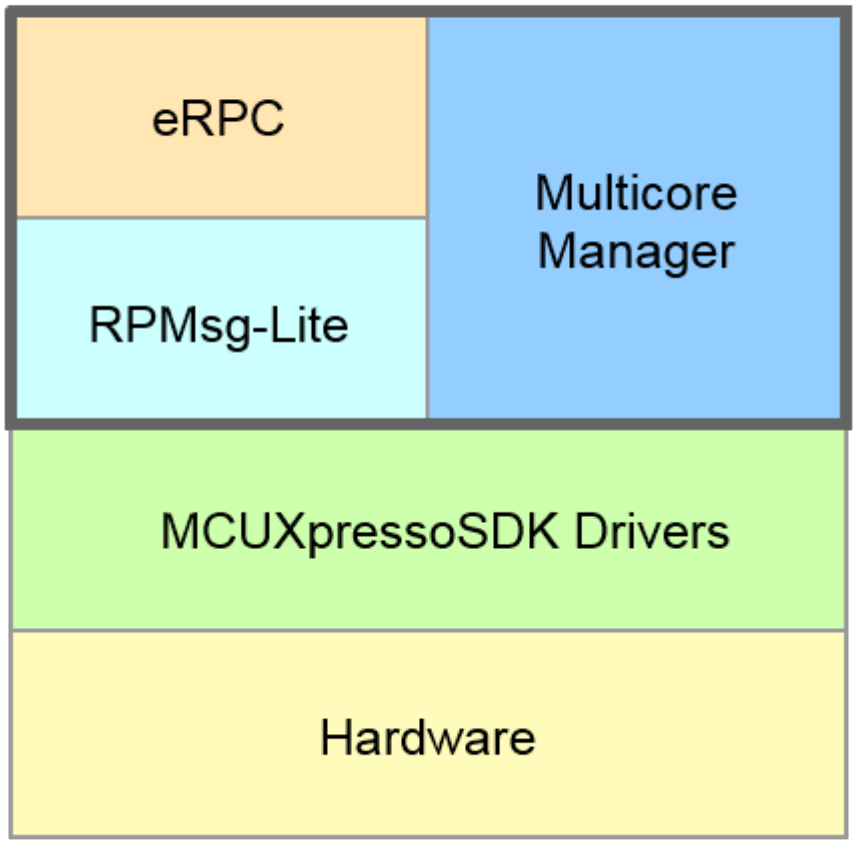
Contribution

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 */
    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 */
    MCMGR_Init();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPMsgRemoteReadyEventHandler, (void␣
↪*)RPMsgRemoteReadyEventData);

    /* 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␣
```

(continues on next page)

(continued from previous page)

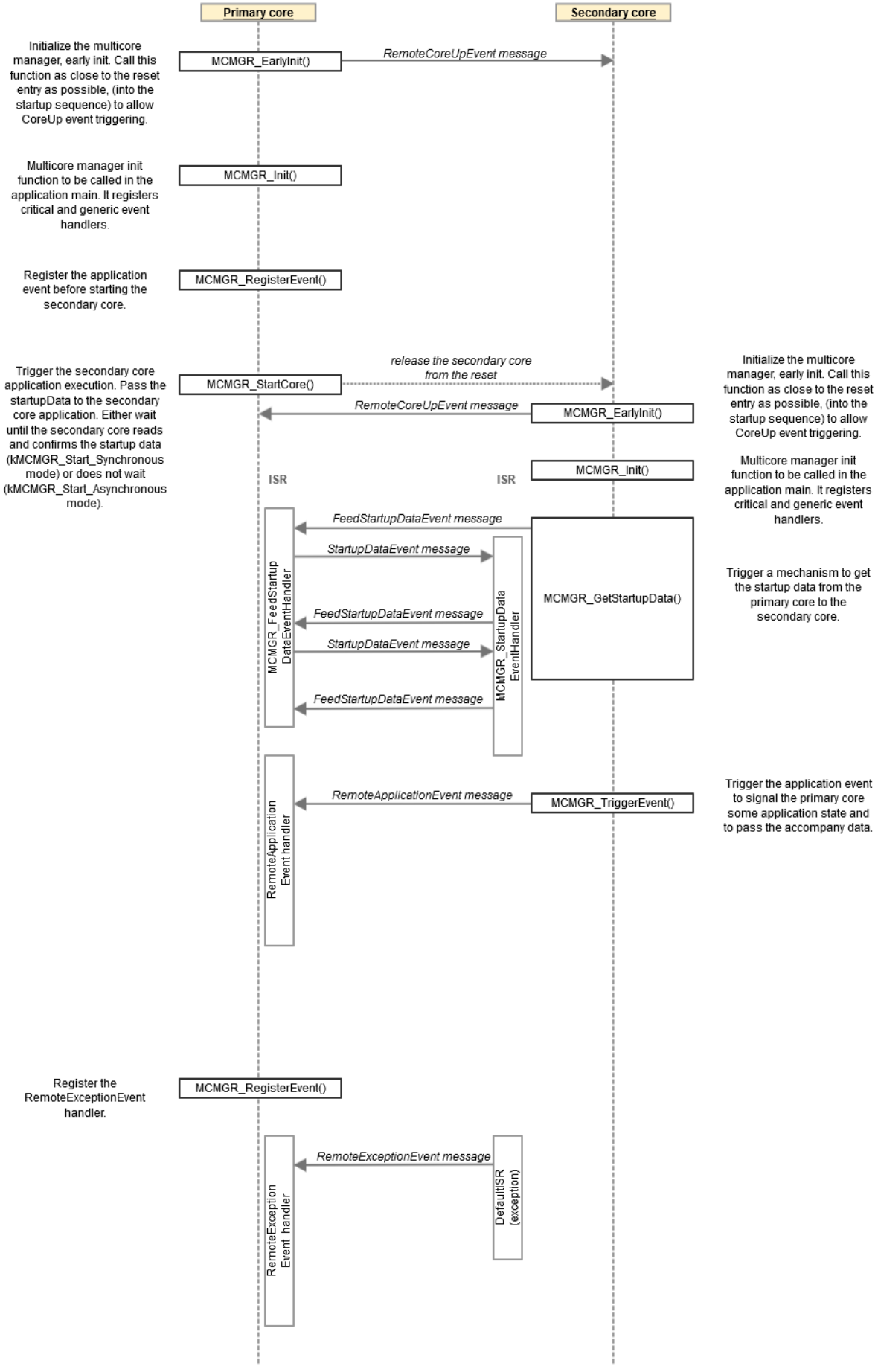
```
↪communicate. */
   while(APP_RPMSG_READY_EVENT_DATA != RPMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)

void main()
{
   /* Initialize MCMGR, low level multicore management library */
   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.

v5.2.0

Changed

- Merged internal platform initialization into a single mcmgr_platform_init_internal() function.
- Simplified initialization API: MCMGR_Init() now handles both early and standard initialization internally.

Deprecated

- MCMGR_EarlyInit() is now deprecated and a no-op. It remains for backward compatibility and may be removed in a future major version. Replace calls to MCMGR_EarlyInit() + MCMGR_Init() with a single MCMGR_Init() call.

Migration Guide

Before:

```
MCMGR_EarlyInit();
// ...
MCMGR_Init();
```

After:

```
MCMGR_Init();
```

v5.1.0

Added

- Added mcxl14x parts support
- Added mcxw70 parts support
- Add new config MCMGR_REMOTE_APP_EVENT_COUNT for configurable remote application event count. This will expand number of application events available for MCMGR users. kMCMGR_RemoteApplicationEvent, kMCMGR_RemoteApplicationEvent1 .. kMCMGR_RemoteApplicationEvent4.
- Align Kinetis parts porting to changed register map

Fixed

- Fix MISRA C-2012 Rule 20.8 violations.

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

eRPC (Embedded RPC) is an open-source remote procedure call (RPC) system suitable for multi-core and multiprocessor embedded systems.

Quick links

- *Getting Started*
- *Changelog*
- Upstream project: https://github.com/EmbeddedRPC/erpc

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:

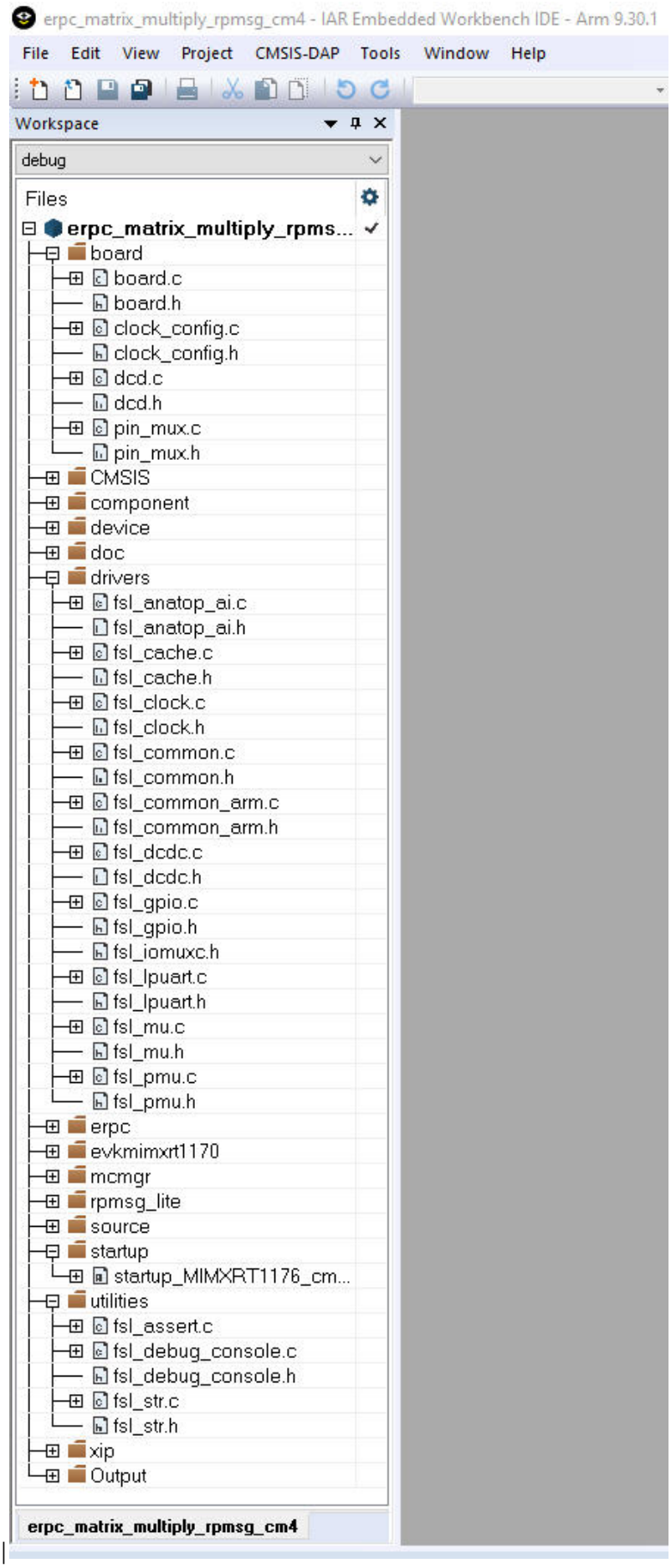
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

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

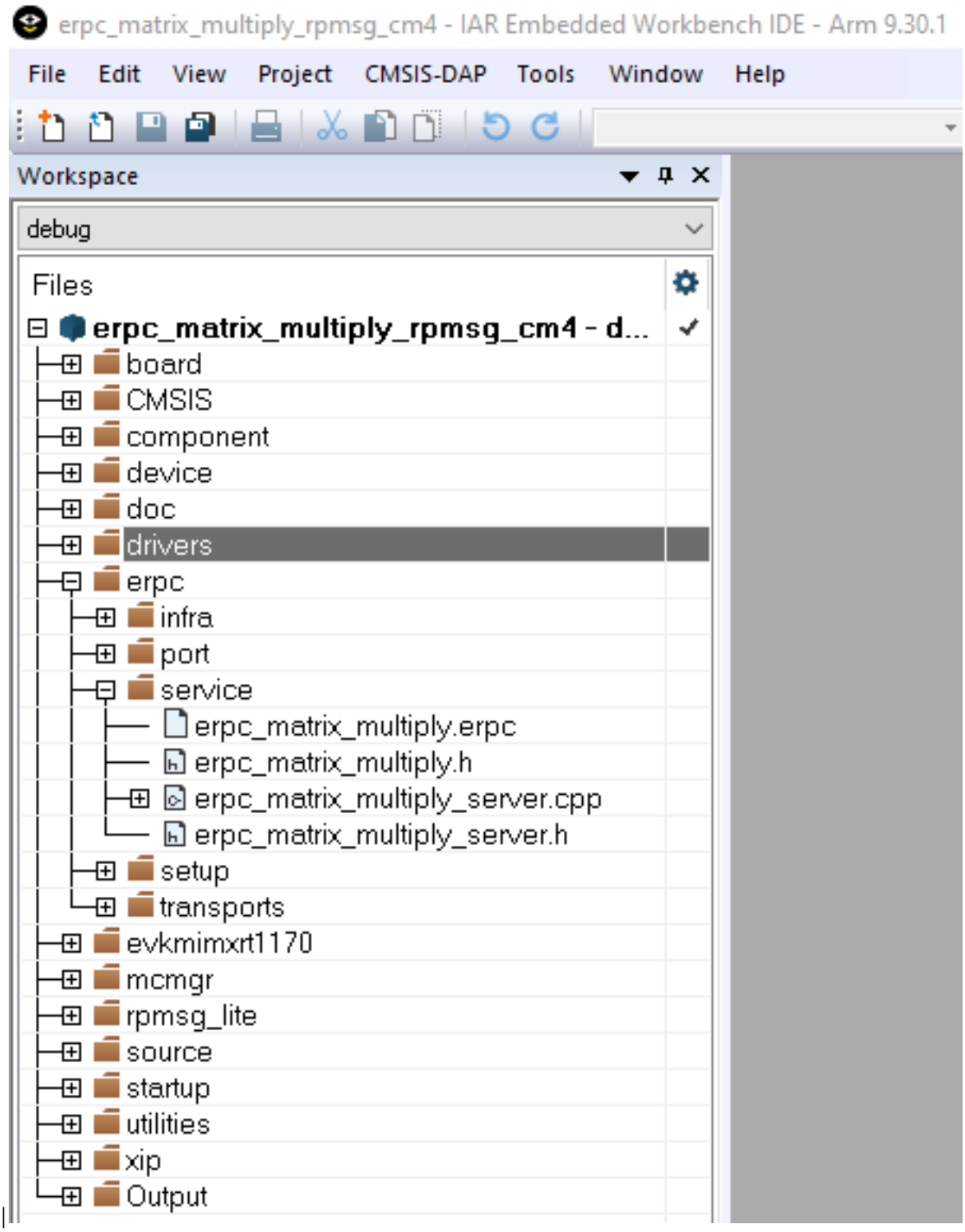
Server related generated files

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

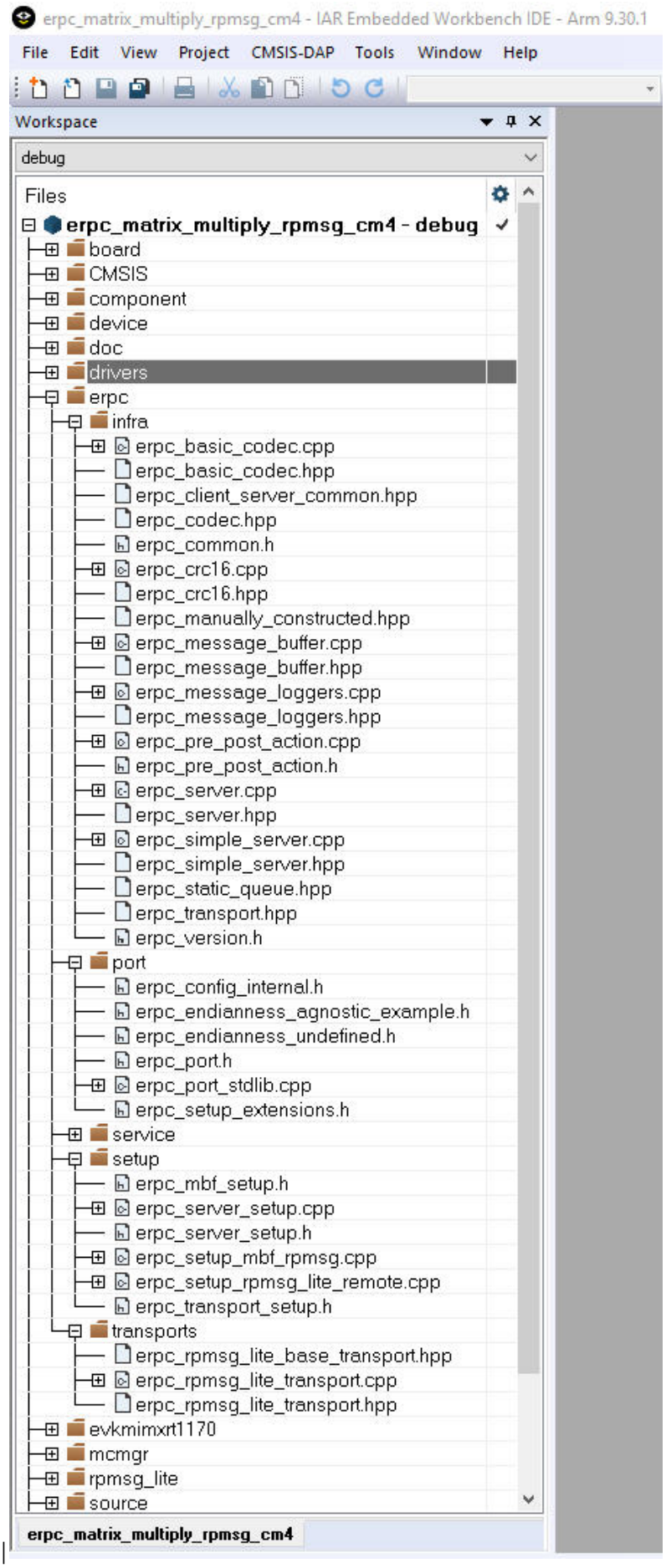
The eRPC infrastructure files are located in the following folder:

<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

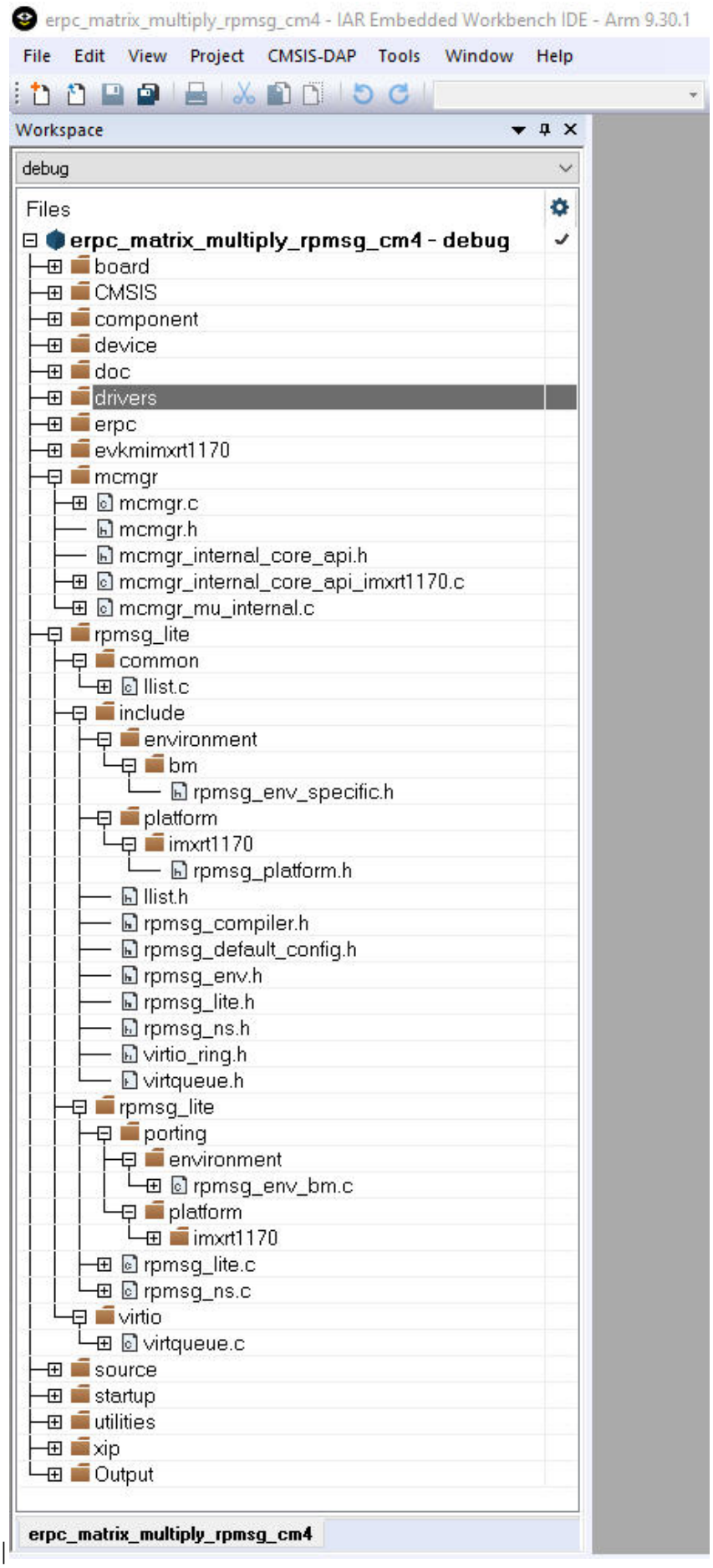
Server multicore infrastructure files

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

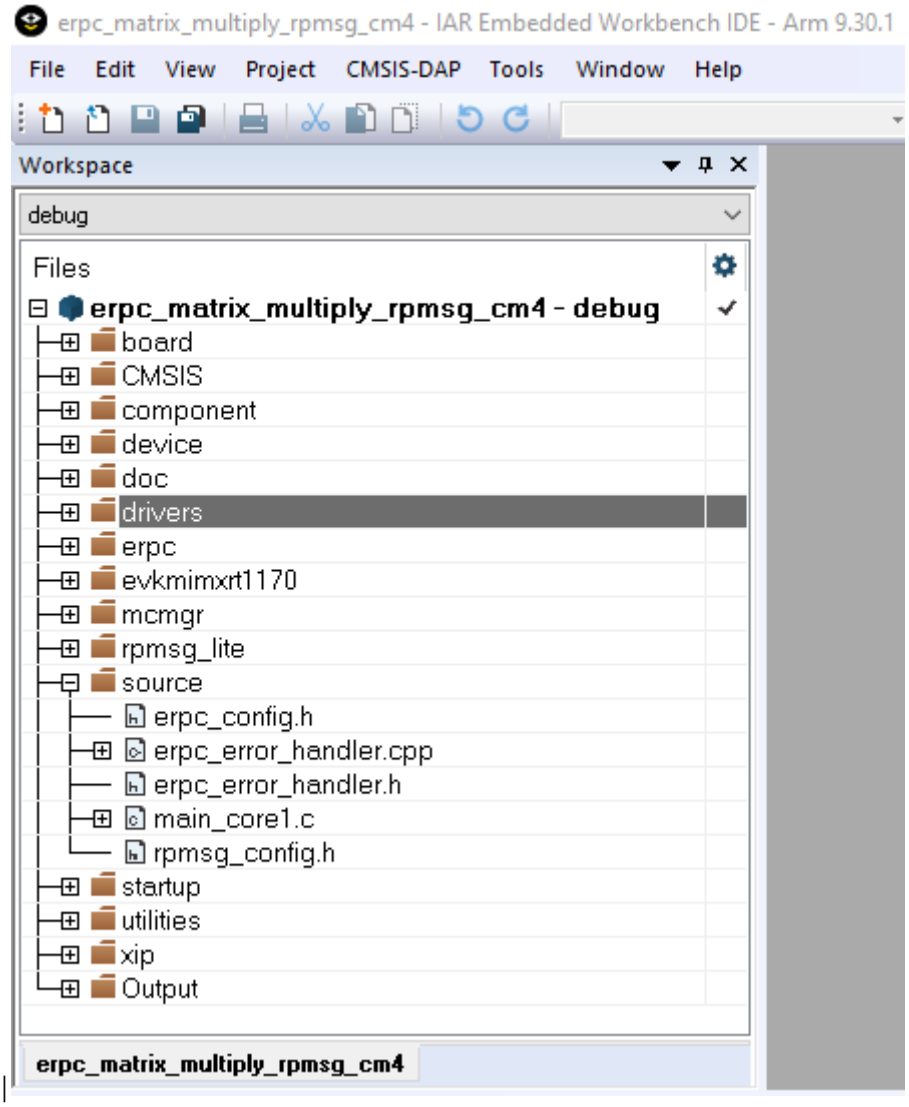
The main_core1.c file contains two functions:

- 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

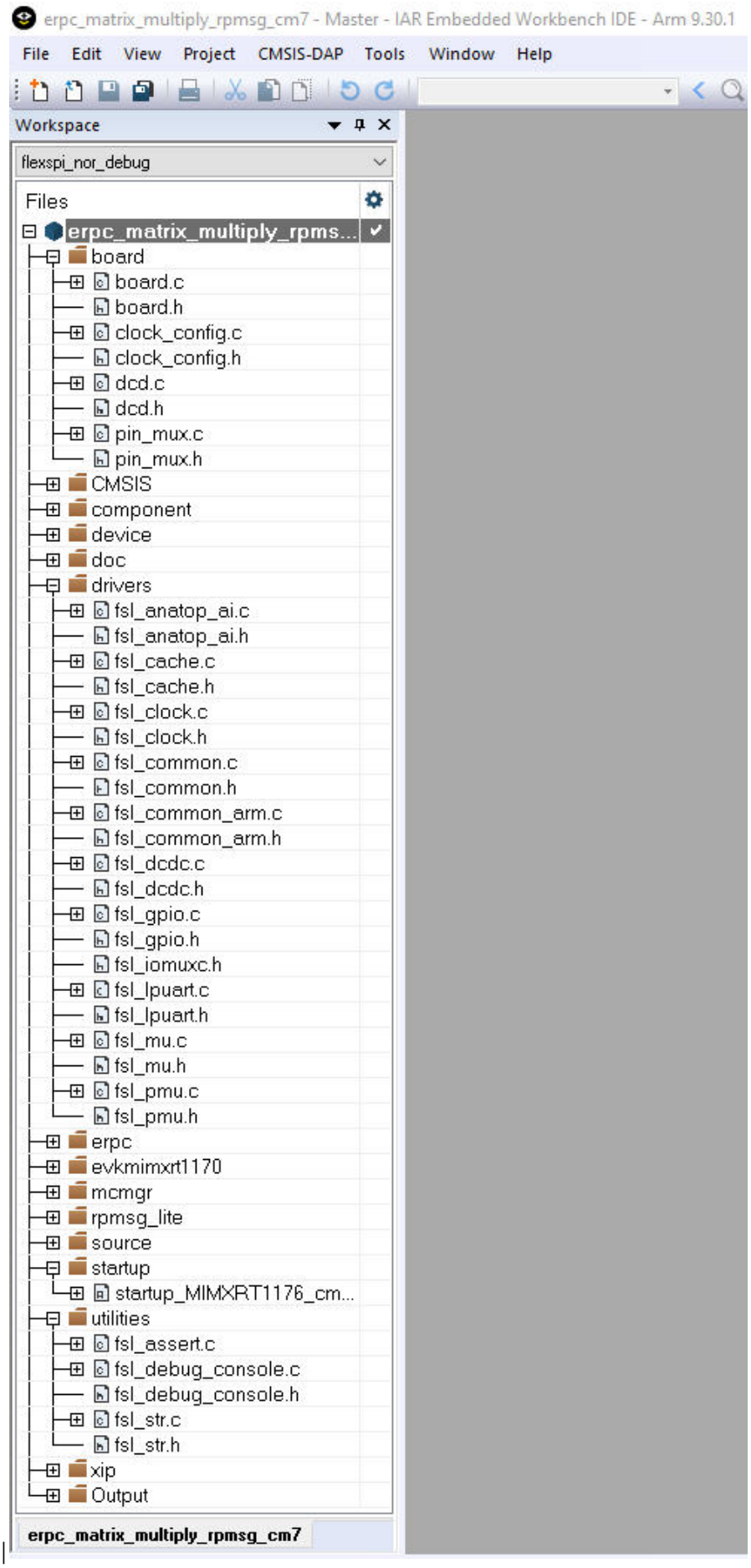
Project files for the eRPC client have the _cm7 suffix.

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

|

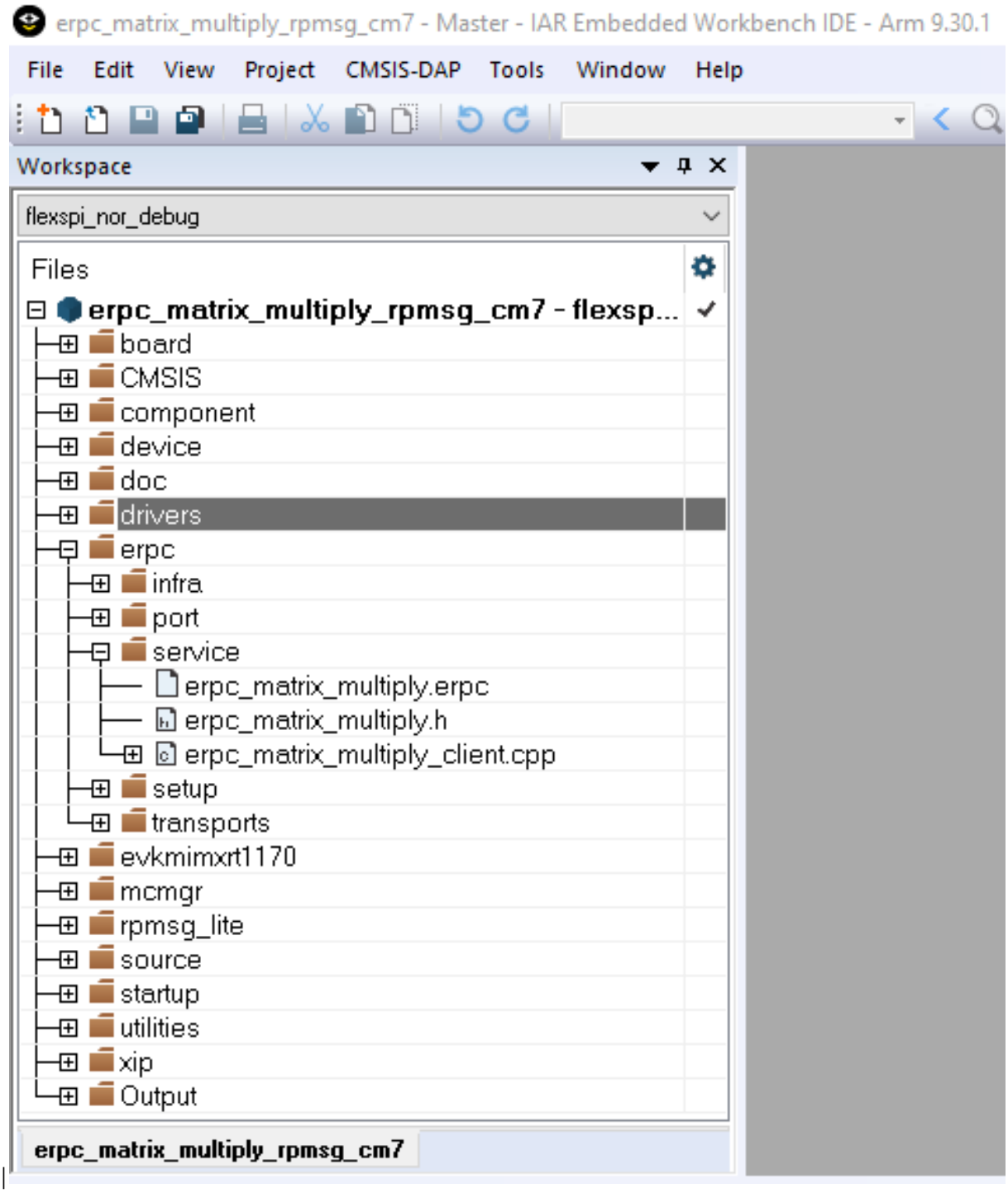
Parent topic: Multicore client application

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.

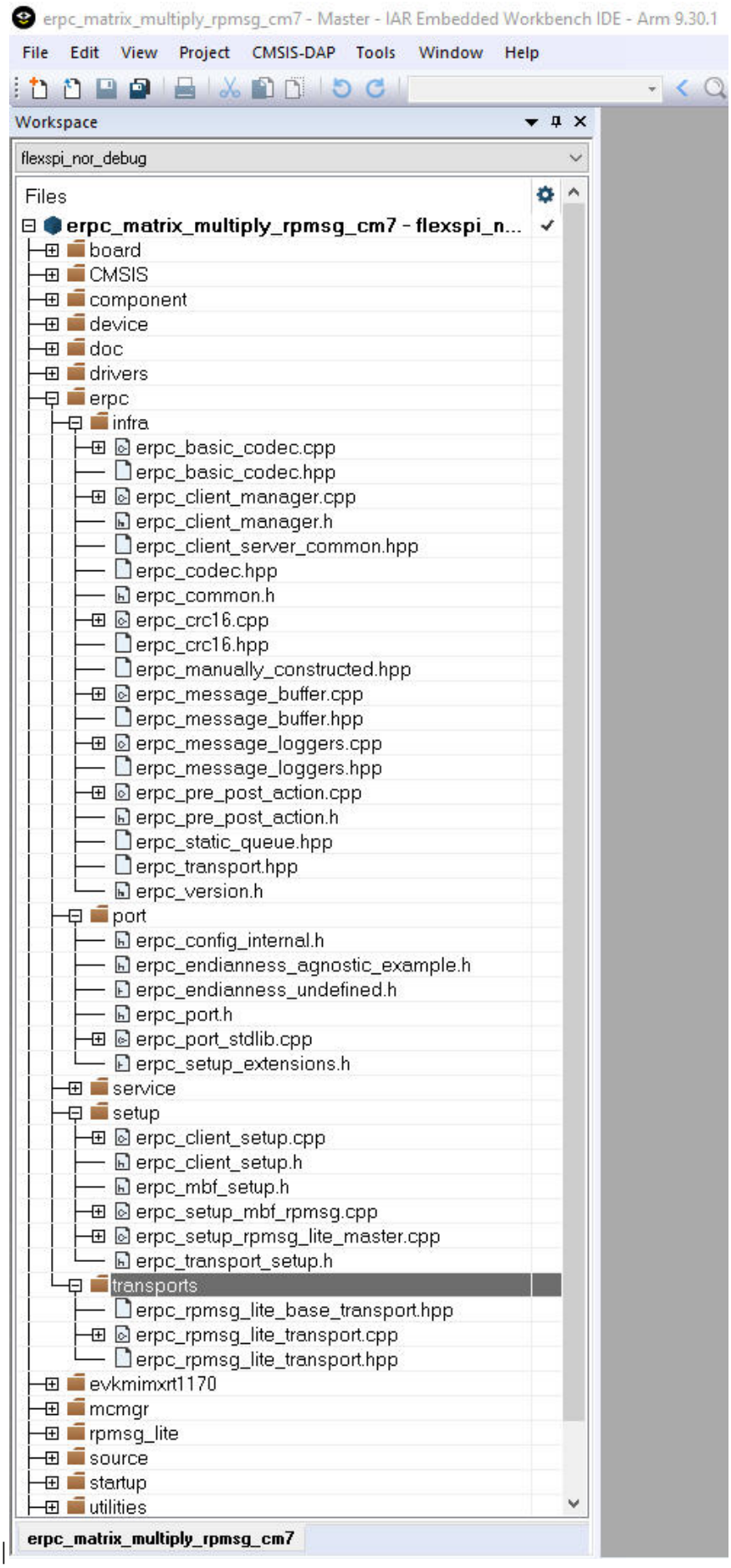
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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

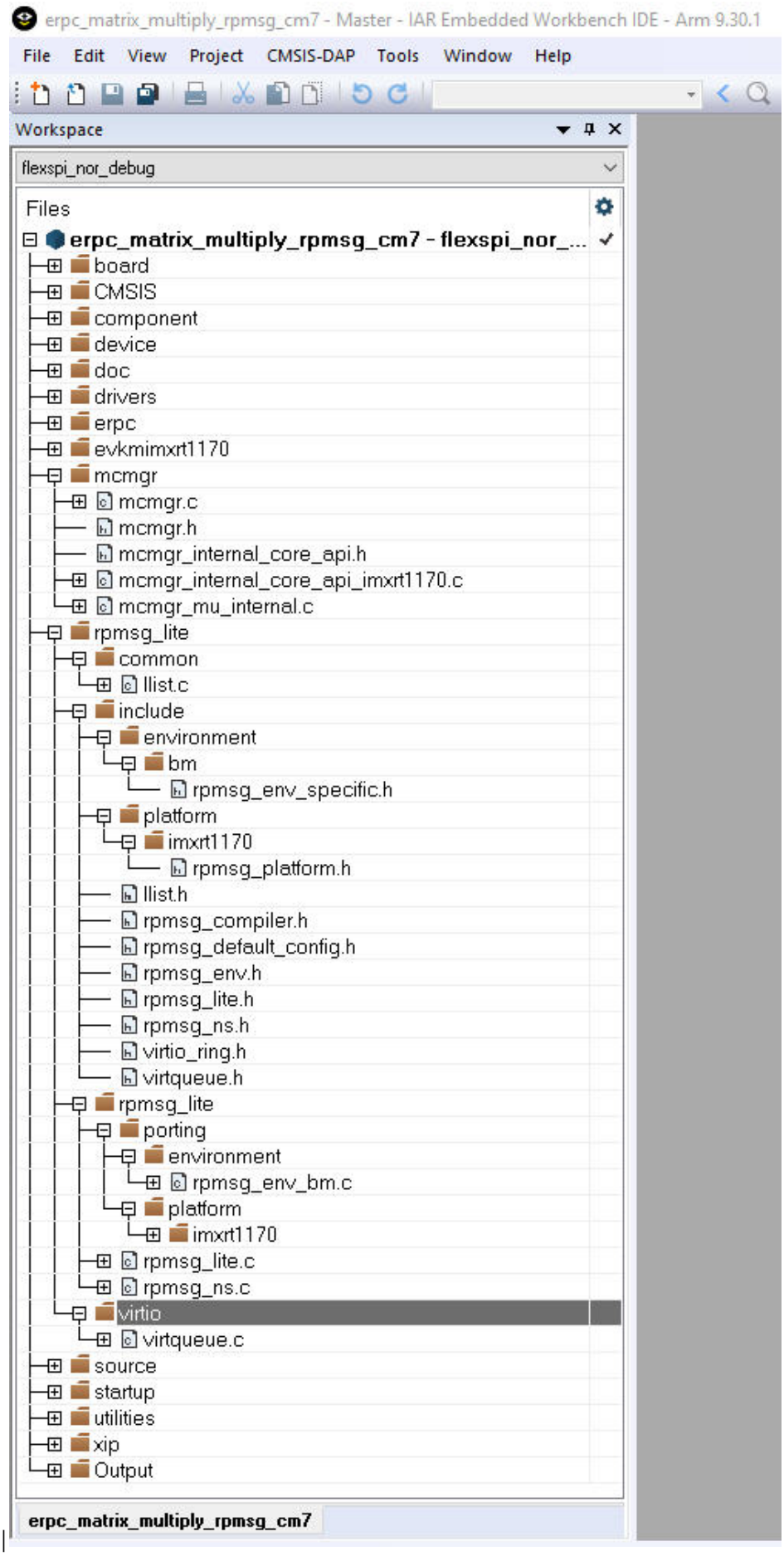
Client multicore infrastructure files

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.

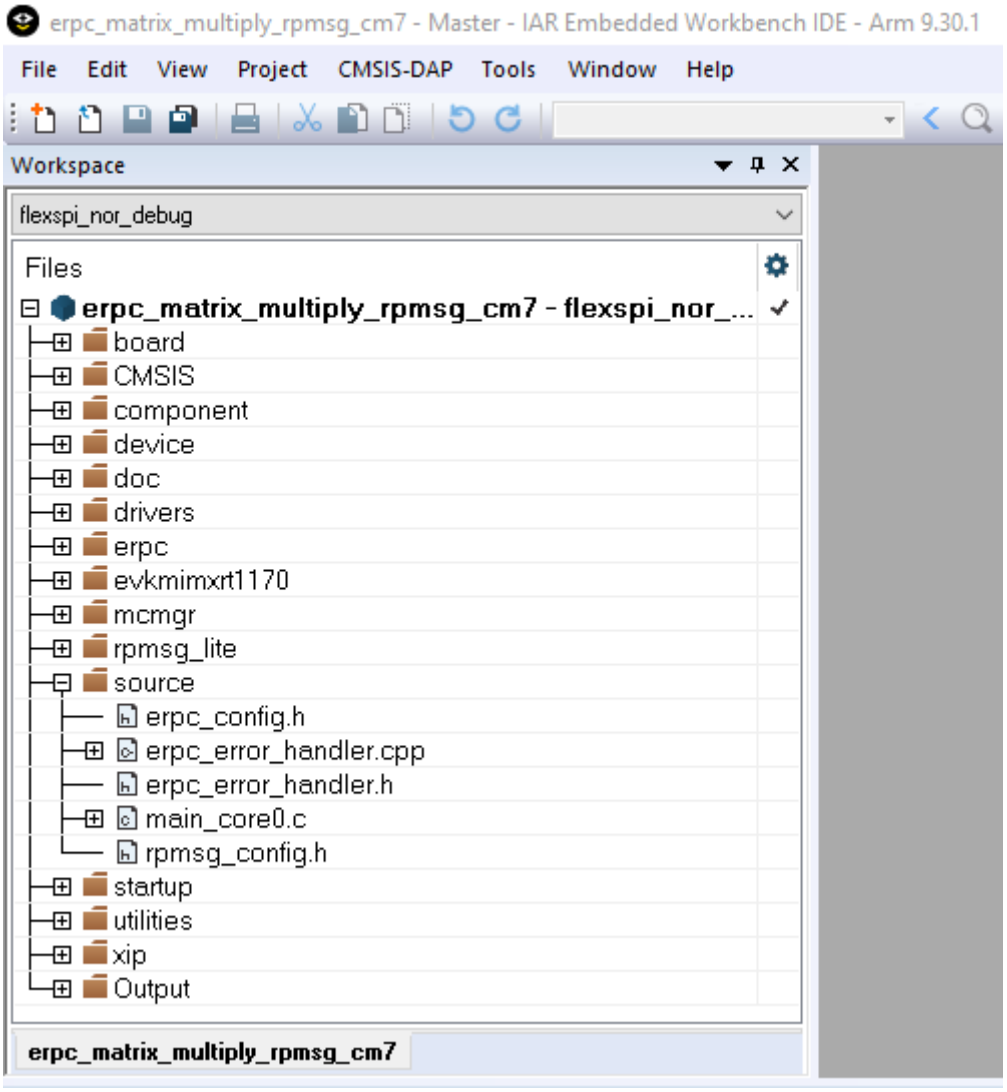
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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_lay* 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_<tr* 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;
  }
  ...
}
```

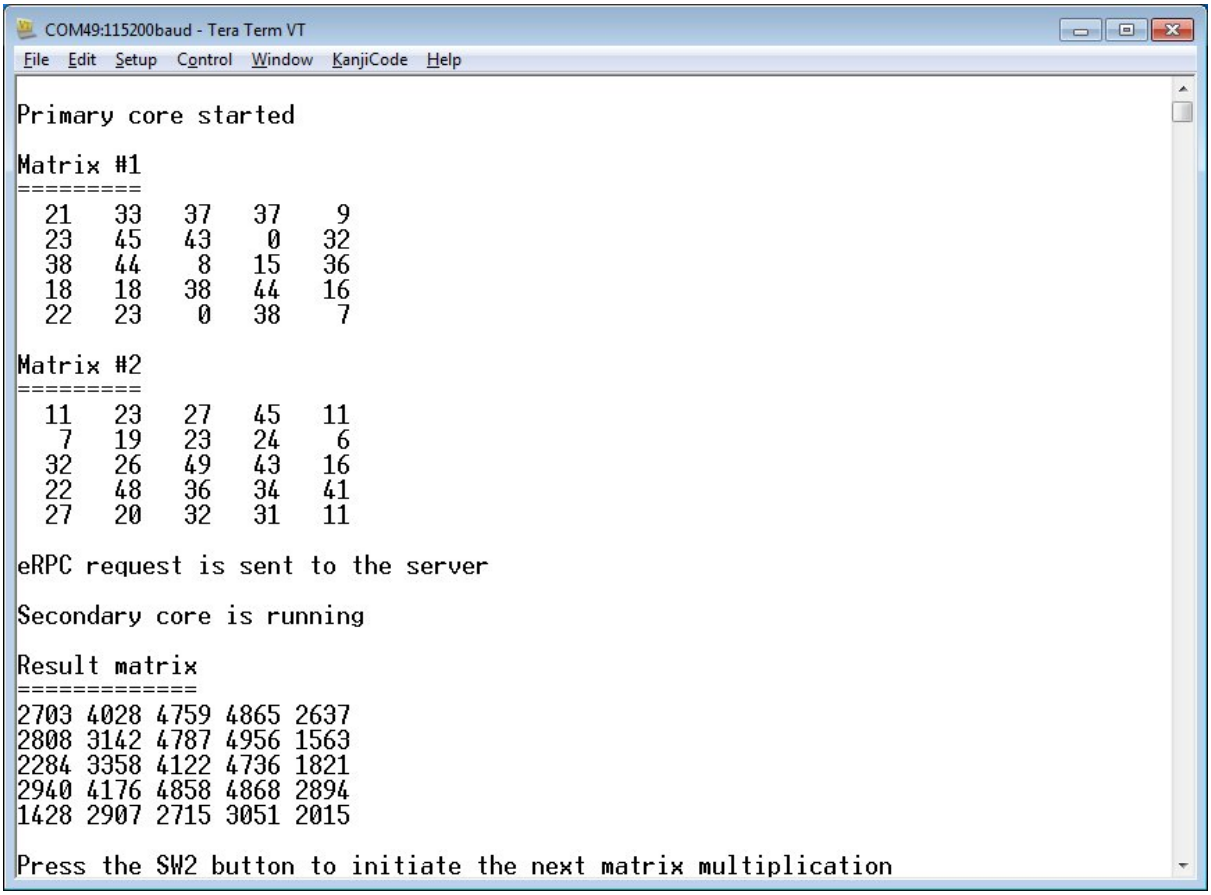
Parent topic: Multiprocessor client application

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/erpcgen/Windows|
|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:

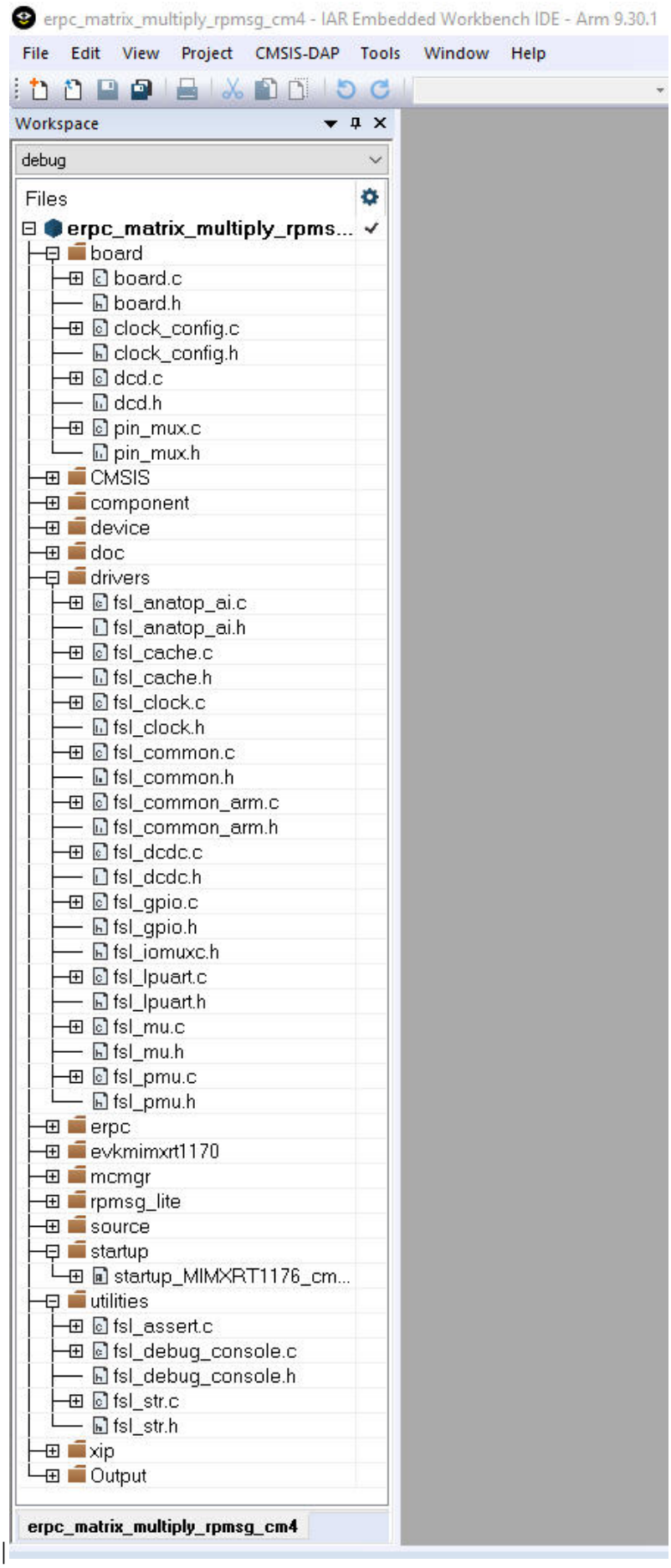
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

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

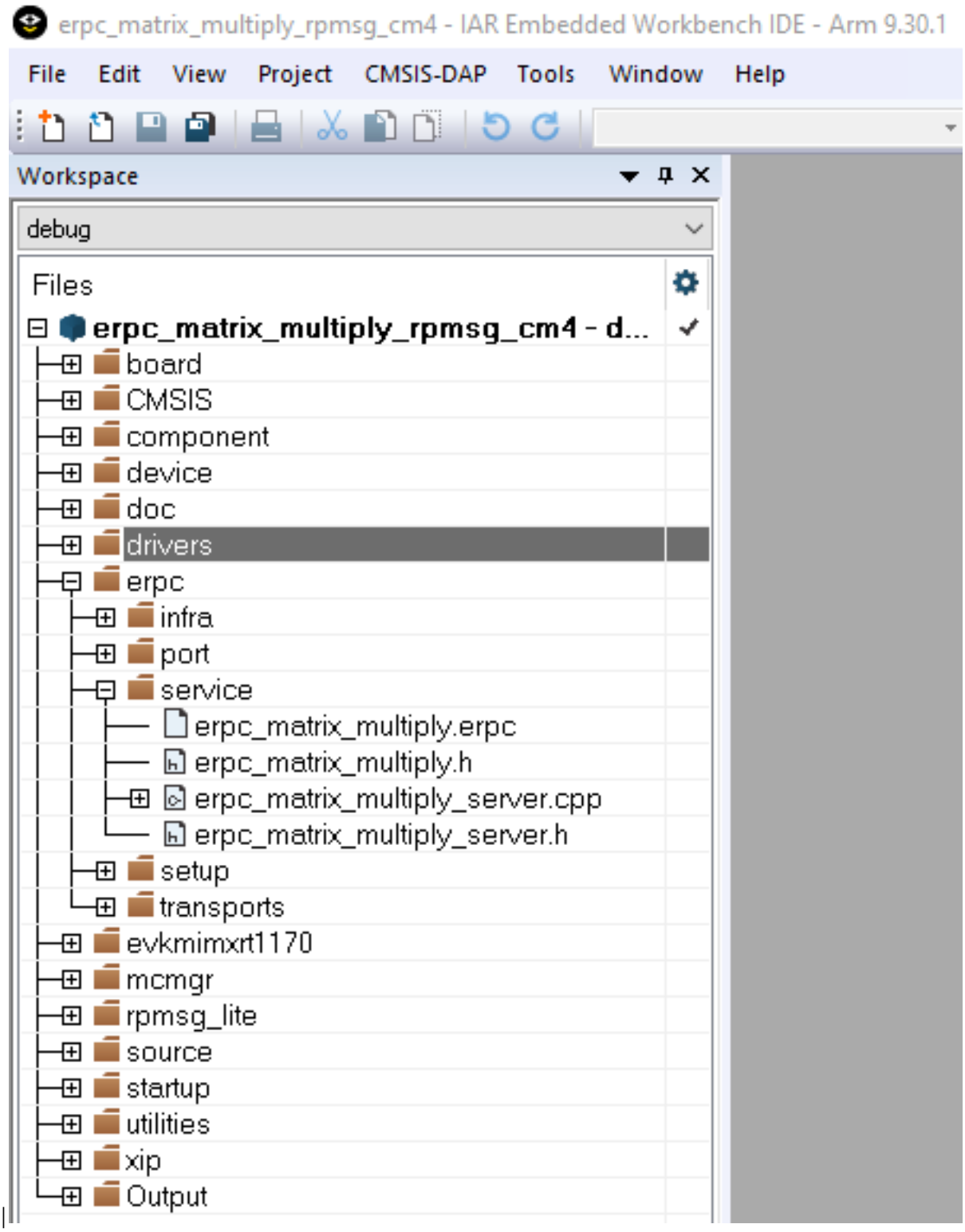
Server related generated files

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

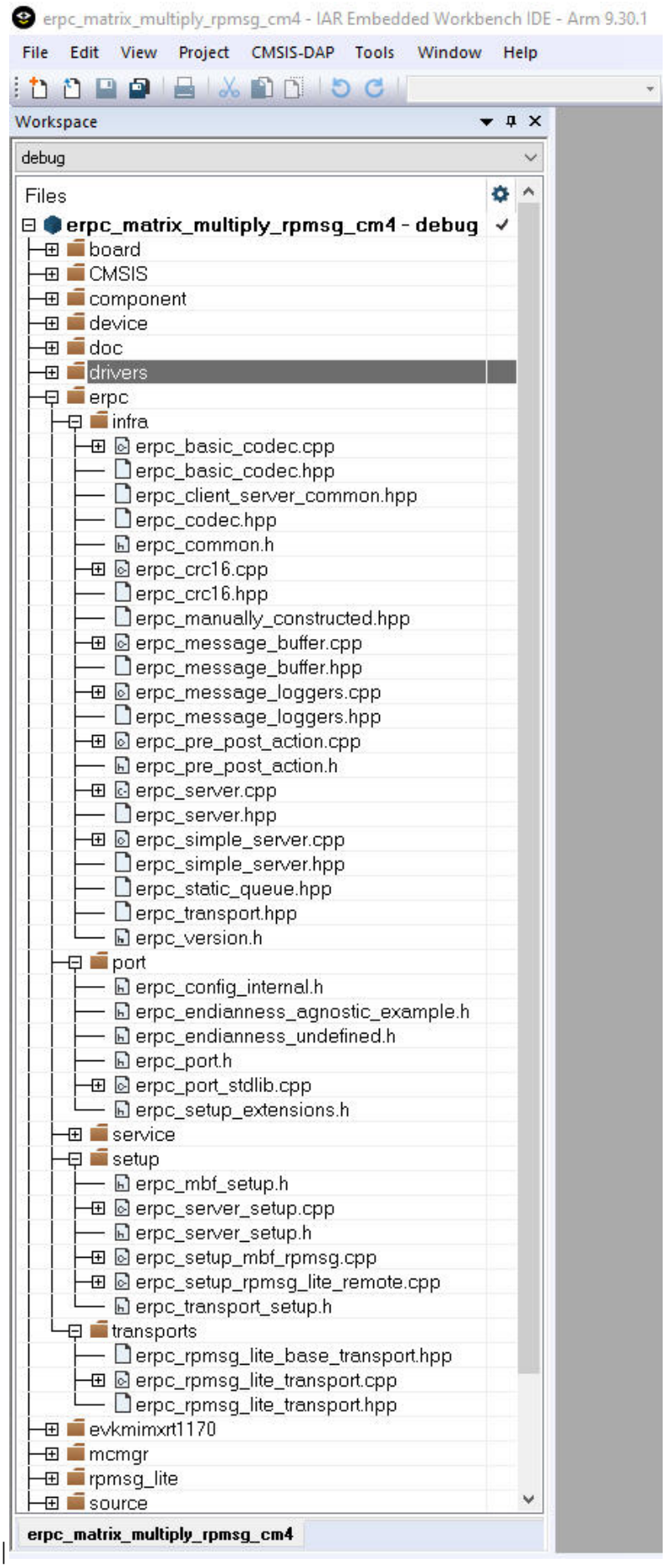
The eRPC infrastructure files are located in the following folder:

<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

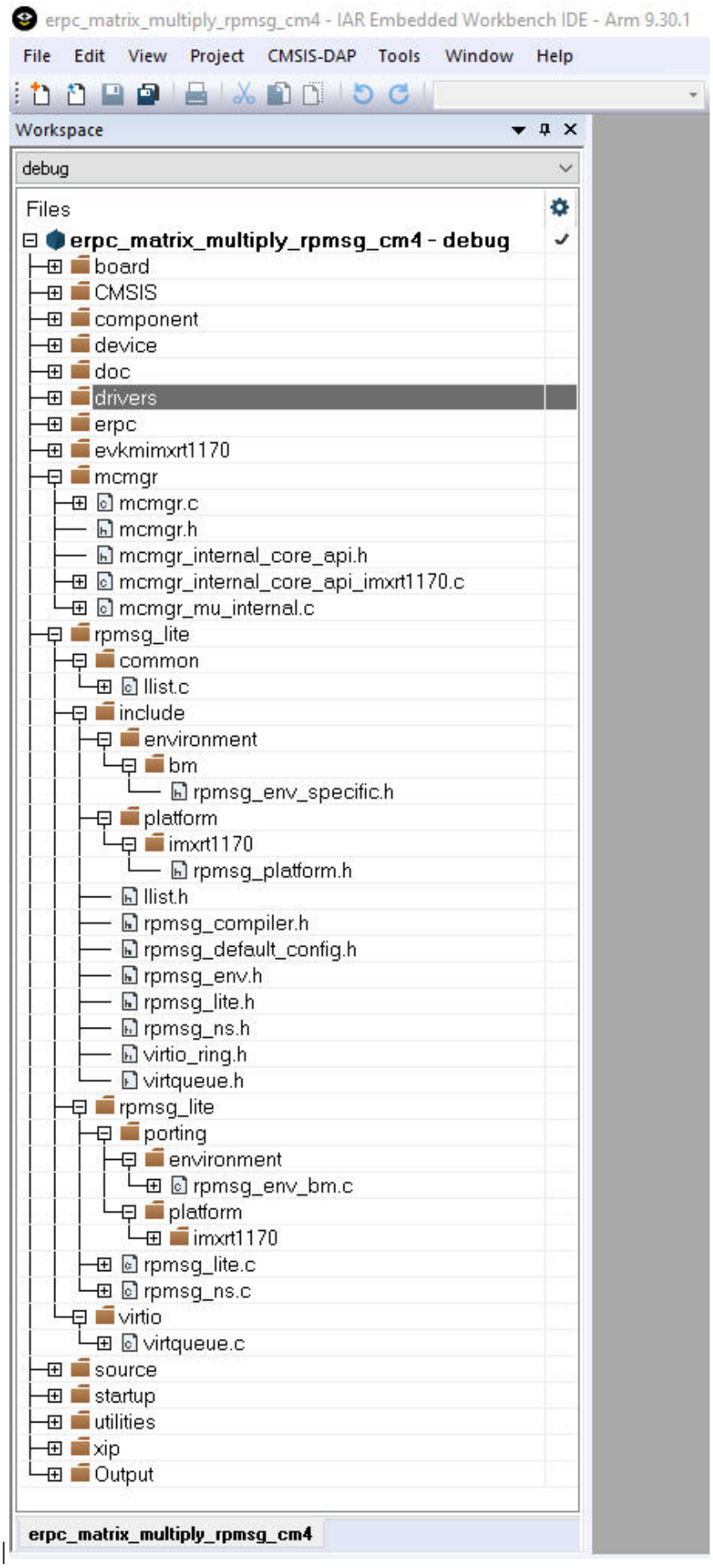
Server multicore infrastructure files

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

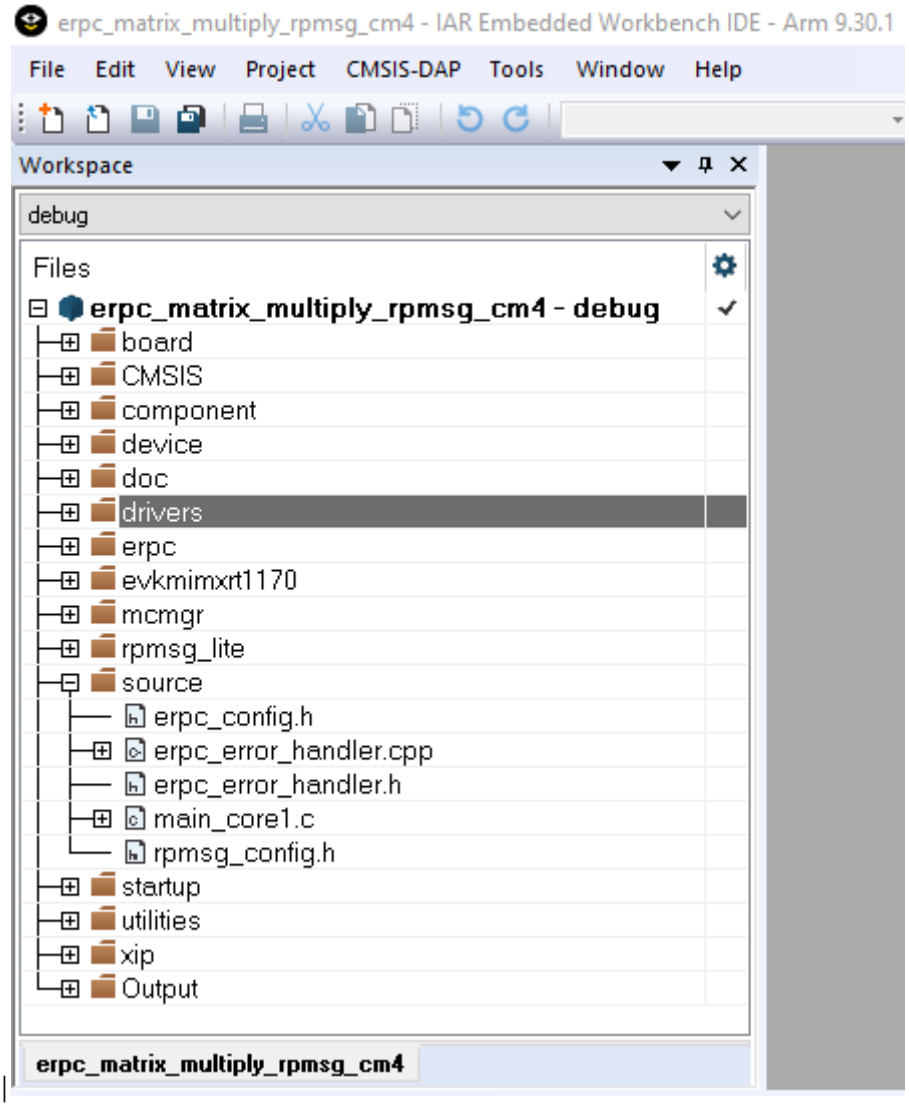
The main_core1.c file contains two functions:

- 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

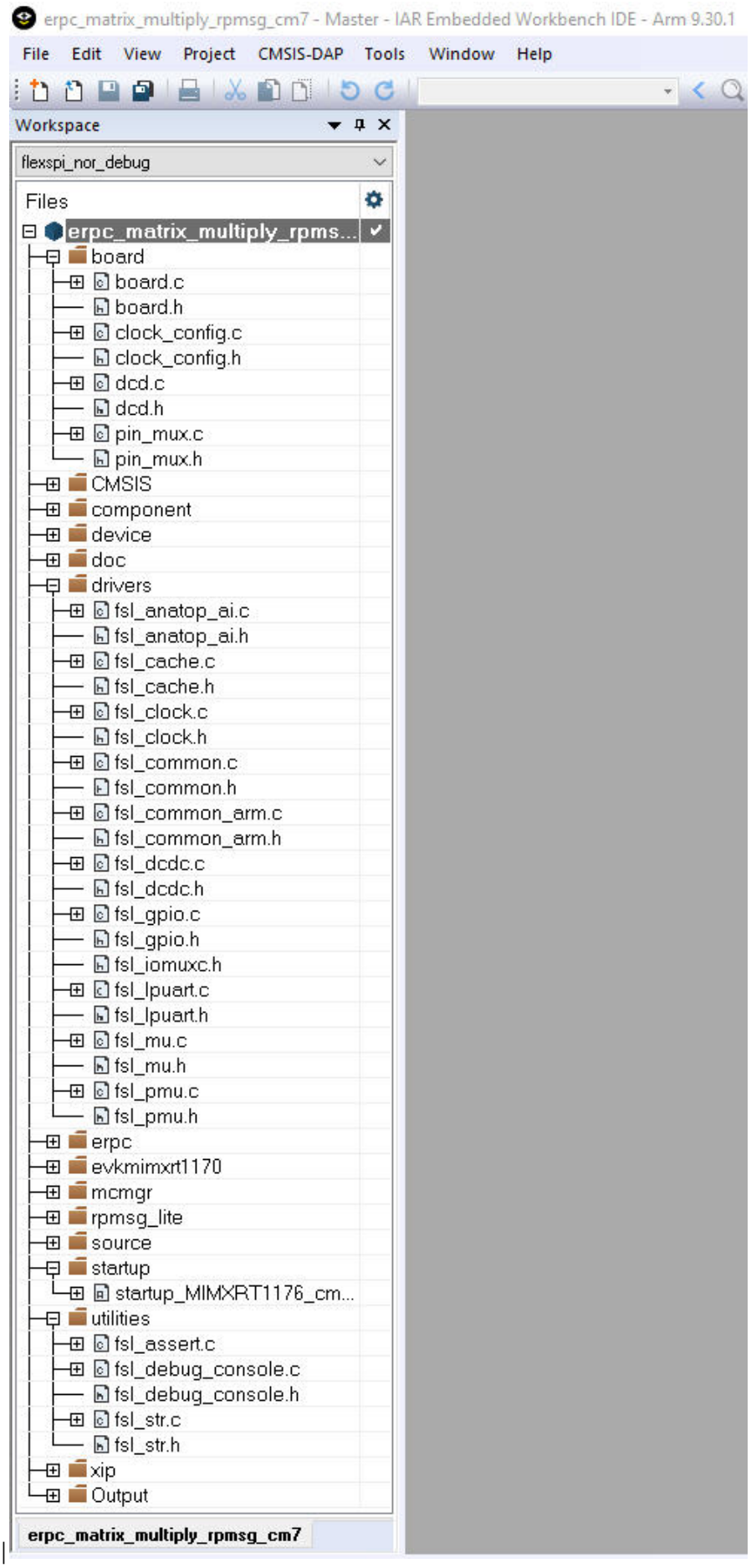
Project files for the eRPC client have the _cm7 suffix.

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

|

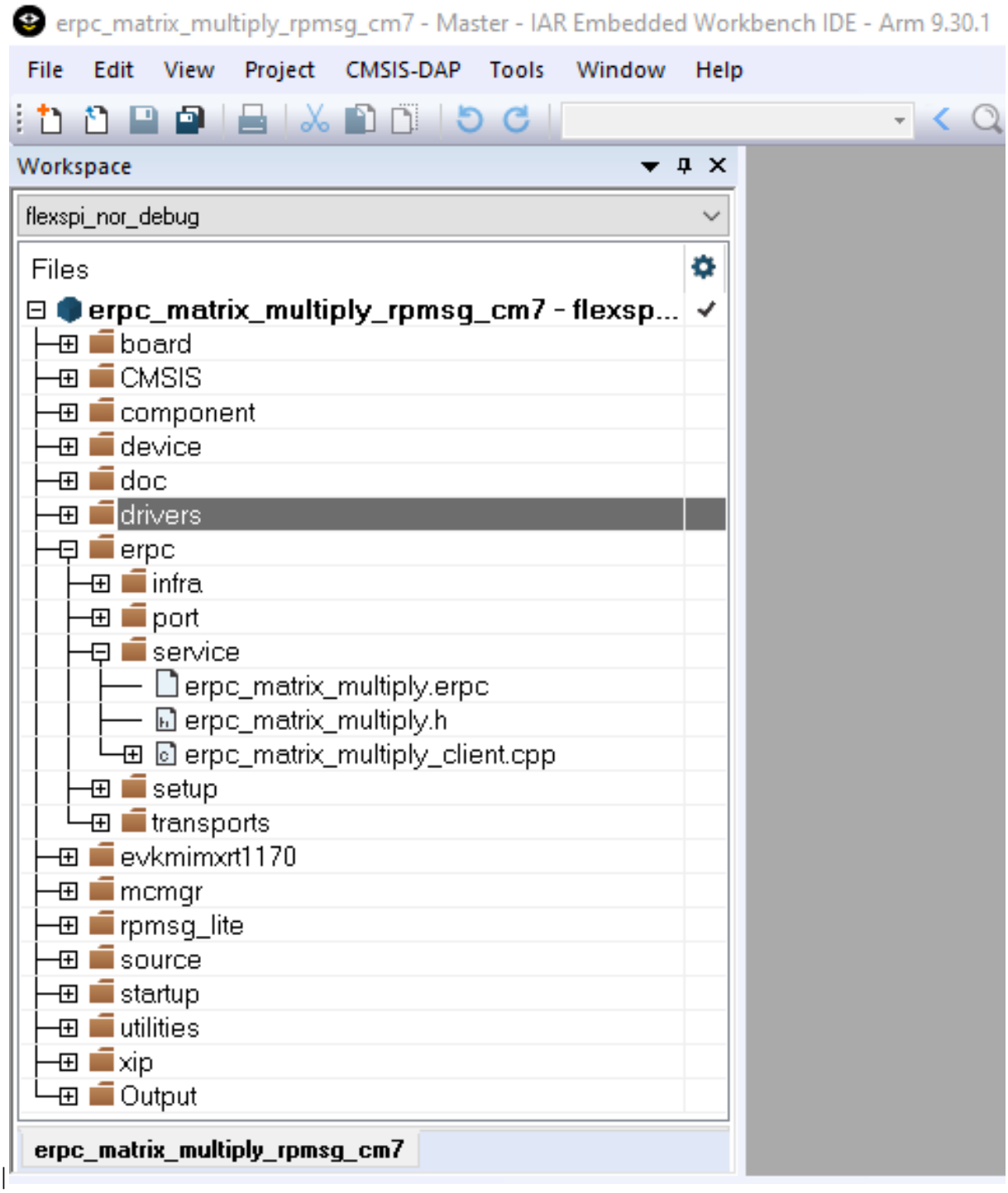
Parent topic: Multicore client application

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.

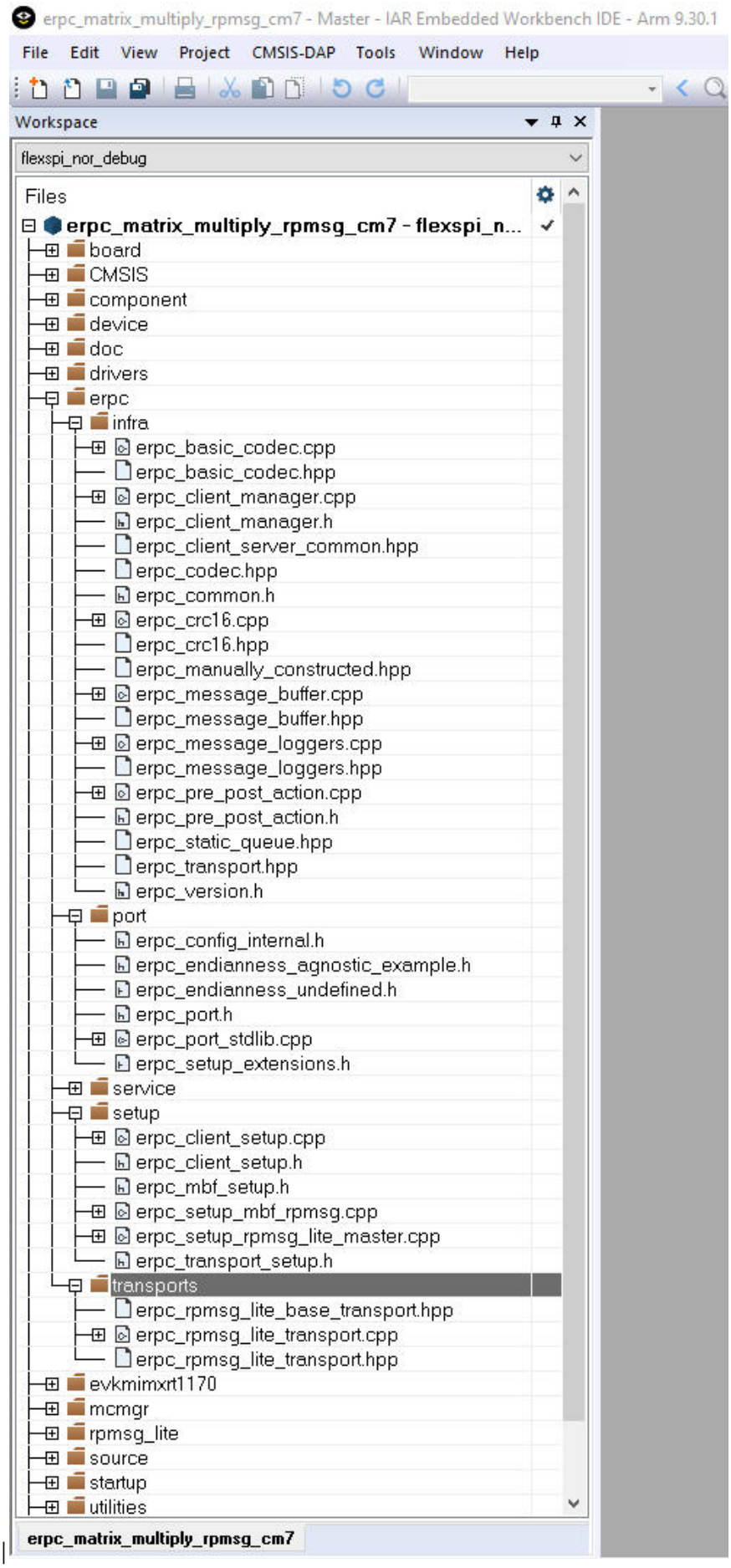
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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

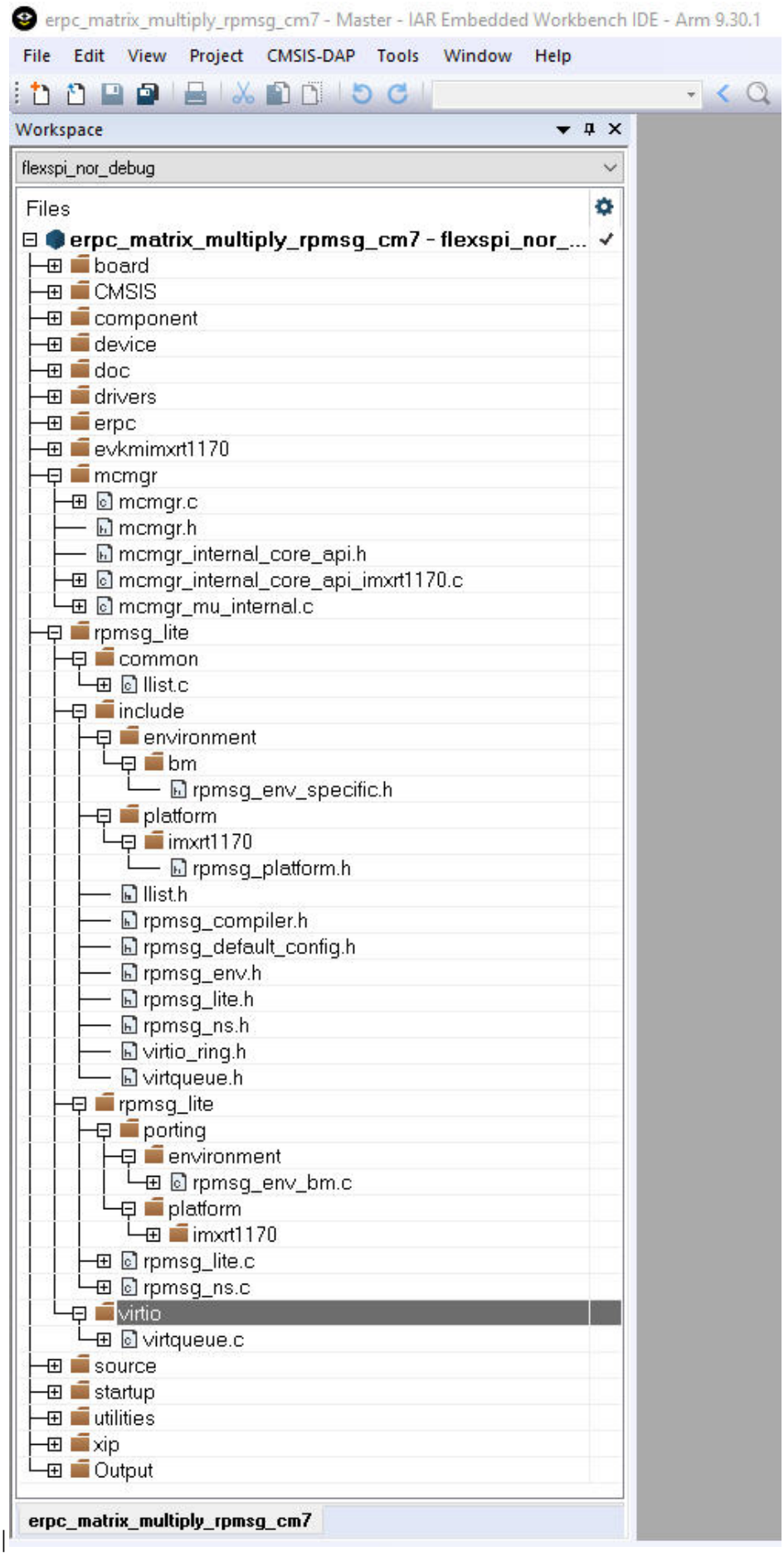
Client multicore infrastructure files

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.

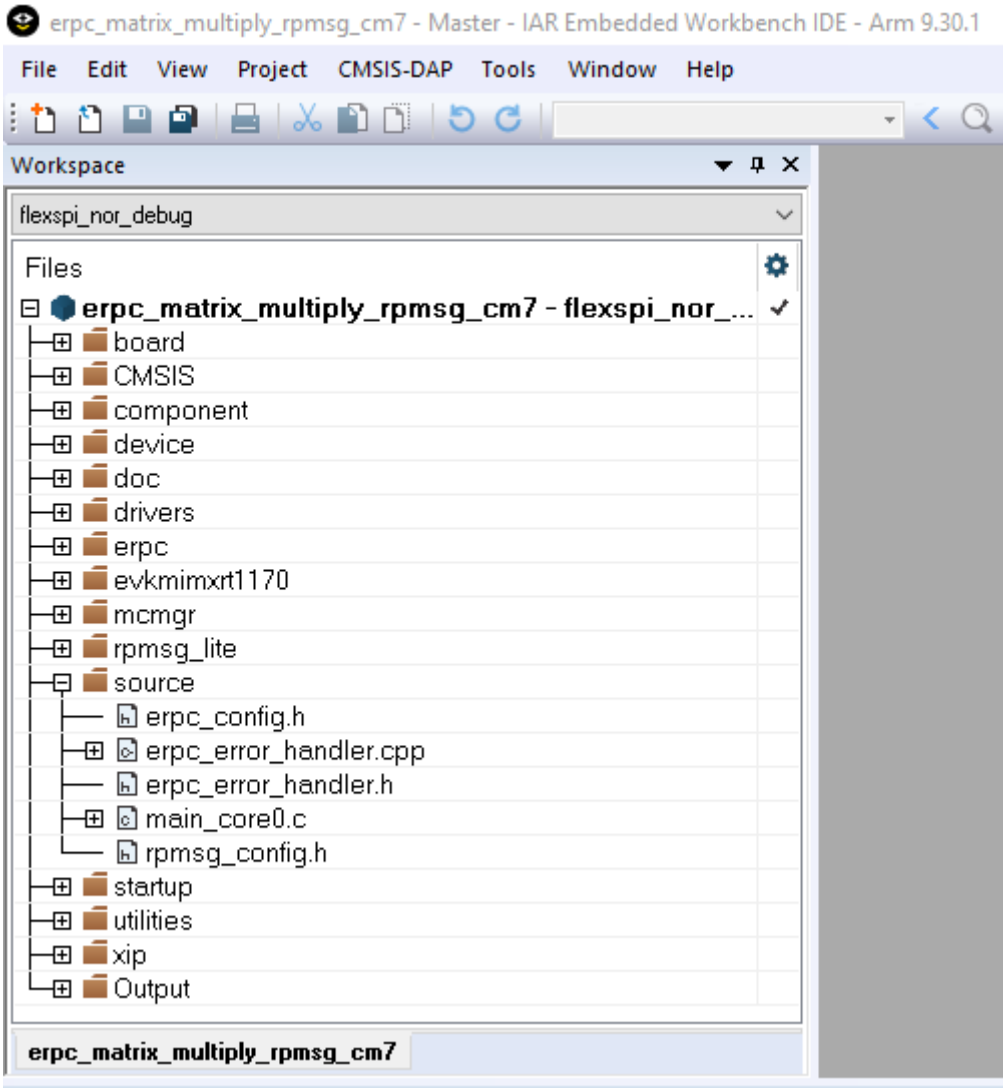
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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_lay* 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_<tr* 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;
  }
  ...
}
```

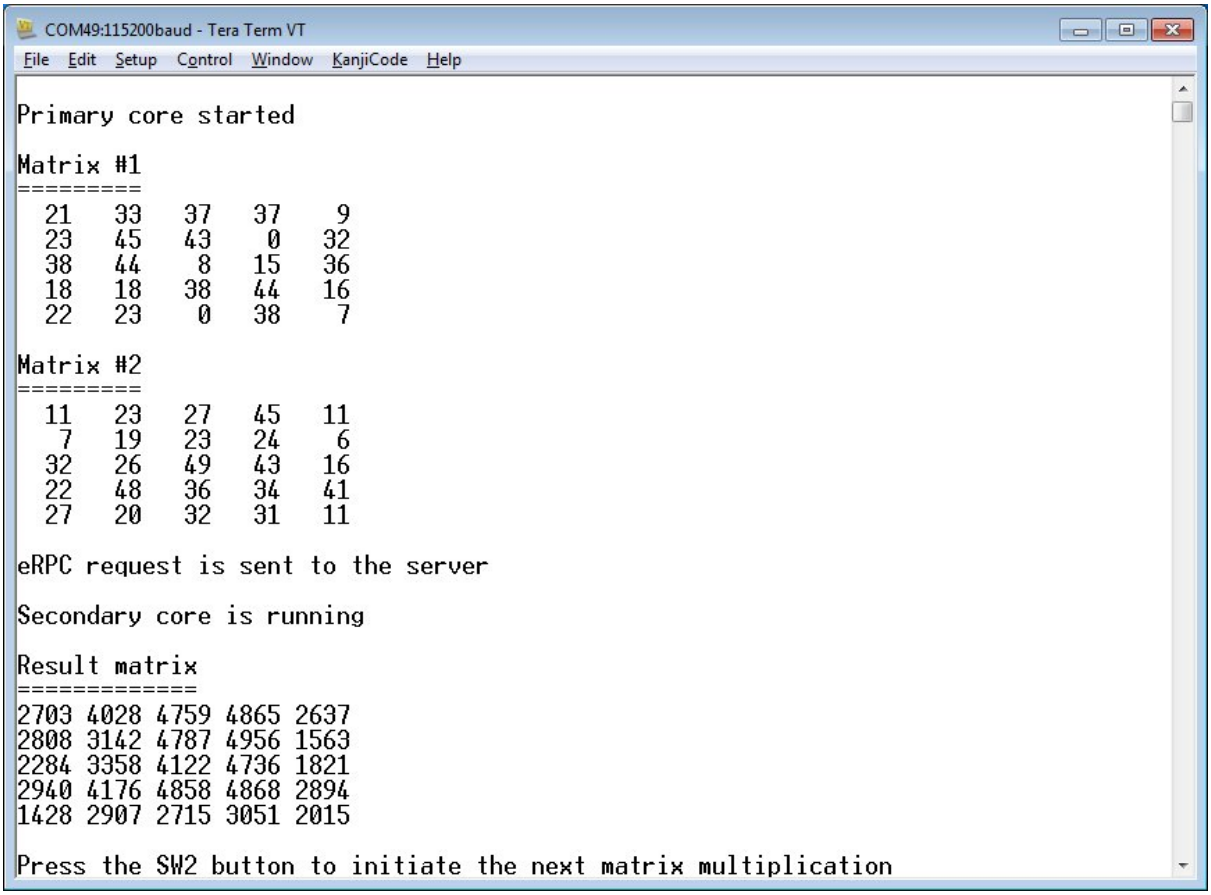
Parent topic: Multiprocessor client application

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

- Added Zephyr IPC Service transport layer support.
- Zephyr examples added support for RT600 and RT700.

Updated

- eRPC Zephyr module port updated for Zephyr version 4.4

Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.
- eRPC: Several MISRA violations addressed.

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

MCUXpresso SDK: FreeRTOS Kernel NXP Integration This repository is the NXP integration layer for the FreeRTOS kernel (https://github.com/FreeRTOS/FreeRTOS-Kernel). It allows original FreeRTOS kernel upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

For more information about the FreeRTOS kernel repo adoption see README_mcuxsdk.md: FreeRTOS kernel for MCUXpresso SDK Readme document.

FreeRTOS kernel for MCUXpresso SDK Overview

The purpose of this document is to describes the FreeRTOS kernel repo integration into the NXP MCUXpresso Software Development Kit: mcuxsdk. MCUXpresso SDK provides a comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on MCUs from NXP. This project involves the FreeRTOS kernel repo fork with:

- cmake and Kconfig support to allow the configuration and build in MCUXpresso SDK ecosystem
- FreeRTOS OS additions, such as FreeRTOS driver wrappers, RTOS ready FatFs file system, and the implementation of FreeRTOS tickless mode

The history of changes in FreeRTOS kernel repo for MCUXpresso SDK are summarized in *CHANGELOG_mcuxsdk.md* file.

The MCUXpresso SDK framework also contains a set of FreeRTOS examples which show basic FreeRTOS OS features. This makes it easy to start a new FreeRTOS project or begin experimenting with FreeRTOS OS. Selected drivers and middleware are RTOS ready with related FreeRTOS adaptation layer.

FreeRTOS example applications

The FreeRTOS examples are written to demonstrate basic FreeRTOS features and the interaction between peripheral drivers and the RTOS.

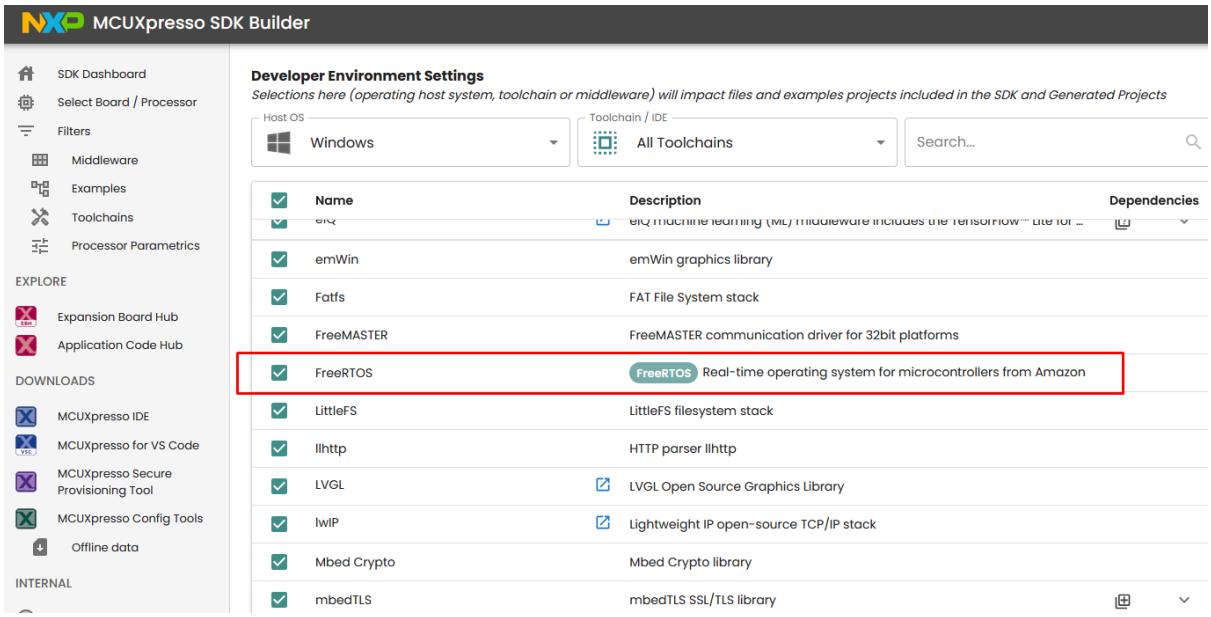
List of examples

The list of freertos_examples, their description and availability for individual supported MCUXpresso SDK development boards can be obtained here: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos_examples/index.html

Location of examples

The FreeRTOS examples are located in mcuxsdk-examples repository, see the freertos_examples folder.

Once using MCUXpresso SDK zip packages created via the MCUXpresso SDK Builder the FreeRTOS kernel library and associated freertos_examples are added into final zip package once FreeRTOS components is selected on the Developer Environment Settings page:



The FreeRTOS examples in MCUXpresso SDK zip packages are located in <MCUXpressoSDK_install_dir>/boards/<board_name>/freertos_examples/ subfolders.

Building a FreeRTOS example application

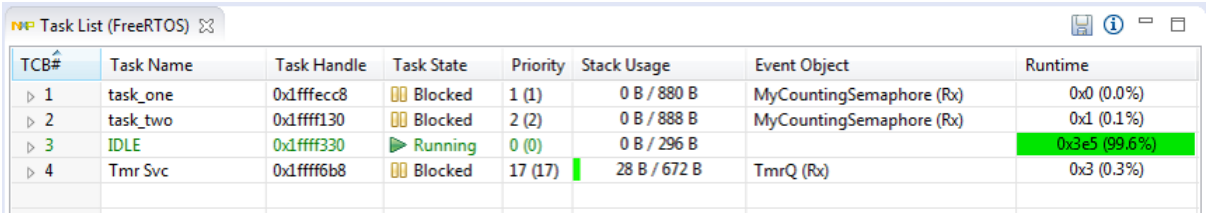
For information how to use the cmake and Kconfig based build and configuration system and how to build freertos_examples visit: MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide

Tip: To list all FreeRTOS example projects and targets that can be built via the west build command, use this west list_project command in mcuxsdk workspace:

```
west list_project -p examples/freertos_examples
```

FreeRTOS aware debugger plugin

NXP provides FreeRTOS task aware debugger for GDB. The plugin is compatible with Eclipse-based (MCUXpressoIDE) and is available after the installation.



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

MCUXpresso SDK: backoffAlgorithm Library NXP Integration This repository is the NXP integration layer for the backoffAlgorithm library (https://github.com/FreeRTOS/backoffalgorithm). It allows original backoffAlgorithm upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

MCUXpresso SDK: coreHTTP Client Library NXP Integration

This repository is the NXP integration layer for the coreHTTP Client library (https://github.com/FreeRTOS/corehttp). It allows original coreHTTP Client upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.5 corejson

JSON parser.

Readme

MCUXpresso SDK: coreJSON Library NXP Integration This repository is the NXP integration layer for the coreJSON library (https://github.com/FreeRTOS/corejson). It allows original coreJSON upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

MCUXpresso SDK: coreMQTT Library NXP Integration

This repository is the NXP integration layer for the coreMQTT library (https://github.com/FreeRTOS/coremqtt). It allows original coreMQTT upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.7 corepkcs11

PKCS #11 key management library.

Readme

MCUXpresso SDK: corePKCS11 Library NXP Integration This repository is the NXP integration layer for the PKCS #11 key management library (https://github.com/FreeRTOS/corePKCS11). It allows original PKCS #11 key management upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme

MCUXpresso SDK: FreeRTOS-Plus-TCP Library NXP Integration This repository is the NXP integration layer for the FreeRTOS-Plus-TCP library (https://github.com/FreeRTOS/freertos-plus-tcp). It allows original FreeRTOS-Plus-TCP upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.