MCUXpresso SDK Documentation

Release 25.12.00

NXP
Dec 18, 2025

Table of contents

1 FRDM-KE16Z **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 . . . 4
1.3.1 Getting Started with MCUXpresso SDK Repository . . . 4
1.4 Release Notes . . . 28
1.4.1 MCUXpresso SDK Release Notes . . . 28
1.5 ChangeLog . . . 31
1.5.1 MCUXpresso SDK Changelog . . . 31
1.6 Driver API Reference Manual . . . 78
1.7 Middleware Documentation . . . 78
1.7.1 FreeMASTER . . . 78
1.7.2 FreeRTOS . . . 78

2 MKE16Z4 **79**
2.1 ACMP: Analog Comparator Driver . . . 79
2.2 ADC12: Analog-to-Digital Converter . . . 85
2.3 Clock Driver . . . 91
2.4 CRC: Cyclic Redundancy Check Driver . . . 110
2.5 EWM: External Watchdog Monitor Driver . . . 113
2.6 FGPIO Driver . . . 116
2.7 C90TFS Flash Driver . . . 118
2.8 ftfx adapter . . . 118
2.9 Ftftx CACHE Driver . . . 118
2.10 ftfx controller . . . 120
2.11 ftfx feature . . . 136
2.12 Ftftx FLASH Driver . . . 137
2.13 Ftftx FLEXNVM Driver . . . 150
2.14 ftfx utilities . . . 161
2.15 FTM: FlexTimer Driver . . . 162
2.16 GPIO: General-Purpose Input/Output Driver . . . 188
2.17 GPIO Driver . . . 189
2.18 Common Driver . . . 191
2.19 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 204
2.20 LPI2C Master Driver . . . 205
2.21 LPI2C Slave Driver . . . 220
2.22 LPIT: Low-Power Interrupt Timer . . . 230
2.23 LPSPI: Low Power Serial Peripheral Interface . . . 237
2.24 LPSPI Peripheral driver . . . 237
2.25 LPTMR: Low-Power Timer . . . 258
2.26 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 264
2.27 LPUART Driver . . . 264
2.28 MCM: Miscellaneous Control Module . . . 282
2.29 MMDVSQ: Memory-Mapped Divide and Square Root . . . 287
2.30 MSCAN: Scalable Controller Area Network . . . 289

2.31 MSCAN Driver 289
2.32 PDB: Programmable Delay Block 304
2.33 PMC: Power Management Controller 314
2.34 PORT: Port Control and Interrupts 320
2.35 PWT: Pulse Width Timer 327
2.36 RCM: Reset Control Module Driver 331
2.37 RTC: Real Time Clock 337
2.38 SIM: System Integration Module Driver 344
2.39 SMC: System Mode Controller Driver 345
2.40 TRGMUX: Trigger Mux Driver 351
2.41 Tsi_v5_driver 352
2.42 WDOG32: 32-bit Watchdog Timer 378

3 Middleware **387**
3.1 Motor Control 387
3.1.1 FreeMASTER 387

4 RTOS **425**
4.1 FreeRTOS 425
4.1.1 FreeRTOS kernel 425
4.1.2 FreeRTOS drivers 425
4.1.3 backoffalgorithm 425
4.1.4 corehttp 425
4.1.5 corejson 425
4.1.6 coremqtt 426
4.1.7 corepkcs11 426
4.1.8 freertos-plus-tcp 426

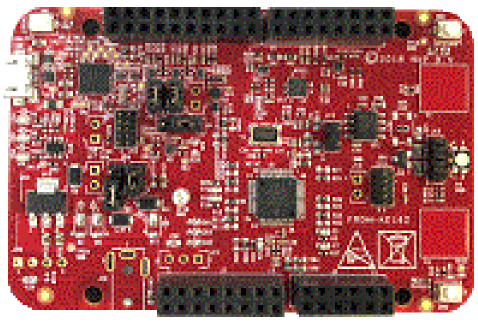
This documentation contains information specific to the frdmke16z board.

Chapter 1

FRDM-KE16Z

1.1 Overview

The FRDM-KE16Z Freedom Board is designed to work in standalone mode or as the main board of FRDM-TOUCH, FRDM-MC-LVBLDC, and Arduino boards. This Freedom board is compatible with DC 5v and 3.3v power supply, and features a KE16Z, a device boasting up to 64KB Flash and 8KB SRAM and numerous analog and digital peripherals. The on-board interfaces include an RGB LED, a 6-axis digital sensor, a 3-axis digital angular rate gyroscope, an ambient light sensor, CAN transceiver and two capacitive touch pads....



MCU device and part on board is shown below:

- Device: MKE16Z4
- PartNumber: MKE16Z64VLF4

1.2 Getting Started with MCUXpresso SDK Package

1.2.1 Getting Started with Package

- Overview
- MCUXpresso SDK board support package folders
  - Example application structure
  - Locating example application source files
- Run a demo using MCUXpresso IDE
  - Select the workspace location
  - Build an example application

  - Run an example application
- Run a demo application using IAR
  - Build an example application
  - Run an example application
- Run a demo using Keil® MDK/µVision
  - Install CMSIS device pack
  - Build an example application
  - Run an example application
- Run a demo using Arm® GCC
- MCUXpresso Config Tools
- MCUXpresso IDE New Project Wizard
- How to define IRQ handler in CPP files

1.3 Getting Started with MCUXpresso SDK GitHub

1.3.1 Getting Started with MCUXpresso SDK Repository

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

Overview

The GitHub Repository SDK approach offers:

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

Benefits of the Multi-Repository Approach

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

Setup and Configuration

Follow these steps to prepare your development environment:

Development Tools Installation

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

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

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

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

Essential Tools

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

Installation:

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

Setup:

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

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

Installation:

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

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

Installation:

```
pip install -U west
```

Minimum version: 1.2.0 or newer

Build System Tools

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

Recommended version: 3.30.0 or newer

Installation:

- **Windows**: Download .msi installer from cmake.org/download
- **Linux**: Use package manager or download from cmake.org

- **macOS**: Use Homebrew (`brew install cmake`) or download from cmake.org

Ninja - Fast Build System

What it does: Compiles your code quickly.

Minimum version: 1.12.1 or newer

Installation:

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

Ruby - IDE Project Generation (Optional)

What it does: Generates project files for IDEs like IAR and Keil.

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

Installation: Follow the Ruby environment setup guide

Compiler Toolchains

Choose and install the compiler toolchain you want to use:

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

Setting Up Environment Variables

After toolchain installation, set an environment variable so the build system locates it:

Windows:

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

Linux/macOS:

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

Verify Your Installation

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

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

Troubleshooting Installation Issues

"Command not found" errors:

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

Python/pip issues:

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

Slow downloads:

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

GitHub Repository Setup

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

Prerequisites

Verify the requirements:

System Requirements:

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

Verification Commands:

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

Workspace Initialization

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

Step 1: Initialize Workspace

Create and initialize your SDK workspace from GitHub:

Get the latest SDK from main branch:

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

Get SDK at specific revision:

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

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

Step 2: Choose Your Repository Update Strategy

Navigate to the SDK workspace:

```
cd mcuxpresso-sdk
```

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

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

```
west update
```

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

Best for:

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

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

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

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

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

Best for:

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

Examples:

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

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

Step 3: Verify Installation Confirm successful setup:

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

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

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

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

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

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

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

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

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

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

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

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

Benefits of Targeted Setup **Reduced Download Size**

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

Optimized Workspace

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

Flexible Development

- Switch between different board configurations easily
- Maintain separate workspaces for different projects
- Include optional components as needed

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

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

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

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

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

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

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

Workspace Management

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

For Complete SDK Workspace:

```
# Update manifest repository
cd manifests
git pull

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

For Targeted Workspace:

```
# Update manifest repository
cd manifests
git pull

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

Workspace Status Check workspace synchronization status:

```
# Show status of all repositories
west status

# Show detailed information about repositories
west list
```

Troubleshooting **Network Issues:**

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

Permission Issues:

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

Disk Space:

- Full SDK workspace requires approximately 7-8 GB
- Targeted workspace typically requires 1-2 GB

- Use board-specific setup to reduce workspace size

Repository Management Issues:

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

Next Steps

With your workspace initialized:

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

For advanced repository management, see the west tool documentation.

Explore SDK Structure and Content

Learn about the organization of the SDK and its components:

SDK Architecture Overview

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

Repository Organization

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

Manifest Repository

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

Base Repositories

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

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

Middleware Repositories

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

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

- **Safety**: IEC60730B safety libraries
- **Motor Control**: Motor control and real-time control libraries

Internal Repositories

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

Repository Hosting

Public repositories are hosted on GitHub under these organizations:

- nxp-mcuxpresso
- NXP
- nxp-zephyr

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

Benefits of This Architecture

Selective Integration: Projects include only required components, reducing memory footprint and build complexity.

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

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

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

Workspace Management

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

Workspace Structure

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

Top-Level Organization

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

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

SDK Component Layout

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

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

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

Common Example Files

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

Board-Specific Files

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

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

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

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

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

Documentation Structure SDK documentation is distributed across multiple locations:

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

For complete documentation, refer to the online documentation.

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

1. General README: `examples/demo_apps/hello_world/readme.md`

- What the example does
- General functionality description
- Common usage information

2. Board-Specific README: `examples/_boards/{board_name}/demo_apps/hello_world/example_board_readme.md`

- Board-specific setup instructions
- Hardware connections required
- Board-specific behavior notes

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

Development Workflows

Get started with building and running projects:

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

Prerequisites

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

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

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

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

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

Basic Build Process

Simple Build Build the hello_world example with default settings:

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

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

Specifying Configuration

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

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

Alternative Toolchains

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

# Other toolchains as supported by the example
```

Multicore Applications

For multicore devices, specify the core ID:

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

For multicore projects using sysbuild:

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

Flash an Application

Flash the built application to your board:

```
west flash -r linkserver
```

Debug

Start a debug session:

```
west debug -r linkserver
```

Common Build Options

Clean Build

Force a complete rebuild:

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

Dry Run

See the commands that get executed without running them:

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

Device Variants

For boards supporting multiple device variants:

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

Project Configuration

CMake Configuration Only

Run configuration without building:

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

Interactive Configuration

Launch the configuration GUI:

```
west build -t guiconfig
```

Troubleshooting

Build Failures

Use pristine builds to resolve dependency issues:

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

Getting Help View the help information for west build:

```
west build -h
```

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

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

Next Steps

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

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

Prerequisites

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

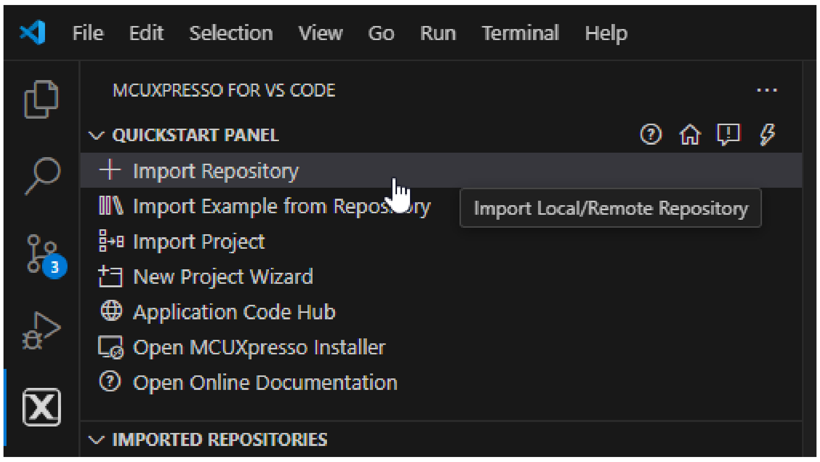
Extension Installation

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

SDK Import and Setup

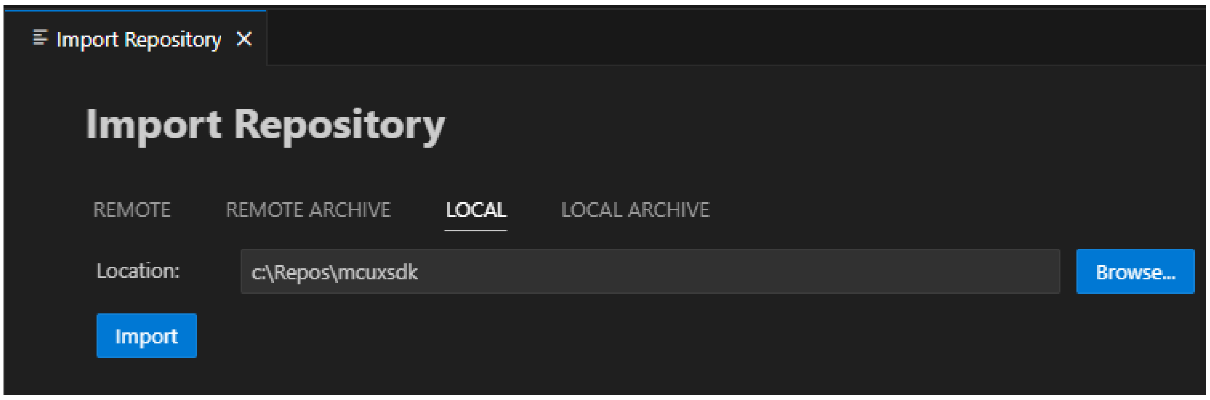
Import Methods The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

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

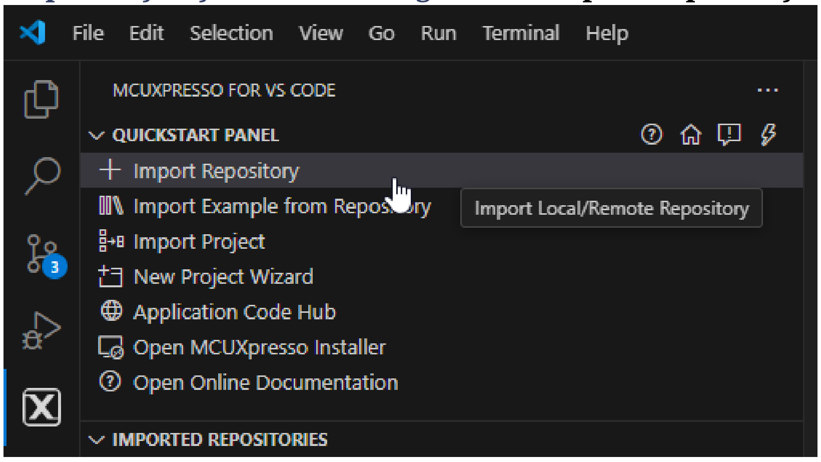

File Edit Selection View Go Run Terminal Help
MCUXPRESSO FOR VS CODE
QUICKSTART PANEL
Import Repository
Import Example from Repository
Import Project
New Project Wizard
Application Code Hub
Open MCUXpresso Installer
Open Online Documentation
Import Local/Remote Repository
IMPORTED REPOSITORIES

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

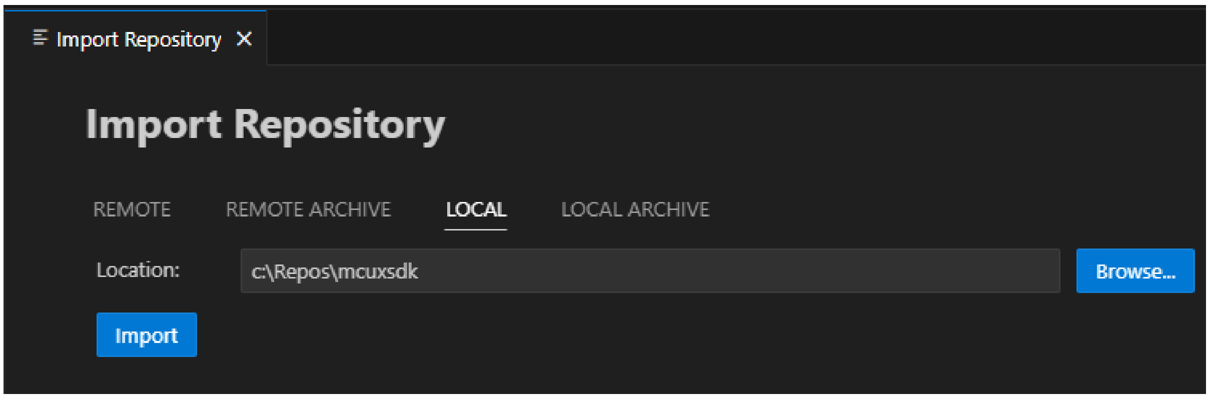
Select **Local** if you've already obtained the SDK according to *setting up the repo*. Select your location and click **Import**.



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

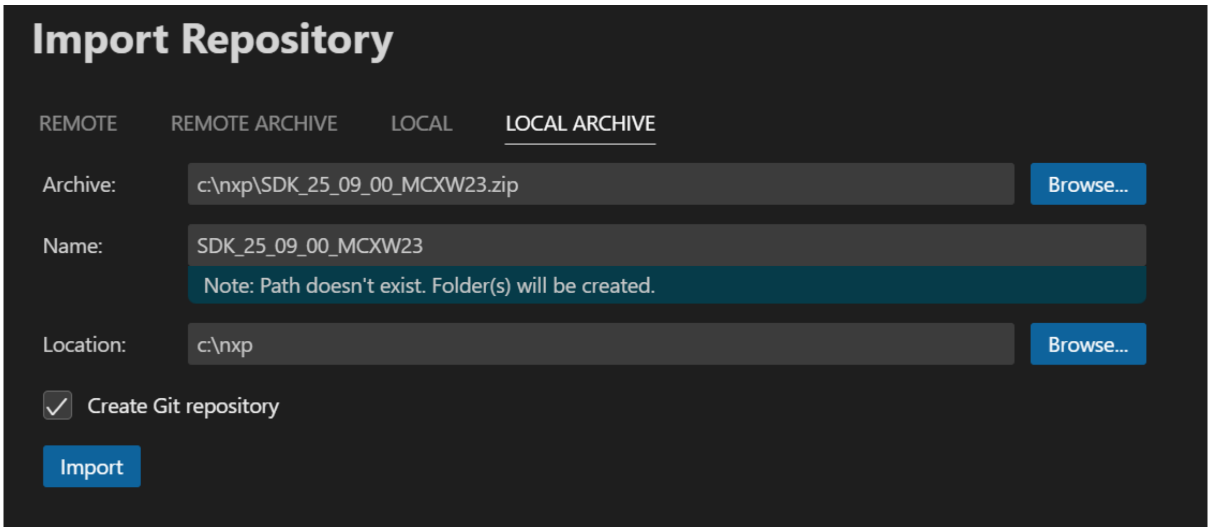

File Edit Selection View Go Run Terminal Help
MCUXPRESSO FOR VS CODE
QUICKSTART PANEL
Import Repository
Import Example from Repository
Import Project
New Project Wizard
Application Code Hub
Open MCUXpresso Installer
Open Online Documentation
Import Local/Remote Repository
IMPORTED REPOSITORIES

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



Location: c:\Repos\mcuxsdk

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**.



Archive: c:\nxp\SDK_25_09_00_MCXW23.zip

Name: SDK_25_09_00_MCXW23

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

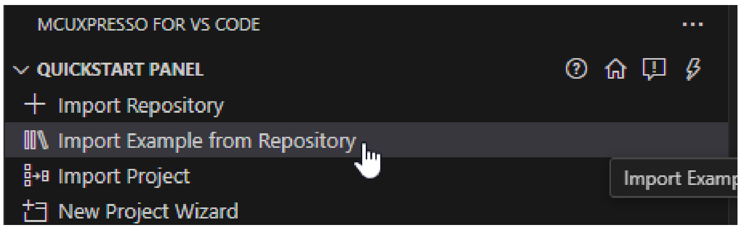
Location: c:\nxp

✓ Create Git repository

Building Example Applications

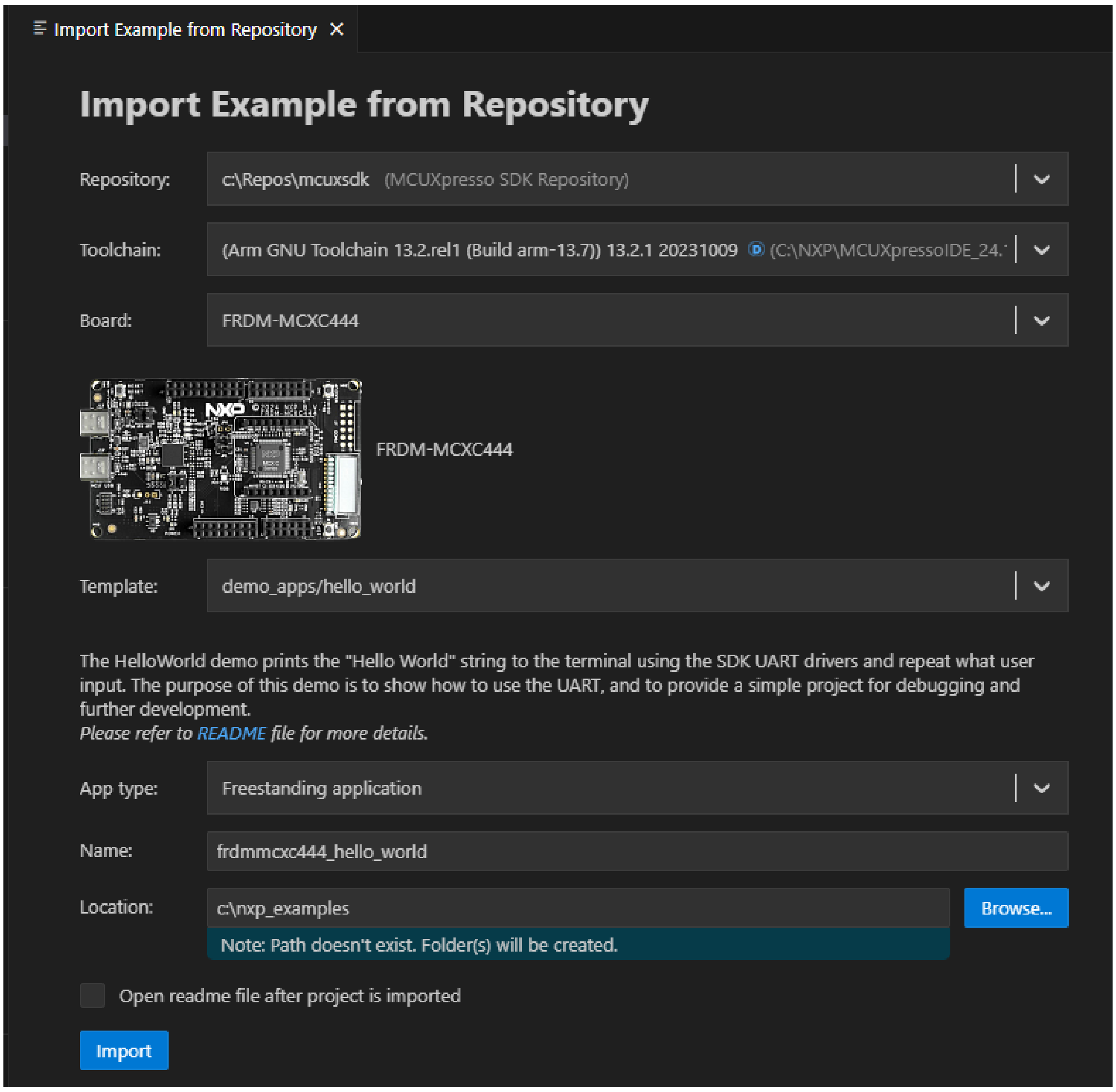
Import Example Project

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**



Import Example from Repository

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

Toolchain: (Arm GNU Toolchain 13.2.rel1 (Build arm-13.7)) 13.2.1 20231009 (C:\NXP\MCUXpressoIDE_24.

Board: FRDM-MCXC444

FRDM-MCXC444

Template: demo_apps/hello_world

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

App type: Freestanding application

Name: frdmmcxc444_hello_world

Location: c:\nxp_examples

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

Open readme file after project is imported

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

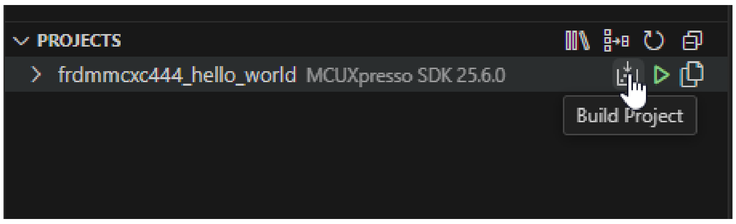
VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

Building Projects

Build Process

1. Navigate to **PROJECTS** view

2. Find your project
3. Click the **Build Project** icon

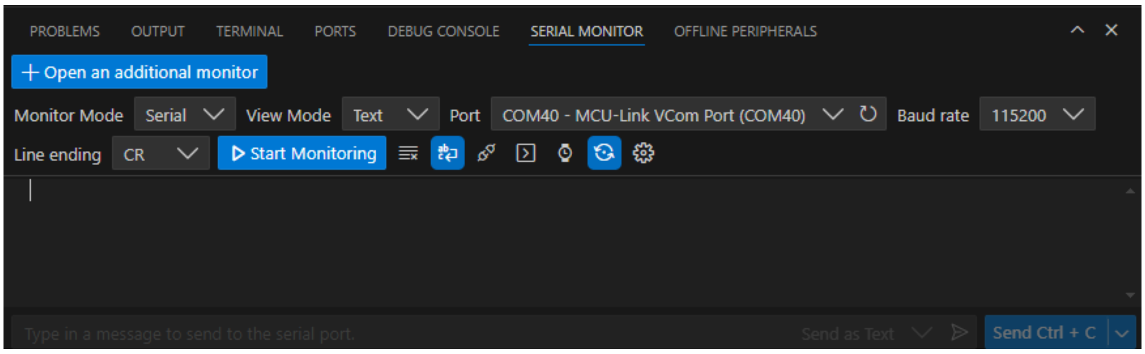


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

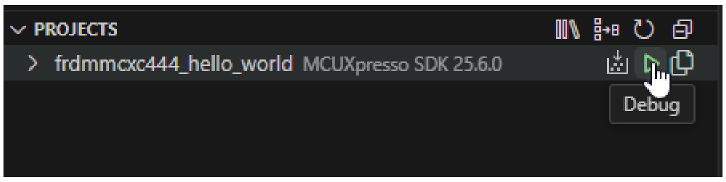
1. Open **Serial Monitor** from VS Code's integrated terminal



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

Debug Session

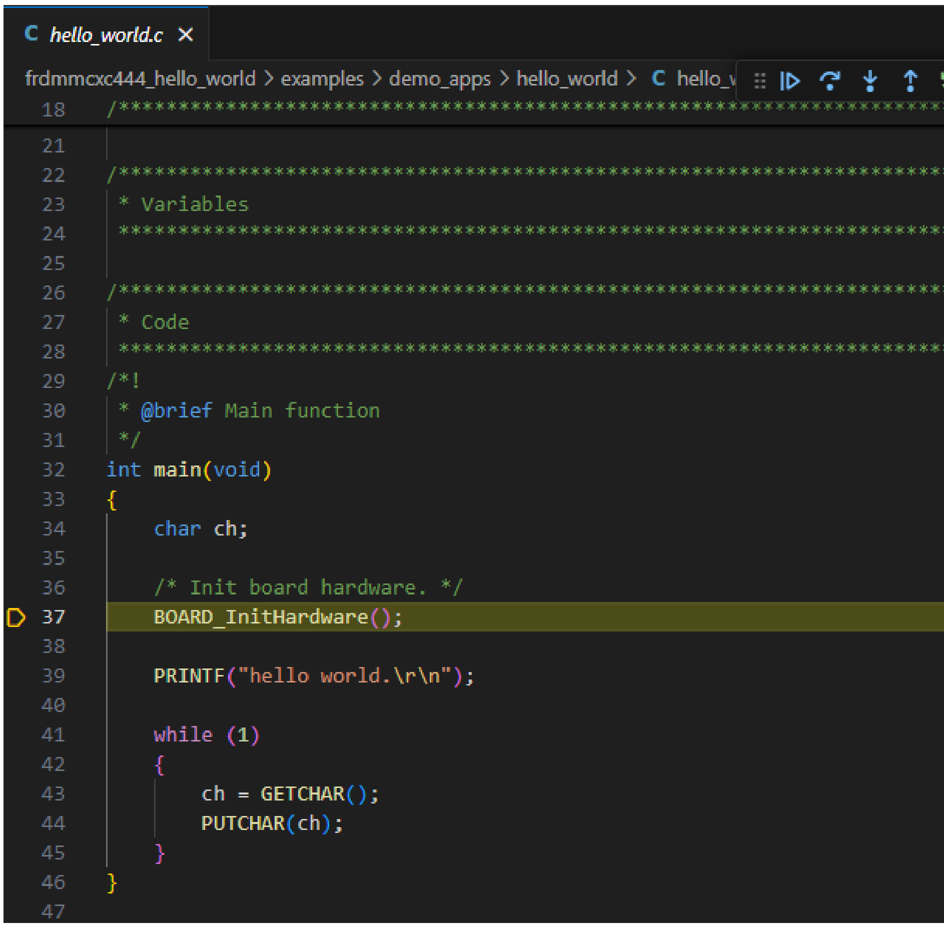
1. Navigate to **PROJECTS** view
2. Click the play button to initiate a debug session



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

Debug Controls Use the debug controls to manage execution:

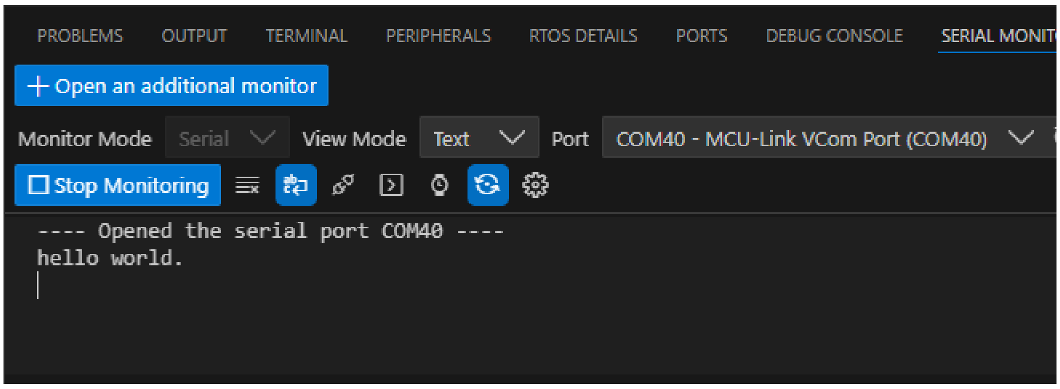
- **Continue**: Resume code execution
- **Step controls**: Navigate through code


hello_world.c
frdmmcxc444_hello_world > examples > demo_apps > hello_world > hello_

```
/************************************************************************
 
/************************************************************************
 * Variables
 ************************************************************************

/************************************************************************
 * Code
 ************************************************************************
/*!
 * @brief Main function
 */
int main(void)
{
    char ch;

    /* Init board hardware. */
    BOARD_InitHardware();

    PRINTF("hello world.\r\n");

    while (1)
    {
        ch = GETCHAR();
        PUTCHAR(ch);
    }
}
```

- **Stop**: Terminate debug session

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



Monitor Mode: Serial
View Mode: Text
Port: COM40 - MCU-Link VCom Port (COM40)

```
---- Opened the serial port COM40 ----
hello world.
```

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

Project Configuration

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

- Example applications
- Board configurations

- Middleware components
- Build system integration

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

Troubleshooting

Import Issues **SDK not detected:**

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

Project import failures:

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

Build Problems **Build failures:**

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

Debug Issues **Debug session fails:**

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

Serial monitor problems:

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

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

Advanced Features

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

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

Next Steps

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

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

Prerequisites

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

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

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

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

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

Basic Build Commands

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

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

Specifying Build Configuration

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

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

Multicore Applications For multicore devices, specify the core ID:

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

For multicore projects using sysbuild:

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

Shield Support For boards with shields:

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

Advanced Build Options

Clean Builds Force a complete rebuild:

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

Dry Run See what commands would be executed:

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

Device Variants For boards supporting multiple device variants:

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

Project Configuration

CMake Configuration Only Run configuration without building:

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

Interactive Configuration Launch the configuration GUI:

```
west build -t guiconfig
```

Flashing and Debugging

Flash Application Flash the built application to your board:

```
west flash -r linkserver
```

Debug Session Start a debugging session:

```
west debug -r linkserver
```

IDE Project Generation Generate IDE project files for traditional IDEs:

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

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

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

Troubleshooting

Build Failures

Use pristine builds to resolve dependency issues:

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

Toolchain Issues

Verify environment variables are set correctly:

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

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

Getting Help

Display help information:

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

Check Supported Configurations

If unsure about supported options for an example:

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

Best Practices

Project Organization

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

Build Efficiency

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

Development Workflow

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

Next Steps

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

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

Prerequisites

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

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

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

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

Manual Workflow Use the following steps:

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

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

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

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

Updating the SDK West project

Note: Updating project is supported with Config Tools V25.12 or newer only.

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

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

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

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

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

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

1.4 Release Notes

1.4.1 MCUXpresso SDK Release Notes

Overview

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

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

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

MCUXpresso SDK

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

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

Development tools

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

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

Supported development systems

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

Development boards	MCU devices
FRDM-KE16Z	MKE14Z32VFP4, MKE14Z32VLD4, MKE14Z32VLF4, MKE14Z64VFP4, MKE14Z64VLD4, MKE14Z64VLF4, MKE15Z32VFP4, MKE15Z32VLD4, MKE15Z32VLF4, MKE15Z64VFP4, MKE15Z64VLD4, MKE15Z64VLF4, MKE16Z32VLD4, MKE16Z32VLF4, MKE16Z64VLD4, **MKE16Z64VLF4**

MCUXpresso SDK release package

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

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

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

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

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Middleware

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

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

NXP Touch Library NXP Touch Library

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".

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.

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

[2.4.0]

- 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.

ADC12

[2.0.8]

- Bug Fixes
  - Fix build warning when FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is defined as 1.

[2.0.7]

- Improvements
  - Change the method for judging the end of calibration. Previously, it was judged based on the COCO bit of the SC1 register. After the update, it is judged based on the CAL bit of the SC3 register. In some cases, SC1[COCO] was already set to 1 before calibration was performed, which caused calibration to fail.

[2.0.6]

- Improvements
  - Removed useless comments of ADC12_DoAutoCalibration() function.

[2.0.5]

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

[2.0.4]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    * Rule 4.7 10.1 10.3 10.4 10.8 12.2 16.4 17.7

[2.0.3]

- Improvements
  - Used conversion control feature macro instead of that in IO map.

[2.0.2]

- Bug Fixes
  - Set ADC clock frequency as half of the maximum value for calibration.

[2.0.1]

- New Features
  - Added a feature to control enablement of DMA.

[2.0.0]

- Initial version.

CLOCK

[2.3.0]

- Bug Fixes
  - Removed unimplemented kSCG_AsyncDiv1Clk.

[2.2.1]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.1, rule 10.4, rule 10.8 and so on.

[2.2.0]

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

[2.1.0]

- New Features
  - Added new API CLOCK_DelayAtLeastUs() implemented by DWT to allow users set delay in unit of microsecond.

[2.0.0]

- Initial version.

COMMON

[2.6.3]

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

[2.6.2]

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

[2.6.1]

- Improvements
  - Support Cortex M23.

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.3]

- New Features
  - Added NETC into status group.

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

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

[2.3.0]

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

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Other Change
  - Added MECC status group definition.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.4]

- New Features
  - Added OTFAD into status group.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

CRC

[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.

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.

FLASH

[3.3.0]

- New Feature
  - Support for EEPROM Quick Write on devices with FTFC

[3.2.0]

- New Feature
  - Basic support for FTFC

[3.1.3]

- New Feature
  - Support 512KB flash for Kinetis E serials.

[3.1.2]

- Bug Fixes — Remove redundant comments.

[3.1.1]

- Bug Fixes — MISRA C-2012 issue fixed: rule 10.3

[3.1.0]

- New Feature
  - Support erase flash asynchronously.

[3.0.2]

- Bug Fixes — MISRA C-2012 issue fixed: rule 8.4, 17.7, 10.4, 16.1, 21.15, 11.3, 10.7 — building warning -Wnull-dereference on arm compiler v6

[3.0.1]

- New Features
  - Added support FlexNVM alias for (kw37/38/39).

[3.0.0]

- Improvements
  - Reorganized FTFx flash driver source file.
  - Extracted flash cache driver from FTFx driver.
  - Extracted flexnvm flash driver from FTFx driver.

[2.3.1]

- Bug Fixes
  - Unified Flash IFR design from K3.
  - New encoding rule for K3 flash size.

[2.3.0]

- New Features
  - Added support for device with LP flash (K3S/G).
  - Added flash prefetch speculation APIs.
- Improvements
  - Refined flash_cache_clear function.
  - Reorganized the member of flash_config_t struct.

[2.2.0]

- New Features
  - Supported FTFL device in FLASH_Swap API.
  - Supported various pflash start addresses.
  - Added support for KV58 in cache clear function.
  - Added support for device with secondary flash (KW40).
- Bug Fixes
  - Compiled execute-in-ram functions as PIC binary code for driver use.
  - Added missed flexram properties.
  - Fixed unaligned variable issue for execute-in-ram function code array.

[2.1.0]

- Improvements
  - Updated coding style to align with KSDK 2.0.
  - Different-alignment-size support for pflash and flexnvm.
  - Improved the implementation of execute-in-ram functions.

[2.0.0]

- Initial version

FTM

[2.7.4]

- Bug Fixes
  - Fixed violations of the CERT INT31-C.
  - Fixed MISRA C-2012 issue: rule 10.1.

[2.7.3]

- Bug Fixes
  - Fixed violations of the CERT INT30-C INT31-C.

[2.7.2]

- Improvements
  - Add API FTM_ERRATA_010856 for ERR010856 workaround.

[2.7.1]

- Bug Fixes
  - Added function macro when accsee FLTCTRL register FSTATE bit to prevent access nonexistent register.
  - Added function macro to prevent access nonexistent FTM channel for API FTM_ConfigSinglePWM() and FTM_ConfigCombinePWM().

[2.7.0]

- Improvements
  - Support period dithering and edge dithering feature with new APIs:
    * FTM_SetPeriodDithering()
    * FTM_SetEdgeDithering()
  - Support get channel n output and input state feature with new APIs:
    * FTM_GetChannelOutputState()
    * FTM_GetChannelInputState()
  - Support configure deadtime for specific combined channel pair with new API:
    * FTM_SetPairDeadTime()
  - Support filter clock prescale, fault output state.
  - Support new APIs to configure PWM and Modified Combine PWM:
    * FTM_ConfigSinglePWM()
    * FTM_ConfigCombinePWM()
  - Support new API to configure channel software output control:
    * FTM_SetSoftwareOutputCtrl()
    * FTM_GetSoftwareOutputValue()
    * FTM_GetSoftwareOutputEnable()
  - Support new API to update FTM counter initial value, modulo value and chanle value:
    * FTM_SetInitialModuloValue()
    * FTM_SetChannelValue()

[2.6.1]

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

[2.6.0]

- Improvements
  - Added support to half and full cycle reload feature with new APIs:
    - FTM_SetLdok()
    - FTM_SetHalfCycPeriod()
    - FTM_LoadFreq()
- Bug Fixes
  - Set the HWRSTCNT and SWRSTCNT bits to optional at initialization.

[2.5.0]

- Improvements
  - Added FTM_CalculateCounterClkDiv to help calculates the counter clock prescaler.
  - Modify FTM_UpdatePwmDutycycle API to make it return pwm duty cycles status.
- Bug Fixes
  - Fixed TPM_SetupPwm can't configure 100% center align combined PWM issues.

[2.4.1]

- Bug Fixes
  - Added function macro to determine if FTM instance has only basic features, to prevent access to protected register bits.

[2.4.0]

- Improvements
  - Added CNTIN register initialization in FTM_SetTimerPeriod API.
  - Added a new API to read the captured value of a FTM channel configured in capture mode:
    - FTM_GetInputCaptureValue()

[2.3.0]

- Improvements
  - Added support of EdgeAligned/CenterAligned/Asymmetrical combine PWM mode in FTM_SetupPWM() and FTM_SetupPwmMode() APIs.
  - Remove kFTM_ComplementaryPwm from support PWM mode, and add new parameter "enableComplementary" in structure ftm_chnl_pwm_signal_param_t.
  - Rename FTM_SetupFault() API to FTM_SetupFaultInput() to avoid ambiguity.

[2.2.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 14.4 and 17.7.

[2.2.2]

- Bug Fixes
  - Fixed the issue that when FTM instance has only TPM features cannot be initialized by FTM_Init() function. By added function macro to assert FTM is TPM only instance.

[2.2.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 10.6, 10.7 and 11.9.

[2.2.0]

- Bug Fixes
  - Fixed the issue of comparison between signed and unsigned integer expressions.
- Improvements
  - Added support of complementary mode in FTM_SetupPWM() and FTM_SetupPwmMode() APIs.
  - Added new parameter "enableDeadtime" in structure ftm_chnl_pwm_signal_param_t.

[2.1.1]

- Bug Fixes
  - Fixed COVERITY integer handing issue where the right operand of a left bit shift statement should not be a negative value. This appears in FTM_SetReloadPoints().

[2.1.0]

- Improvements
  - Added a new API FTM_SetupPwmMode() to allow the user to set the channel match value in units of timer ticks. New configure structure called ftm_chnl_pwm_config_param_t was added to configure the channel's PWM parameters. This API is similar with FTM_SetupPwm() API, but the new API will not set the timer period(MOD value), it will be useful for users to set the PWM parameters without changing the timer period.
- Bug Fixes
  - Added feature macro to enable/disable the external trigger source configuration.

[2.0.4]

- Improvements
  - Added a new API to enable DMA transfer:
    - FTM_EnableDmaTransfer()

[2.0.3]

- Bug Fixes
  - Updated the FTM driver to enable fault input after configuring polarity.

[2.0.2]

- Improvements
  - Added support to Quad Decoder feature with new APIs:
    * FTM_GetQuadDecoderFlags()
    * FTM_SetQuadDecoderModuloValue()
    * FTM_GetQuadDecoderCounterValue()
    * FTM_ClearQuadDecoderCounterValue()

[2.0.1]

- Bug Fixes
  - Updated the FTM driver to fix write to ELSA and ELSB bits.
  - FTM combine mode: set the COMBINE bit before writing to CnV register.

[2.0.0]

- Initial version.

GPIO

[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".

LPI2C

[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.

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

[2.7.4]

- Bug Fixes
  - Clear WIDTH bits from the TCR register before writing a new value in LP-SPI_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.

LPTMR

[2.2.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPUART

[2.10.0]

- New Feature
  - Added support to configure RTS watermark.

[2.9.4]

- Improvements
  - Merged duplicate code.

[2.9.3]

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

[2.9.2]

- Bug Fixes
  - Fixed coverity issues.

[2.9.1]

- Bug Fixes
  - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.7]

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

[2.7.6]

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

[2.7.5]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Bug Fixes
  - Added missing interrupt enable masks kLPUART_Match1InterruptEnable and kLPUART_Match2InterruptEnable.
  - Fixed bug in LPUART_EnableInterrupts, LPUART_DisableInterrupts and LPUART_GetEnabledInterrupts that the BAUD[LBKDIE] bit field should be soc specific.
  - Fixed bug in LPUART_TransferHandleIRQ that idle line interrupt should be disabled when rx data size is zero.

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

[2.4.1]

- Bug Fixes
  - Fixed MISRA advisory 17.7 issues.

[2.4.0]

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

[2.3.1]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.3.0]

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

[2.2.8]

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

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - Update transactional APIs.

MCM

[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.

MMDVSQ

[2.0.4]

- Improvements
  - Fixed CERT-C issues.

[2.0.3]

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

[2.0.2]

- Bug fix:
  - Fixed MMDVSQ_GetExecutionStatus function get execution status wrong.

[2.0.1]

- Other changes:
  - Changed name of MMDVSQ_GetDivideRemainder and MMDVSQ_GetDivideQuotient functions.

[2.0.0]

- Initial version.

MSCAN

[2.1.0]

- Improvements
  - Added wrong data length code(DLC) detection, report kStatus_MSCAN_DataLengthError when DLC is wrong.

[2.0.7]

- Bug Fixes
  - Fix build warning.

[2.0.6]

- Bug Fixes
  - Fix MISRA C-2012 rule 10.3.

[2.0.5]

- Bug Fixes
  - Fixed the code error issue and simplified the algorithm in improved timing APIs.
    - MSCAN_CalculateImprovedTimingValues

[2.0.4]

- Bug Fixes
  - Fixed the non-divisible case in improved timing API.
    - MSCAN_CalculateImprovedTimingValues

[2.0.3]

- Bug Fixes
  - MISRA C-2012 issue check.
    - Fixed MISRA C-2012 issue in mcan driver, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-11.8, rule-14.4, rule-15.5, rule-16.1, rule-16.4, rule-17.7, rule-8.4, rule-8.5.

[2.0.2]

- Bug Fixes
  - Fixed issue Clang - disable declaration alignment.
  - Fixed issue Central repository code formatting.
- Improvements
  - Implemented feature to find classical CAN improved timing configuration.

[2.0.1]

- Bug Fixes
  - Fixed timestamp issue where it cannot be enabled.
  - Fixed standard mode frame buffer configuration.
  - Added RX Message Buffer Mask helper macro, MSCAN_RX_MB_STD_MASK, MSCAN_RX_MB_EXT_MASK.
  - Fixed dataByte[0-7] order in struct type mscan_frame_t.
  - Updated function MSCAN_WriteTxMb. The MSCAN registers are 8 bits, using 8 bits write for registers.

[2.0.0]

- Initial version.

PDB

[2.0.4]

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

[2.0.3]

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

[2.0.2]

- Improvement:
  - Used macros in feature file instead of that in IO map.

[2.0.1]

- Changed PDB register base array to const.

[2.0.0]

- Initial version.

PMC

[2.0.3]

- Bug Fixes
  - Fixed the violation of MISRA C-2012 rule 11.3.

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed MISRA issues.
    - Rule 10.8, Rule 10.3.

[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.

PWT

[2.0.2]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.8, 10.3, 10.6.

[2.0.0]

- Initial version.

RCM

[2.0.4]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rule 10.3

[2.0.3]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rules.

[2.0.2]

- Bug Fixes
  - Fixed MISRA issue.
    * Rule 10.8, rule 10.1, rule 13.2, rule 3.1.

[2.0.1]

- Bug Fixes
  - Fixed kRCM_SourceSw bit shift issue.

[2.0.0]

- Initial version.

RTC

[2.4.0]

- New features
  - Add support for RTC clock output.
  - Add support for RTC time seconds interrupt configuration.

[2.3.3]

- Bug Fixes
  - Fix RTC_GetDatetime function validating datetime issue.

[2.3.2]

- Improvements
  - Handle errata 010716: Disable the counter before setting alarm register and then reenable the counter.

[2.3.1]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.3.0]

- Improvements
  - Added API RTC_EnableLPOClock to set 1kHz LPO clock.
  - Added API RTC_EnableCrystalClock to replace API RTC_SetClockSource.

[2.2.2]

- Improvements
  - Refine _rtc_interrupt_enable order.

[2.2.1]

- Bug Fixes
  - Fixed the issue of Pa082 warning.
  - Fixed the issue of bit field mask checking.
  - Fixed the issue of hard code in RTC_Init.

[2.2.0]

- Bug Fixes
  - Fixed MISRA C-2012 issue.
    - Fixed rule contain: rule-17.7, rule-14.4, rule-10.4, rule-10.7, rule-10.1, rule-10.3.
  - Fixed central repository code formatting issue.
- Improvements
  - Added an API for enabling wakeup pin.

[2.1.0]

- Improvements
  - Added feature macro check for many features.

[2.0.0]

- Initial version.

SIM

[2.2.0]

- Improvements
  - Added API to trigger TRGMUX.

[2.1.3]

- Improvements
  - Updated function SIM_GetUniqueId to support different register names.

[2.1.2]

- Bug Fixes
  - Fixed SIM_GetUniqueId bug that could not get UIDH.

[2.1.1]

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

[2.1.0]

- Improvements
  - Added new APIs: SIM_GetRfAddr() and SIM_EnableSystickClock().

[2.0.0]

- Initial version.

SMC

[2.0.7]

- Bug Fixes
  - Fixed MISRA-2012 issue 10.3.

[2.0.6]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    * Fixed rule 10.3, rule 11.3.

[2.0.5]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    * Fixed rule 15.7, rule 14.4, rule 10.3, rule 10.1, rule 10.4.

[2.0.4]

- Bug Fixes
  - When entering stop modes, used RAM function for the flash synchronization issue. Application should make sure that, the RW data of fsl_smc.c is located in memory region which is not powered off in stop modes.

[2.0.3]

- Improvements
  - Added APIs SMC_PreEnterStopModes, SMC_PreEnterWaitModes, SMC_PostExitWaitModes, and SMC_PostExitStopModes.

[2.0.2]

- Bug Fixes
  - Added DSB before WFI while ISB after WFI.
- Other Changes
  - Updated SMC_SetPowerModeVlpw implementation.

[2.0.1]

- Other Changes
  - Updated for KL8x.

[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.

TSI_V5

[2.6.1]

- Improvements
  - The SHIELD register write was fixed to be rewritten.

[2.6.0]

- Improvements
  - Add API TSI_ShieldChannelConfig to config all TSI shield channels.

[2.5.0]

- Improvements
  - Add API TSI_EnableShieldChannels to enable/disable TSI shield channels.

[2.4.0]

- Bug Fixes
  - Fixed some elements in the array tsi_sensitivity_xdn_option_t does not match the S_XDN bits.

[2.3.0]

- Other Changes
  - Changed the TSI SINC cutoff divider number.

[2.2.0]

- Improvements
  - Extended enableShield items from tsi_selfCap_config_t structure to cover three shields in the ke17z series.
  - Added interface for getting instance from TSI base address and apply it for clock and IRQ enable/disable.

[2.1.2]

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

[2.1.1]

- Improvements
  - Improved the module's noise immunity in mutual cap mode by setting M_TRIM2[0] to 1 in TSI_MUL1 register.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.1, 10.3, 10.4, 10.8, 12.2, 14.4, 17.7.

[2.1.0]

- Bug Fixes
  - Fixed incorrect TSI SSC clock calculation.

[2.0.1]

- Improvements
  - Added functions for M_TX_USED bitfield for ke16z only (Unused TX mutual pins can work as GPIO).

[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.

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.

MKE16Z4

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 FreeMASTER

freemaster

1.7.2 FreeRTOS

FreeRTOS

Chapter 2

MKE16Z4

2.1 ACMP: Analog Comparator Driver

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

Initializes the ACMP.

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

Parameters

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

void ACMP_Deinit(CMP_Type *base)

Deinitializes the ACMP.

Parameters

- base – ACMP peripheral base address.

void ACMP_GetDefaultConfig(*acmp_config_t* *config)

Gets the default configuration for ACMP.

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

Example:

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

Parameters

- config – Pointer to ACMP configuration structure.

void ACMP_Enable(CMP_Type *base, bool enable)

Enables or disables the ACMP.

Parameters

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

void ACMP_EnableLinkToDAC(CMP_Type *base, bool enable)

Enables the link from CMP to DAC enable.

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

Parameters

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

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

Sets the channel configuration.

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

Example:

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

Parameters

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

void ACMP_EnableDMA(CMP_Type *base, bool enable)

Enables or disables DMA.

Parameters

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

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

Configures the filter.

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

Example:

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

Parameters

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

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

Configures the internal DAC.

Example:

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

Parameters

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

void ACMP_EnableInterrupts(CMP_Type *base, uint32_t mask)

Enables interrupts.

Parameters

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

void ACMP_DisableInterrupts(CMP_Type *base, uint32_t mask)

Disables interrupts.

Parameters

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

uint32_t ACMP_GetStatusFlags(CMP_Type *base)

Gets status flags.

Parameters

- base – ACMP peripheral base address.

Returns

Status flags asserted mask. See "_acmp_status_flags".

void ACMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)

Clears status flags.

Parameters

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

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

Configure the discrete mode.

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

Parameters

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

void ACMP_GetDefaultDiscreteModeConfig(*acmp_discrete_mode_config_t* *config)

Get the default configuration for discrete mode setting.

Parameters

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

FSL_ACMP_DRIVER_VERSION

ACMP driver version 2.4.0.

enum _acmp_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kACMP_OutputRisingInterruptEnable

Enable the interrupt when comparator outputs rising.

enumerator kACMP_OutputFallingInterruptEnable

Enable the interrupt when comparator outputs falling.

enum _acmp_status_flags

Status flag mask.

Values:

enumerator kACMP_OutputRisingEventFlag

Rising-edge on compare output has occurred.

enumerator kACMP_OutputFallingEventFlag

Falling-edge on compare output has occurred.

enumerator kACMP_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum _acmp_offset_mode

Comparator hard block offset control.

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

Values:

enumerator kACMP_OffsetLevel0

The comparator hard block output has level 0 offset internally.

enumerator kACMP_OffsetLevel1

The comparator hard block output has level 1 offset internally.

enum _acmp_hysteresis_mode

Comparator hard block hysteresis control.

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

Values:

enumerator kACMP_HysteresisLevel0

Offset is level 0 and Hysteresis is level 0.

enumerator kACMP_HysteresisLevel1

Offset is level 0 and Hysteresis is level 1.

enumerator kACMP_HysteresisLevel2

Offset is level 0 and Hysteresis is level 2.

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

enum _acmp_reference_voltage_source
: CMP Voltage Reference source.

Values:

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

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

enum _acmp_port_input
: Port input source.

Values:

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

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

enum _acmp_dac_work_mode
: Internal DAC's work mode.

Values:

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

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

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

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

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

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

typedef enum *_acmp_reference_voltage_source* acmp_reference_voltage_source_t
: CMP Voltage Reference source.

typedef enum *_acmp_port_input* acmp_port_input_t
: Port input source.

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

typedef struct *_acmp_config* acmp_config_t
: Configuration for ACMP.

typedef struct *_acmp_channel_config* acmp_channel_config_t
    Configuration for channel.

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

typedef struct *_acmp_filter_config* acmp_filter_config_t
    Configuration for filter.

typedef struct *_acmp_dac_config* acmp_dac_config_t
    Configuration for DAC.

typedef struct *_acmp_discrete_mode_config* acmp_discrete_mode_config_t
    Configuration for discrete mode.

CMP_C0_CFx_MASK
    The mask of status flags cleared by writing 1.

struct _acmp_config
    #include <fsl_acmp.h> Configuration for ACMP.

Public Members

acmp_offset_mode_t offsetMode
    Offset mode.

acmp_hysteresis_mode_t hysteresisMode
    Hysteresis mode.

bool enableHighSpeed
    Enable High Speed (HS) comparison mode.

bool enableInvertOutput
    Enable inverted comparator output.

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

bool enablePinOut
    The comparator output is available on the associated pin.

struct _acmp_channel_config
    #include <fsl_acmp.h> Configuration for channel.

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

Public Members

acmp_port_input_t positivePortInput
    Input source of the comparator's positive port.

uint32_t plusMuxInput
    Plus mux input channel(0~7).

acmp_port_input_t negativePortInput
    Input source of the comparator's negative port.

uint32_t minusMuxInput
    Minus mux input channel(0~7).

struct _acmp_filter_config
#include <fsl_acmp.h> Configuration for filter.

Public Members

uint32_t filterCount
Filter Sample Count. Available range is 1-7, 0 would cause the filter disabled.

uint32_t filterPeriod
Filter Sample Period. The divider to bus clock. Available range is 0-255.

struct _acmp_dac_config
#include <fsl_acmp.h> Configuration for DAC.

Public Members

acmp_reference_voltage_source_t referenceVoltageSource
Supply voltage reference source.

uint32_t DACValue
Value for DAC Output Voltage. Available range is 0-255.

bool enableOutput
Enable the DAC output.

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

Public Members

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

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

2.2 ADC12: Analog-to-Digital Converter

void ADC12_Init(ADC_Type *base, const *adc12_config_t* *config)
Initialize the ADC12 module.

Parameters

- base – ADC12 peripheral base address.
- config – Pointer to "adc12_config_t" structure.

void ADC12_Deinit(ADC_Type *base)
De-initialize the ADC12 module.

Parameters

- base – ADC12 peripheral base address.

void ADC12_GetDefaultConfig(*adc12_config_t* *config)

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

This function initializes the converter configuration structure with an available settings. The default values are:

Example:

```
config->referenceVoltageSource = kADC12_ReferenceVoltageSourceVref;
config->clockSource = kADC12_ClockSourceAlt0;
config->clockDivider = kADC12_ClockDivider1;
config->resolution = kADC12_Resolution8Bit;
config->sampleClockCount = 12U;
config->enableContinuousConversion = false;
```

Parameters

- config – Pointer to "adc12_config_t" structure.

void ADC12_SetChannelConfig(ADC_Type *base, uint32_t channelGroup, const *adc12_channel_config_t* *config)

Configure the conversion channel.

This operation triggers the conversion in software trigger mode. In hardware trigger mode, this API configures the channel while the external trigger source helps to trigger the conversion.

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC can have more than one group of status and control register, one for each conversion. The channel group parameter indicates which group of registers are used, channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any time, only one of the channel groups is actively controlling ADC conversions. Channel group 0 is used for both software and hardware trigger modes of operation. Channel groups 1 and greater indicate potentially multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the MCU reference manual about the number of SC1n registers (channel groups) specific to this device. None of the channel groups 1 or greater are used for software trigger operation and therefore writes to these channel groups do not initiate a new conversion. Updating channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

Parameters

- base – ADC12 peripheral base address.
- channelGroup – Channel group index.
- config – Pointer to "adc12_channel_config_t" structure.

static inline uint32_t ADC12_GetChannelConversionValue(ADC_Type *base, uint32_t channelGroup)

Get the conversion value.

Parameters

- base – ADC12 peripheral base address.
- channelGroup – Channel group index.

Returns

Conversion value.

uint32_t ADC12_GetChannelStatusFlags(ADC_Type *base, uint32_t channelGroup)

Get the status flags of channel.

Parameters

- base – ADC12 peripheral base address.
- channelGroup – Channel group index.

Returns

Flags' mask if indicated flags are asserted. See to "_adc12_channel_status_flags".

status_t ADC12_DoAutoCalibration(ADC_Type *base)

Automate the hardware calibration.

This auto calibration helps to adjust the gain automatically according to the converter's working environment. Execute the calibration before conversion. Note that the software trigger should be used during calibration.

Parameters

- base – ADC12 peripheral base address.

Return values

- kStatus_Success – Calibration is done successfully.
- kStatus_Fail – Calibration is failed.

static inline void ADC12_SetOffsetValue(ADC_Type *base, uint32_t value)

Set the offset value for the conversion result.

This offset value takes effect on the conversion result. If the offset value is not zero, the conversion result is substracted by it.

Parameters

- base – ADC12 peripheral base address.
- value – Offset value.

static inline void ADC12_SetGainValue(ADC_Type *base, uint32_t value)

Set the gain value for the conversion result.

This gain value takes effect on the conversion result. If the gain value is not zero, the conversion result is amplified as it.

Parameters

- base – ADC12 peripheral base address.
- value – Gain value.

static inline void ADC12_EnableDMA(ADC_Type *base, bool enable)

Enable generating the DMA trigger when conversion is completed.

Parameters

- base – ADC12 peripheral base address.
- enable – Switcher of DMA feature. "true" means to enable, "false" means to disable.

static inline void ADC12_EnableHardwareTrigger(ADC_Type *base, bool enable)

Enable of disable the hardware trigger mode.

Parameters

- base – ADC12 peripheral base address.

- enable – Switcher of hardware trigger feature. "true" means to enable, "false" means not.

void ADC12_SetHardwareCompareConfig(ADC_Type *base, const *adc12_hardware_compare_config_t* *config)

Configure the hardware compare mode.

The hardware compare mode provides a way to process the conversion result automatically by hardware. Only the result in compare range is available. To compare the range, see "adc12_hardware_compare_mode_t", or the reference manual document for more detailed information.

Parameters

- base – ADC12 peripheral base address.
- config – Pointer to "adc12_hardware_compare_config_t" structure. Pass "NULL" to disable the feature.

void ADC12_SetHardwareAverage(ADC_Type *base, *adc12_hardware_average_mode_t* mode)

Set the hardware average mode.

Hardware average mode provides a way to process the conversion result automatically by hardware. The multiple conversion results are accumulated and averaged internally. This aids to get more accurate conversion result.

Parameters

- base – ADC12 peripheral base address.
- mode – Setting hardware average mode. See to "adc12_hardware_average_mode_t".

uint32_t ADC12_GetStatusFlags(ADC_Type *base)

Get the status flags of the converter.

Parameters

- base – ADC12 peripheral base address.

Returns

Flags' mask if indicated flags are asserted. See to "_adc12_status_flags".

enum _adc12_channel_status_flags

Channel status flags' mask.

Values:

enumerator kADC12_ChannelConversionCompletedFlag

Conversion done.

enum _adc12_status_flags

Converter status flags' mask.

Values:

enumerator kADC12_ActiveFlag

Converter is active.

enumerator kADC12_CalibrationFailedFlag

Calibration is failed.

enum _adc12_clock_divider

Clock divider for the converter.

Values:

enumerator kADC12_ClockDivider1
For divider 1 from the input clock to the module.

enumerator kADC12_ClockDivider2
For divider 2 from the input clock to the module.

enumerator kADC12_ClockDivider4
For divider 4 from the input clock to the module.

enumerator kADC12_ClockDivider8
For divider 8 from the input clock to the module.

enum _adc12_resolution
Converter's resolution.

Values:

enumerator kADC12_Resolution8Bit
8 bit resolution.

enumerator kADC12_Resolution12Bit
12 bit resolution.

enumerator kADC12_Resolution10Bit
10 bit resolution.

enum _adc12_clock_source
Conversion clock source.

Values:

enumerator kADC12_ClockSourceAlt0
Alternate clock 1 (ADC_ALTCLK1).

enumerator kADC12_ClockSourceAlt1
Alternate clock 2 (ADC_ALTCLK2).

enumerator kADC12_ClockSourceAlt2
Alternate clock 3 (ADC_ALTCLK3).

enumerator kADC12_ClockSourceAlt3
Alternate clock 4 (ADC_ALTCLK4).

enum _adc12_reference_voltage_source
Reference voltage source.

Values:

enumerator kADC12_ReferenceVoltageSourceVref
For external pins pair of VrefH and VrefL.

enumerator kADC12_ReferenceVoltageSourceValt
For alternate reference pair of ValtH and ValtL.

enum _adc12_hardware_average_mode
Hardware average mode.

Values:

enumerator kADC12_HardwareAverageCount4
For hardware average with 4 samples.

enumerator kADC12_HardwareAverageCount8
For hardware average with 8 samples.

enumerator kADC12_HardwareAverageCount16
For hardware average with 16 samples.

enumerator kADC12_HardwareAverageCount32
For hardware average with 32 samples.

enumerator kADC12_HardwareAverageDisabled
Disable the hardware average feature.

enum _adc12_hardware_compare_mode
Hardware compare mode.

Values:

enumerator kADC12_HardwareCompareMode0
x < value1.

enumerator kADC12_HardwareCompareMode1
x > value1.

enumerator kADC12_HardwareCompareMode2
if value1 <= value2, then x < value1 || x > value2; else, value1 > x > value2.

enumerator kADC12_HardwareCompareMode3
if value1 <= value2, then value1 <= x <= value2; else x >= value1 || x <= value2.

typedef enum *_adc12_clock_divider* adc12_clock_divider_t
Clock divider for the converter.

typedef enum *_adc12_resolution* adc12_resolution_t
Converter's resolution.

typedef enum *_adc12_clock_source* adc12_clock_source_t
Conversion clock source.

typedef enum *_adc12_reference_voltage_source* adc12_reference_voltage_source_t
Reference voltage source.

typedef enum *_adc12_hardware_average_mode* adc12_hardware_average_mode_t
Hardware average mode.

typedef enum *_adc12_hardware_compare_mode* adc12_hardware_compare_mode_t
Hardware compare mode.

typedef struct *_adc12_config* adc12_config_t
Converter configuration.

typedef struct *_adc12_hardware_compare_config* adc12_hardware_compare_config_t
Hardware compare configuration.

typedef struct *_adc12_channel_config* adc12_channel_config_t
Channel conversion configuration.

FSL_ADC12_DRIVER_VERSION
ADC12 driver version.

struct _adc12_config
#include <fsl_adc12.h> Converter configuration.

Public Members

adc12_reference_voltage_source_t referenceVoltageSource
    Select the reference voltage source.

adc12_clock_source_t clockSource
    Select the input clock source to converter.

adc12_clock_divider_t clockDivider
    Select the divider of input clock source.

adc12_resolution_t resolution
    Select the sample resolution mode.

uint32_t sampleClockCount
    Select the sample clock count. Add its value may improve the stability of the conversion result.

bool enableContinuousConversion
    Enable continuous conversion mode.

struct _adc12_hardware_compare_config
    #include <fsl_adc12.h> Hardware compare configuration.

Public Members

adc12_hardware_compare_mode_t hardwareCompareMode
    Select the hardware compare mode.

int16_t value1
    Setting value1 for hardware compare mode.

int16_t value2
    Setting value2 for hardware compare mode.

struct _adc12_channel_config
    #include <fsl_adc12.h> Channel conversion configuration.

Public Members

uint32_t channelNumber
    Setting the conversion channel number. The available range is 0-31. See channel connection information for each chip in Reference Manual document.

bool enableInterruptOnConversionCompleted
    Generate a interrupt request once the conversion is completed.

2.3 Clock Driver

enum _clock_name
    Clock name used to get clock frequency.

    Values:

    enumerator kCLOCK_CoreSysClk
        Core/system clock

enumerator kCLOCK_BusClk
: Bus clock

enumerator kCLOCK_FlashClk
: Flash clock

enumerator kCLOCK_ScgSysOscClk
: SCG system OSC clock. (SYSOSC)

enumerator kCLOCK_ScgSircClk
: SCG SIRC clock.

enumerator kCLOCK_ScgFircClk
: SCG FIRC clock.

enumerator kCLOCK_ScgLpFllClk
: SCG low power FLL clock. (LPFLL)

enumerator kCLOCK_ScgSysOscAsyncDiv2Clk
: SOSCDIV2_CLK.

enumerator kCLOCK_ScgSircAsyncDiv2Clk
: SIRCDIV2_CLK.

enumerator kCLOCK_ScgFircAsyncDiv2Clk
: FIRCDIV2_CLK.

enumerator kCLOCK_ScgLpFllAsyncDiv2Clk
: LPFLLDIV2_CLK.

enumerator kCLOCK_LpoClk
: LPO clock

enumerator kCLOCK_ErClk
: ERCLK. The external reference clock from SCG.

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

Values:

enumerator kCLOCK_IpSrcNoneOrExt
: Clock is off or external clock is used.

enumerator kCLOCK_IpSrcSysOscAsync
: System Oscillator async clock.

enumerator kCLOCK_IpSrcSircAsync
: Slow IRC async clock.

enumerator kCLOCK_IpSrcFircAsync
: Fast IRC async clock.

enumerator kCLOCK_IpSrcLpFllAsync
: LPFLL async 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_Flash0

enumerator kCLOCK_Mscan0

enumerator kCLOCK_Lpspi0

enumerator kCLOCK_Crc0

enumerator kCLOCK_Pdb0

enumerator kCLOCK_Lpit0

enumerator kCLOCK_Ftm0

enumerator kCLOCK_Ftm1

enumerator kCLOCK_Adc0

enumerator kCLOCK_Rtc0

enumerator kCLOCK_Lptmr0

enumerator kCLOCK_Tsi0

enumerator kCLOCK_PortA

enumerator kCLOCK_PortB

enumerator kCLOCK_PortC

enumerator kCLOCK_PortD

enumerator kCLOCK_PortE

enumerator kCLOCK_Pwt0

enumerator kCLOCK_Ewm0

enumerator kCLOCK_Lpi2c0

enumerator kCLOCK_Lpuart0

enumerator kCLOCK_Lpuart1

enumerator kCLOCK_Lpuart2

enumerator kCLOCK_Cmp0

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_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_SysClkSrcLpFll
Low power FLL.

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 kClockClkoutSelScgSlow
SCG slow clock.

enumerator kClockClkoutSelSysOsc
System OSC.

enumerator kClockClkoutSelSirc
Slow IRC.

enumerator kClockClkoutSelFirc
Fast IRC.

enumerator kClockClkoutSelLpFll
Low power FLL.

enum _scg_async_clk
SCG asynchronous clock type.

Values:

enumerator kSCG_AsyncDiv2Clk
The async clock by DIV2, e.g. SOSCDIV2_CLK, SIRCDIV2_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.

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.

enumerator kSCG_SysOscEnableErClk
Enable OSCERCLK.

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 7 ULP).

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

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_FircTrimSrcSysOsc
System OSC.

enum _scg_firc_range
SCG fast IRC clock frequency range.

Values:

enumerator kSCG_FircRange48M
Fast IRC is trimmed to 48 MHz.

FIRC enable mode.

Values:

enumerator kSCG_FircEnable
Enable FIRC clock.

enumerator kSCG_FircEnableInStop
Enable FIRC in stop mode.

enumerator kSCG_FircEnableInLowPower
Enable FIRC in low power mode.

enumerator kSCG_FircDisableRegulator
Disable regulator.

LPFLL enable mode.

Values:

enumerator kSCG_LpFllEnable
Enable LPFLL clock.

enum _scg_lpfll_range
SCG LPFLL clock frequency range.

Values:

enumerator kSCG_LpFllRange48M
LPFLL is trimmed to 48MHz.

enum _scg_lpfll_trim_mode
SCG LPFLL trim mode.

Values:

enumerator kSCG_LpFllTrimNonUpdate
LPFLL trim is enabled but the trim value update is not enabled. In this mode, the trim value is fixed to the initialized value, which is defined by the Member variable trimValue in the structure scg_lpfll_trim_config_t.

enumerator kSCG_LpFllTrimUpdate
FIRC trim is enabled and trim value update is enabled. In this mode, the trim value is automatically updated.

enum _scg_lpfll_trim_src
SCG LPFLL trim source.

Values:

enumerator kSCG_LpFllTrimSrcSirc
SIRC.

enumerator kSCG_LpFllTrimSrcFirc
FIRC.

enumerator kSCG_LpFllTrimSrcSysOsc
System OSC.

enumerator kSCG_LpFllTrimSrcRtcOsc
RTC OSC (32.768 kHz).

enum _scg_lpfll_lock_mode
SCG LPFLL lock mode.

Values:

enumerator kSCG_LpFllLock1Lsb
Lock with 1 LSB.

enumerator kSCG_LpFllLock2Lsb
Lock with 2 LSB.

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_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_lpfll_range* scg_lpfll_range_t
: SCG LPFLL clock frequency range.

typedef enum *_scg_lpfll_trim_mode* scg_lpfll_trim_mode_t
: SCG LPFLL trim mode.

typedef enum *_scg_lpfll_trim_src* scg_lpfll_trim_src_t
: SCG LPFLL trim source.

typedef enum *_scg_lpfll_lock_mode* scg_lpfll_lock_mode_t
: SCG LPFLL lock mode.

typedef struct *_scg_lpfll_trim_config* scg_lpfll_trim_config_t
: SCG LPFLL clock trim configuration.

typedef struct *_scg_lpfll_config* scg_lpfll_config_t
: SCG low power FLL 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.

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 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.

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_GetBusClkFreq(void)

Get the bus clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetFlashClkFreq(void)

Get the flash clock frequency.

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_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 2.3.0.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

RTC_CLOCKS

Clock ip name array for RTC.

PORT_CLOCKS

Clock ip name array for PORT.

LPI2C_CLOCKS

Clock ip name array for LPI2C.

TSI_CLOCKS
Clock ip name array for TSI.

LPUART_CLOCKS
Clock ip name array for LPUART.

LPTMR_CLOCKS
Clock ip name array for LPTMR.

ADC12_CLOCKS
Clock ip name array for ADC12.

LPSPI_CLOCKS
Clock ip name array for LPSPI.

LPIT_CLOCKS
Clock ip name array for LPIT.

CRC_CLOCKS
Clock ip name array for CRC.

CMP_CLOCKS
Clock ip name array for CMP.

FLASH_CLOCKS
Clock ip name array for FLASH.

EWM_CLOCKS
Clock ip name array for EWM.

FTM_CLOCKS
Clock ip name array for FLEXTMR.

PDB_CLOCKS
Clock ip name array for PDB.

PWT_CLOCKS
Clock ip name array for PWT.

MSCAN_CLOCKS
Clock ip name array for MSCAN.

LPO_CLK_FREQ
LPO clock frequency.

kCLOCK_Osc0ErClk

CLOCK_GetOsc0ErClkFreq
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_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.

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.

status_t CLOCK_InitLpFll(const *scg_lpfll_config_t* *config)

Initializes the SCG LPFLL clock.

This function enables the SCG LPFLL clock according to the configuration.

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

Parameters

- config – Pointer to the configuration structure.

Return values

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

status_t CLOCK_DeinitLpFll(void)

De-initializes the SCG LPFLL.

This function disables the SCG LPFLL.

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

Return values

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

static inline void CLOCK_SetLpFllAsyncClkDiv(*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_GetLpFllFreq(void)

Gets the SCG LPFLL clock frequency.

Returns

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

uint32_t CLOCK_GetLpFllAsyncFreq(*scg_async_clk_t* type)

Gets the SCG asynchronous clock frequency from the LPFLL.

Parameters

- type – The asynchronous clock type.

Returns

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

static inline bool CLOCK_IsLpFllValid(void)

Checks whether the LPFLL 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.

uint32_t divSlow

Slow clock divider, see scg_sys_clk_div_t.

uint32_t __pad0__

Reserved.

uint32_t __pad1__

Reserved.

uint32_t __pad2__

Reserved.

uint32_t divCore

Core clock divider, see scg_sys_clk_div_t.

uint32_t __pad3__

Reserved.

uint32_t src

System clock source, see scg_sys_clk_src_t.

uint32_t ___pad4___
: 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 div2
: SOSCDIV2 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 div2
: SIRCDIV2 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 div2
: FIRCDIV2 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.

scg_lpfll_trim_mode_t trimMode
: Trim mode.

scg_lpfll_lock_mode_t lockMode
: Lock mode; Irrelevant if the trimMode is kSCG_LpFllTrimNonUpdate.

scg_lpfll_trim_src_t trimSrc
: Trim source.

uint8_t trimDiv

Trim predivideds value, which can be 0 ~ 31. [Trim source frequency / (trimDiv + 1)] must be 2 MHz or 32768 Hz.

uint8_t trimValue

Trim value; Irrelevant if trimMode is the kSCG_LpFllTrimUpdate.

uint8_t enableMode

Enable mode, OR'ed value of _scg_lpfll_enable_mode

scg_async_clk_div_t div2

LPFLLDIV2 value.

scg_lpfll_range_t range

LPFLL frequency range.

const *scg_lpfll_trim_config_t* *trimConfig

Trim configuration; set NULL to disable trim.

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_lpfll_trim_config

#include <fsl_clock.h> SCG LPFLL clock trim configuration.

struct _scg_lpfll_config

#include <fsl_clock.h> SCG low power FLL configuration.

2.4 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.0.5.

Current version: 2.0.5

Change log:

- 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.

static inline void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

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

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

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

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

Parameters

- config – CRC protocol configuration structure.

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

Writes data to the CRC module.

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

Parameters

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

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

Returns

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

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

Returns

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

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

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

struct _crc_config

#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

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

uint32_t seed

Starting checksum value

bool reflectIn

Reflect bits on input.

bool reflectOut

Reflect bits on output.

bool complementChecksum

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

crc_bits_t crcBits

Selects 16- or 32- bit CRC protocol.

crc_result_t crcResult

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

2.5 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

2.6 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

2.7 C90TFS Flash Driver

2.8 ftfx adapter

2.9 Ftftx CACHE Driver

enum _ftfx_cache_ram_func_constants

Constants for execute-in-RAM flash function.

Values:

enumerator kFTFx_CACHE_RamFuncMaxSizeInWords

The maximum size of execute-in-RAM function.

typedef struct *_flash_prefetch_speculation_status* ftfx_prefetch_speculation_status_t

FTFx prefetch speculation status.

typedef struct *_ftfx_cache_config* ftfx_cache_config_t

FTFx cache driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

status_t FTFx_CACHE_Init(*ftfx_cache_config_t* *config)

Initializes the global FTFx cache structure members.

This function checks and initializes the Flash module for the other FTFx cache APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.

- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

status_t FTFx_CACHE_ClearCachePrefetchSpeculation(*ftfx_cache_config_t* *config, bool isPreProcess)

Process the cache/prefetch/speculation to the flash.

Parameters

- config – A pointer to the storage for the driver runtime state.
- isPreProcess – The possible option used to control flash cache/prefetch/speculation

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

status_t FTFx_CACHE_PflashSetPrefetchSpeculation(*ftfx_prefetch_speculation_status_t* *speculationStatus)

Sets the PFlash prefetch speculation to the intended speculation status.

Parameters

- speculationStatus – The expected protect status to set to the PFlash protection register. Each bit is

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidSpeculationOption – An invalid speculation option argument is provided.

status_t FTFx_CACHE_PflashGetPrefetchSpeculation(*ftfx_prefetch_speculation_status_t* *speculationStatus)

Gets the PFlash prefetch speculation status.

Parameters

- speculationStatus – Speculation status returned by the PFlash IP.

Return values

kStatus_FTFx_Success – API was executed successfully.

struct _flash_prefetch_speculation_status

#include <fsl_ftfx_cache.h> FTFx prefetch speculation status.

Public Members

bool instructionOff

Instruction speculation.

bool dataOff

Data speculation.

union function_bit_operation_ptr_t

#include <fsl_ftfx_cache.h>

Public Members

uint32_t commadAddr

void (*callFlashCommand)(volatile uint32_t *base, uint32_t bitMask, uint32_t bitShift, uint32_t bitValue)

struct _ftfx_cache_config

#include <fsl_ftfx_cache.h> FTFx cache driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Public Members

uint8_t flashMemoryIndex

0 - primary flash; 1 - secondary flash

function_bit_operation_ptr_t bitOperFuncAddr

An buffer point to the flash execute-in-RAM function.

2.10 ftfx controller

FTFx driver status codes.

Values:

enumerator kStatus_FTFx_Success

API is executed successfully

enumerator kStatus_FTFx_InvalidArgument

Invalid argument

enumerator kStatus_FTFx_SizeError

Error size

enumerator kStatus_FTFx_AlignmentError

Parameter is not aligned with the specified baseline

enumerator kStatus_FTFx_AddressError

Address is out of range

enumerator kStatus_FTFx_AccessError

Invalid instruction codes and out-of bound addresses

enumerator kStatus_FTFx_ProtectionViolation

The program/erase operation is requested to execute on protected areas

enumerator kStatus_FTFx_CommandFailure

Run-time error during command execution.

enumerator kStatus_FTFx_UnknownProperty

Unknown property.

enumerator kStatus_FTFx_EraseKeyError

API erase key is invalid.

enumerator kStatus_FTFx_RegionExecuteOnly
The current region is execute-only.

enumerator kStatus_FTFx_ExecuteInRamFunctionNotReady
Execute-in-RAM function is not available.

enumerator kStatus_FTFx_PartitionStatusUpdateFailure
Failed to update partition status.

enumerator kStatus_FTFx_SetFlexramAsEepromError
Failed to set FlexRAM as EEPROM.

enumerator kStatus_FTFx_RecoverFlexramAsRamError
Failed to recover FlexRAM as RAM.

enumerator kStatus_FTFx_SetFlexramAsRamError
Failed to set FlexRAM as RAM.

enumerator kStatus_FTFx_RecoverFlexramAsEepromError
Failed to recover FlexRAM as EEPROM.

enumerator kStatus_FTFx_CommandNotSupported
Flash API is not supported.

enumerator kStatus_FTFx_SwapSystemNotInUninitialized
Swap system is not in an uninitialzed state.

enumerator kStatus_FTFx_SwapIndicatorAddressError
The swap indicator address is invalid.

enumerator kStatus_FTFx_ReadOnlyProperty
The flash property is read-only.

enumerator kStatus_FTFx_InvalidPropertyValue
The flash property value is out of range.

enumerator kStatus_FTFx_InvalidSpeculationOption
The option of flash prefetch speculation is invalid.

enumerator kStatus_FTFx_CommandOperationInProgress
The option of flash command is processing.

enum _ftfx_driver_api_keys
Enumeration for FTFx driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFTFx_ApiEraseKey
Key value used to validate all FTFx erase APIs.

void FTFx_API_Init(*ftfx_config_t* *config)
Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

status_t FTFx_API_UpdateFlexnvmPartitionStatus(*ftfx_config_t* *config)

Updates FlexNVM memory partition status according to data flash 0 IFR.

This function updates FlexNVM memory partition status.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FTFx_CMD_Erase(*ftfx_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_EraseSectorNonBlocking(*ftfx_config_t* *config, uint32_t start, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases one flash sector size based on the start address.

Parameters

- config – The pointer to the storage for the driver runtime state.

- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

status_t FTFx_CMD_EraseAll(*ftfx_config_t* *config, uint32_t key)

Erases entire flash.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FTFx_CMD_EraseAllUnsecure(*ftfx_config_t* *config, uint32_t key)

Erases the entire flash, including protected sectors.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FTFx_CMD_EraseAllExecuteOnlySegments(*ftfx_config_t* *config, uint32_t key)

Erases all program flash execute-only segments defined by the FXACC registers.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_Program(*ftfx_config_t* *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.

- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_ProgramOnce(*ftfx_config_t* *config, uint32_t index, const uint8_t *src, uint32_t lengthInBytes)

Programs Program Once Field through parameters.

This function programs the Program Once Field with the desired data for a given flash area as determined by the index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating which area of the Program Once Field to be programmed.
- src – A pointer to the source buffer of data that is to be programmed into the Program Once Field.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_ProgramSection(*ftfx_config_t* *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.

- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_SetFlexramAsRamError – Failed to set flexram as RAM.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_RecoverFlexramAsEepromError – Failed to recover FlexRAM as EEPROM.

status_t FTFx_CMD_ProgramPartition(*ftfx_config_t* *config, *ftfx_partition_flexram_load_opt_t* option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode, uint8_t CSEcKeySize, uint8_t CFE)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Parameters

- config – Pointer to storage for the driver runtime state.
- option – The option used to set FlexRAM load behavior during reset.
- eepromDataSizeCode – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- flexnvmPartitionCode – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FTFx_CMD_ReadOnce(*ftfx_config_t* *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)

Reads the Program Once Field through parameters.

This function reads the read once feild with given index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating the area of program once field to be read.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_ReadResource(*ftfx_config_t* *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, *ftfx_read_resource_opt_t* option)

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- option – The resource option which indicates which area should be read back.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyErase(*ftfx_config_t* *config, uint32_t start, uint32_t lengthInBytes, *ftfx_margin_value_t* margin)

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyEraseAll(*ftfx_config_t* *config, *ftfx_margin_value_t* margin)

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments(*ftfx_config_t* *config, *ftfx_margin_value_t* margin)

Verifies whether the program flash execute-only segments have been erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyProgram(*ftfx_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, *ftfx_margin_value_t* margin, uint32_t *failedAddress, uint32_t *failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- expectedData – A pointer to the expected data that is to be verified against.
- margin – Read margin choice.
- failedAddress – A pointer to the returned failing address.
- failedData – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- kStatus_FTFx_Success – API was executed successfully.

- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_REG_GetSecurityState(*ftfx_config_t* *config, *ftfx_security_state_t* *state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

- config – A pointer to storage for the driver runtime state.
- state – A pointer to the value returned for the current security status code:

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FTFx_CMD_SecurityBypass(*ftfx_config_t* *config, const uint8_t *backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

- config – A pointer to the storage for the driver runtime state.
- backdoorKey – A pointer to the user buffer containing the backdoor key.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_SetFlexramFunction(*ftfx_config_t* *config, *ftfx_flexram_func_opt_t* option)

Sets the FlexRAM function command.

Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_SwapControl(*ftfx_config_t* *config, uint32_t address, *ftfx_swap_control_opt_t* option, *ftfx_swap_state_config_t* *returnInfo)

Configures the Swap function or checks the swap state of the Flash module.

Parameters

- config – A pointer to the storage for the driver runtime state.
- address – Address used to configure the flash Swap function.
- option – The possible option used to configure Flash Swap function or check the flash Swap status
- returnInfo – A pointer to the data which is used to return the information of flash Swap.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_SwapIndicatorAddressError – Swap indicator address is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

enum _ftfx_partition_flexram_load_option

Enumeration for the FlexRAM load during reset option.

Values:

enumerator kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData

FlexRAM is loaded with valid EEPROM data during reset sequence.

enumerator kFTFx_PartitionFlexramLoadOptNotLoaded
FlexRAM is not loaded during reset sequence.

enum _ftfx_read_resource_opt
Enumeration for the two possible options of flash read resource command.

Values:

enumerator kFTFx_ResourceOptionFlashIfr
Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR

enumerator kFTFx_ResourceOptionVersionId
Select code for the version ID

enum _ftfx_margin_value
Enumeration for supported FTFx margin levels.

Values:

enumerator kFTFx_MarginValueNormal
Use the 'normal' read level for 1s.

enumerator kFTFx_MarginValueUser
Apply the 'User' margin to the normal read-1 level.

enumerator kFTFx_MarginValueFactory
Apply the 'Factory' margin to the normal read-1 level.

enumerator kFTFx_MarginValueInvalid
Not real margin level, Used to determine the range of valid margin level.

enum _ftfx_security_state
Enumeration for the three possible FTFx security states.

Values:

enumerator kFTFx_SecurityStateNotSecure
Flash is not secure.

enumerator kFTFx_SecurityStateBackdoorEnabled
Flash backdoor is enabled.

enumerator kFTFx_SecurityStateBackdoorDisabled
Flash backdoor is disabled.

enum _ftfx_flexram_function_option
Enumeration for the two possilbe options of set FlexRAM function command.

Values:

enumerator kFTFx_FlexramFuncOptAvailableAsRam
An option used to make FlexRAM available as RAM

enumerator kFTFx_FlexramFuncOptEepromQuickWriteRecovery
An option used to complete interrupted EEPROM quick write process

enumerator kFTFx_FlexramFuncOptEepromQuickWriteStatus
An option used to make EEPROM quick write status query

enumerator kFTFx_FlexramFuncOptAvailableForEepromQuickWrite
An option used to make FlexRAM available for EEPROM in Quick Write mode

enumerator kFTFx_FlexramFuncOptAvailableForEeprom
An option used to make FlexRAM available for EEPROM

enum _flash_acceleration_ram_property
  Enumeration for acceleration ram property.

  Values:

  enumerator kFLASH_AccelerationRamSize

enum _ftfx_swap_control_option
  Enumeration for the possible options of Swap control commands.

  Values:

  enumerator kFTFx_SwapControlOptionIntializeSystem
    An option used to initialize the Swap system

  enumerator kFTFx_SwapControlOptionSetInUpdateState
    An option used to set the Swap in an update state

  enumerator kFTFx_SwapControlOptionSetInCompleteState
    An option used to set the Swap in a complete state

  enumerator kFTFx_SwapControlOptionReportStatus
    An option used to report the Swap status

  enumerator kFTFx_SwapControlOptionDisableSystem
    An option used to disable the Swap status

enum _ftfx_swap_state
  Enumeration for the possible flash Swap status.

  Values:

  enumerator kFTFx_SwapStateUninitialized
    Flash Swap system is in an uninitialized state.

  enumerator kFTFx_SwapStateReady
    Flash Swap system is in a ready state.

  enumerator kFTFx_SwapStateUpdate
    Flash Swap system is in an update state.

  enumerator kFTFx_SwapStateUpdateErased
    Flash Swap system is in an updateErased state.

  enumerator kFTFx_SwapStateComplete
    Flash Swap system is in a complete state.

  enumerator kFTFx_SwapStateDisabled
    Flash Swap system is in a disabled state.

enum _ftfx_swap_block_status
  Enumeration for the possible flash Swap block status.

  Values:

  enumerator kFTFx_SwapBlockStatusLowerHalfProgramBlocksAtZero
    Swap block status is that lower half program block at zero.

  enumerator kFTFx_SwapBlockStatusUpperHalfProgramBlocksAtZero
    Swap block status is that upper half program block at zero.

enum _ftfx_memory_type
  Enumeration for FTFx memory type.

  Values:

enumerator kFTFx_MemTypePflash

enumerator kFTFx_MemTypeFlexnvm

typedef enum *_ftfx_partition_flexram_load_option* ftfx_partition_flexram_load_opt_t
Enumeration for the FlexRAM load during reset option.

typedef enum *_ftfx_read_resource_opt* ftfx_read_resource_opt_t
Enumeration for the two possible options of flash read resource command.

typedef enum *_ftfx_margin_value* ftfx_margin_value_t
Enumeration for supported FTFx margin levels.

typedef enum *_ftfx_security_state* ftfx_security_state_t
Enumeration for the three possible FTFx security states.

typedef enum *_ftfx_flexram_function_option* ftfx_flexram_func_opt_t
Enumeration for the two possilbe options of set FlexRAM function command.

typedef enum *_ftfx_swap_control_option* ftfx_swap_control_opt_t
Enumeration for the possible options of Swap control commands.

typedef enum *_ftfx_swap_state* ftfx_swap_state_t
Enumeration for the possible flash Swap status.

typedef enum *_ftfx_swap_block_status* ftfx_swap_block_status_t
Enumeration for the possible flash Swap block status.

typedef struct *_ftfx_swap_state_config* ftfx_swap_state_config_t
Flash Swap information.

typedef struct *_ftfx_special_mem* ftfx_spec_mem_t
ftfx special memory access information.

typedef struct *_ftfx_mem_descriptor* ftfx_mem_desc_t
Flash memory descriptor.

typedef struct *_ftfx_ops_config* ftfx_ops_config_t
Active FTFx information for the current operation.

typedef struct *_ftfx_ifr_descriptor* ftfx_ifr_desc_t
Flash IFR memory descriptor.

typedef struct *_ftfx_config* ftfx_config_t
Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

struct _ftfx_swap_state_config
#include <fsl_ftfx_controller.h> Flash Swap information.

Public Members

ftfx_swap_state_t flashSwapState
The current Swap system status.

ftfx_swap_block_status_t currentSwapBlockStatus
The current Swap block status.

ftfx_swap_block_status_t nextSwapBlockStatus
The next Swap block status.

struct _ftfx_special_mem
#include <fsl_ftfx_controller.h> ftfx special memory access information.

Public Members

uint32_t base
Base address of flash special memory.

uint32_t size
size of flash special memory.

uint32_t count
flash special memory count.

struct _ftfx_mem_descriptor
#include <fsl_ftfx_controller.h> Flash memory descriptor.

Public Members

uint32_t blockBase
A base address of the flash block

uint32_t aliasBlockBase
A base address of the alias flash block

uint32_t totalSize
The size of the flash block.

uint32_t sectorSize
The size in bytes of a sector of flash.

uint32_t blockCount
A number of flash blocks.

struct _ftfx_ops_config
#include <fsl_ftfx_controller.h> Active FTFx information for the current operation.

Public Members

uint32_t convertedAddress
A converted address for the current flash type.

struct _ftfx_ifr_descriptor
#include <fsl_ftfx_controller.h> Flash IFR memory descriptor.

union function_ptr_t
#include <fsl_ftfx_controller.h>

Public Members

uint32_t commadAddr

void (*callFlashCommand)(volatile uint8_t *FTMRx_fstat)

struct _ftfx_config
#include <fsl_ftfx_controller.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Public Members

uint32_t flexramBlockBase
: The base address of the FlexRAM/acceleration RAM

uint32_t flexramTotalSize
: The size of the FlexRAM/acceleration RAM

uint16_t eepromTotalSize
: The size of EEPROM area which was partitioned from FlexRAM

function_ptr_t runCmdFuncAddr
: An buffer point to the flash execute-in-RAM function.

struct ___unnamed8___

Public Members

uint8_t type
: Type of flash block.

uint8_t index
: Index of flash block.

struct feature

struct addrAligment

struct feature

struct resRange

Public Members

uint8_t versionIdStart
: Version ID start address

uint32_t pflashIfrStart
: Program Flash 0 IFR start address

uint32_t dflashIfrStart
: Data Flash 0 IFR start address

uint32_t pflashSwapIfrStart
: Program Flash Swap IFR start address

struct idxInfo

2.11 ftfx feature

FTFx_DRIVER_IS_FLASH_RESIDENT
: Flash driver location.

  Used for the flash resident application.

FTFx_DRIVER_IS_EXPORTED
: Flash Driver Export option.

  Used for the MCUXpresso SDK application.

FTFx_FLASH1_HAS_PROT_CONTROL

Indicates whether the secondary flash has its own protection register in flash module.

FTFx_FLASH1_HAS_XACC_CONTROL

Indicates whether the secondary flash has its own Execute-Only access register in flash module.

FTFx_DRIVER_HAS_FLASH1_SUPPORT

Indicates whether the secondary flash is supported in the Flash driver.

FTFx_FLASH_COUNT

FTFx_FLASH1_IS_INDEPENDENT_BLOCK

2.12 Ftftx FLASH Driver

status_t FLASH_Init(*flash_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLASH_Erase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.

- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_EraseSectorNonBlocking(*flash_config_t* *config, uint32_t start, uint32_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases one flash sector size based on the start address, and it is executed asynchronously.

NOTE: This function can only erase one flash sector at a time, and the other commands can be executed after the previous command has been completed.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.

status_t FLASH_EraseAll(*flash_config_t* *config, uint32_t key)

Erases entire flexnvm.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the all pflash and flexnvm were erased successfully, the swap and eeprom have been reset to unconfigured state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.

- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLASH_EraseAllUnsecure(*flash_config_t* *config, uint32_t key)

Erases the entire flexnvm, including protected sectors.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the protected sectors of flash were reset to unprotected status.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLASH_Program(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data were programed successfully into flash based on desired start address and length.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_ProgramOnce(*flash_config_t* *config, uint32_t index, uint8_t *src, uint32_t lengthInBytes)

Program the Program-Once-Field through parameters.

This function Program the Program-once-feild with given index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating the area of program once field to be read.
- src – A pointer to the source buffer of data that is used to store data to be write.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; The index indicating the area of program once field was programed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_ProgramSection(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.

- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data have been programed successfully into flash based on start address and length.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_SetFlexramAsRamError – Failed to set flexram as RAM.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_RecoverFlexramAsEepromError – Failed to recover FlexRAM as EEPROM.

status_t FLASH_ReadResource(*flash_config_t* *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, *ftfx_read_resource_opt_t* option)

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- option – The resource option which indicates which area should be read back.

Return values

- kStatus_FTFx_Success – API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.

- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_ReadOnce(*flash_config_t* *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)

Reads the Program Once Field through parameters.

This function reads the read once feild with given index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating the area of program once field to be read.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the data have been successfuly read form Program flash0 IFR map and Program Once field based on index and length.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_VerifyErase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, *ftfx_margin_value_t* margin)

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified FLASH region has been erased.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_VerifyEraseAll(*flash_config_t* *config, *ftfx_margin_value_t* margin)

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; all program flash and flexnvm were in erased state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_VerifyProgram(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, *ftfx_margin_value_t* margin, uint32_t *failedAddress, uint32_t *failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.

- expectedData – A pointer to the expected data that is to be verified against.
- margin – Read margin choice.
- failedAddress – A pointer to the returned failing address.
- failedData – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data have been successfully programed into specified FLASH region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_GetSecurityState(*flash_config_t* *config, *ftfx_security_state_t* *state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

- config – A pointer to storage for the driver runtime state.
- state – A pointer to the value returned for the current security status code:

Return values

- kStatus_FTFx_Success – API was executed successfully; the security state of flash was stored to state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FLASH_SecurityBypass(*flash_config_t* *config, const uint8_t *backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

- config – A pointer to the storage for the driver runtime state.
- backdoorKey – A pointer to the user buffer containing the backdoor key.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_SetFlexramFunction(*flash_config_t* *config, *ftfx_flexram_func_opt_t* option)

Sets the FlexRAM function command.

Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_Swap(*flash_config_t* *config, uint32_t address, bool isSetEnable)

Swaps the lower half flash with the higher half flash.

Parameters

- config – A pointer to the storage for the driver runtime state.
- address – Address used to configure the flash swap function
- isSetEnable – The possible option used to configure the Flash Swap function or check the flash Swap status.

Return values

- kStatus_FTFx_Success – API was executed successfully; the lower half flash and higher half flash have been swaped.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_SwapIndicatorAddressError – Swap indicator address is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.

- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_SwapSystemNotInUninitialized – Swap system is not in an uninitialized state.

status_t FLASH_IsProtected(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, *flash_prot_state_t* *protection_state)

Returns the protection state of the desired flash area via the pointer passed into the function.

This function retrieves the current flash protect status for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be checked. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.
- protection_state – A pointer to the value returned for the current protection status code for the desired flash area.

Return values

- kStatus_FTFx_Success – API was executed successfully; the protection state of specified FLASH region was stored to protection_state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.

status_t FLASH_IsExecuteOnly(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, *flash_xacc_state_t* *access_state)

Returns the access state of the desired flash area via the pointer passed into the function.

This function retrieves the current flash access status for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be checked. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be checked. Must be word-aligned.
- access_state – A pointer to the value returned for the current access status code for the desired flash area.

Return values

- kStatus_FTFx_Success – API was executed successfully; the executeOnly state of specified FLASH region was stored to access_state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

- kStatus_FTFx_AlignmentError – The parameter is not aligned to the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.

status_t FLASH_PflashSetProtection(*flash_config_t* *config, *pflash_prot_status_t* *protectStatus)

Sets the PFlash Protection to the intended protection status.

Parameters

- config – A pointer to storage for the driver runtime state.
- protectStatus – The expected protect status to set to the PFlash protection register. Each bit is corresponding to protection of 1/32(64) of the total PFlash. The least significant bit is corresponding to the lowest address area of PFlash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified FLASH region is protected.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLASH_PflashGetProtection(*flash_config_t* *config, *pflash_prot_status_t* *protectStatus)

Gets the PFlash protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – Protect status returned by the PFlash IP. Each bit is corresponding to the protection of 1/32(64) of the total PFlash. The least significant bit corresponds to the lowest address area of the PFlash. The most significant bit corresponds to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully; the Protection state was stored to protectStatus;
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FLASH_GetProperty(*flash_config_t* *config, *flash_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flash property.

Parameters

- config – A pointer to the storage for the driver runtime state.
- whichProperty – The desired property from the list of properties in enum flash_property_tag_t
- value – A pointer to the value returned for the desired flash property.

Return values

- kStatus_FTFx_Success – API was executed successfully; the flash property was stored to value.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

- kStatus_FTFx_UnknownProperty – An unknown property tag.

status_t FLASH_GetCommandState(void)

Get previous command status.

This function is used to obtain the execution status of the previous command.

Return values

- kStatus_FTFx_Success – The previous command is executed successfully.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

FSL_FLASH_DRIVER_VERSION

Flash driver version for SDK.

Version 3.3.0.

FSL_FLASH_DRIVER_VERSION_ROM

Flash driver version for ROM.

Version 3.0.0.

enum _flash_protection_state

Enumeration for the three possible flash protection levels.

Values:

enumerator kFLASH_ProtectionStateUnprotected

Flash region is not protected.

enumerator kFLASH_ProtectionStateProtected

Flash region is protected.

enumerator kFLASH_ProtectionStateMixed

Flash is mixed with protected and unprotected region.

enum _flash_execute_only_access_state

Enumeration for the three possible flash execute access levels.

Values:

enumerator kFLASH_AccessStateUnLimited

Flash region is unlimited.

enumerator kFLASH_AccessStateExecuteOnly

Flash region is execute only.

enumerator kFLASH_AccessStateMixed

Flash is mixed with unlimited and execute only region.

enum _flash_property_tag

Enumeration for various flash properties.

Values:

enumerator kFLASH_PropertyPflash0SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash0TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash0BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash0BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash0BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash0FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash0AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash0AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyPflash1SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash1TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash1BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash1BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash1BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash1FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash1AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash1AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyFlexRamBlockBaseAddr
FlexRam block base address property.

enumerator kFLASH_PropertyFlexRamTotalSize
FlexRam total size property.

typedef enum *_flash_protection_state* flash_prot_state_t
Enumeration for the three possible flash protection levels.

typedef union *_pflash_protection_status* pflash_prot_status_t
PFlash protection status.

typedef enum *_flash_execute_only_access_state* flash_xacc_state_t
Enumeration for the three possible flash execute access levels.

typedef enum *_flash_property_tag* flash_property_tag_t

Enumeration for various flash properties.

typedef struct *_flash_config* flash_config_t

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

kStatus_FLASH_Success

kFLASH_ApiEraseKey

union _pflash_protection_status

#include <fsl_ftfx_flash.h> PFlash protection status.

Public Members

uint32_t protl

PROT[31:0] .

uint32_t proth

PROT[63:32].

uint8_t protsl

PROTS[7:0] .

uint8_t protsh

PROTS[15:8] .

uint8_t reserved[2]

struct _flash_config

#include <fsl_ftfx_flash.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

2.13 Ftftx FLEXNVM Driver

status_t FLEXNVM_Init(*flexnvm_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLEXNVM_DflashErase(*flexnvm_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the appropriate number of date flash sectors based on the desired start address and length were erased successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_EraseAll(*flexnvm_config_t* *config, uint32_t key)

Erases entire flexnvm.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the entire flexnvm has been erased successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.

- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLEXNVM_EraseAllUnsecure(*flexnvm_config_t* *config, uint32_t key)

Erases the entire flexnvm, including protected sectors.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the flexnvm is not in securityi state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLEXNVM_DflashProgram(*flexnvm_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired date have been successfully programed into specified date flash region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.

- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashProgramSection(*flexnvm_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired date have been successfully programed into specified date flash area.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_SetFlexramAsRamError – Failed to set flexram as RAM.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_RecoverFlexramAsEepromError – Failed to recover FlexRAM as EEPROM.

status_t FLEXNVM_ProgramPartition(*flexnvm_config_t* *config, *ftfx_partition_flexram_load_opt_t* option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Parameters

- config – Pointer to storage for the driver runtime state.
- option – The option used to set FlexRAM load behavior during reset.
- eepromDataSizeCode – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- flexnvmPartitionCode – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_ProgramPartition_CSE(*flexnvm_config_t* *config, *ftfx_partition_flexram_load_opt_t* option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode, uint8_t CSEcKeySize, uint8_t SFE)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM. This is the CSE enabled version for IP's like FTFC.

Parameters

- config – Pointer to storage for the driver runtime state.
- option – The option used to set FlexRAM load behavior during reset.
- eepromDataSizeCode – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- flexnvmPartitionCode – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.
- CSEcKeySize – CSEc/SHE key size, see RM for details and possible values
- SFE – Security Flag Extension (SFE), see RM for details and possible values

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_ReadResource(*flexnvm_config_t* *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, *ftfx_read_resource_opt_t* option)

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- option – The resource option which indicates which area should be read back.

Return values

- kStatus_FTFx_Success – API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashVerifyErase(*flexnvm_config_t* *config, uint32_t start, uint32_t lengthInBytes, *ftfx_margin_value_t* margin)

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.

- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified data flash region is in erased state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_VerifyEraseAll(*flexnvm_config_t* *config, *ftfx_margin_value_t* margin)

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; the entire flexnvm region is in erased state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashVerifyProgram(*flexnvm_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, *ftfx_margin_value_t* margin, uint32_t *failedAddress, uint32_t *failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- expectedData – A pointer to the expected data that is to be verified against.
- margin – Read margin choice.
- failedAddress – A pointer to the returned failing address.
- failedData – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data hve been programed successfully into specified data flash region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_GetSecurityState(*flexnvm_config_t* *config, *ftfx_security_state_t* *state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

- config – A pointer to storage for the driver runtime state.
- state – A pointer to the value returned for the current security status code:

Return values

- kStatus_FTFx_Success – API was executed successfully; the security state of flexnvm was stored to state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FLEXNVM_SecurityBypass(*flexnvm_config_t* *config, const uint8_t *backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

- config – A pointer to the storage for the driver runtime state.
- backdoorKey – A pointer to the user buffer containing the backdoor key.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_SetFlexramFunction(*flexnvm_config_t* *config, *ftfx_flexram_func_opt_t* option)

Sets the FlexRAM function command.

Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashSetProtection(*flexnvm_config_t* *config, uint8_t protectStatus)

Sets the DFlash protection to the intended protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – The expected protect status to set to the DFlash protection register. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified DFlash region is protected.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandNotSupported – Flash API is not supported.

- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_DflashGetProtection(*flexnvm_config_t* *config, uint8_t *protectStatus)

Gets the DFlash protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash, and so on. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandNotSupported – Flash API is not supported.

status_t FLEXNVM_EepromSetProtection(*flexnvm_config_t* *config, uint8_t protectStatus)

Sets the EEPROM protection to the intended protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – The expected protect status to set to the EEPROM protection register. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of EEPROM, and so on. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandNotSupported – Flash API is not supported.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_EepromGetProtection(*flexnvm_config_t* *config, uint8_t *protectStatus)

Gets the EEPROM protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of the EEPROM. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

- kStatus_FTFx_CommandNotSupported – Flash API is not supported.

status_t FLEXNVM_GetProperty(*flexnvm_config_t* *config, *flexnvm_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flexnvm property.

Parameters

- config – A pointer to the storage for the driver runtime state.
- whichProperty – The desired property from the list of properties in enum flexnvm_property_tag_t
- value – A pointer to the value returned for the desired flexnvm property.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_UnknownProperty – An unknown property tag.

enum _flexnvm_property_tag

Enumeration for various flexnvm properties.

Values:

enumerator kFLEXNVM_PropertyDflashSectorSize

Dflash sector size property.

enumerator kFLEXNVM_PropertyDflashTotalSize

Dflash total size property.

enumerator kFLEXNVM_PropertyDflashBlockSize

Dflash block size property.

enumerator kFLEXNVM_PropertyDflashBlockCount

Dflash block count property.

enumerator kFLEXNVM_PropertyDflashBlockBaseAddr

Dflash block base address property.

enumerator kFLEXNVM_PropertyAliasDflashBlockBaseAddr

Dflash block base address Alias property.

enumerator kFLEXNVM_PropertyFlexRamBlockBaseAddr

FlexRam block base address property.

enumerator kFLEXNVM_PropertyFlexRamTotalSize

FlexRam total size property.

enumerator kFLEXNVM_PropertyEepromTotalSize

EEPROM total size property.

typedef enum *_flexnvm_property_tag* flexnvm_property_tag_t

Enumeration for various flexnvm properties.

typedef struct *_flexnvm_config* flexnvm_config_t

Flexnvm driver state information.

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

status_t FLEXNVM_EepromWrite(*flexnvm_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs the EEPROM with data at locations passed in through parameters.

This function programs the emulated EEPROM with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desires data have been successfully programed into specified eeprom region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_SetFlexramAsEepromError – Failed to set flexram as eeprom.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_RecoverFlexramAsRamError – Failed to recover the FlexRAM as RAM.

struct _flexnvm_config

#include <fsl_ftfx_flexnvm.h> Flexnvm driver state information.

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

2.14 ftfx utilities

ALIGN_DOWN(x, a)

Alignment(down) utility.

ALIGN_UP(x, a)

Alignment(up) utility.

MAKE_VERSION(major, minor, bugfix)

Constructs the version number for drivers.

MAKE_STATUS(group, code)

Constructs a status code value from a group and a code number.

FOUR_CHAR_CODE(a, b, c, d)

Constructs the four character code for the Flash driver API key.

B1P4(b)

bytes2word utility.

B1P3(b)

B1P2(b)

B1P1(b)

B2P3(b)

B2P2(b)

B2P1(b)

B3P2(b)

B3P1(b)

BYTE2WORD_1_3(x, y)

BYTE2WORD_2_2(x, y)

BYTE2WORD_3_1(x, y)

BYTE2WORD_1_1_2(x, y, z)

BYTE2WORD_1_2_1(x, y, z)

BYTE2WORD_2_1_1(x, y, z)

BYTE2WORD_1_1_1_1(x, y, z, w)

2.15 FTM: FlexTimer Driver

status_t FTM_Init(FTM_Type *base, const *ftm_config_t* *config)

Ungates the FTM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application which is using the FTM driver. If the FTM instance has only TPM features, please use the TPM driver.

Parameters

- base – FTM peripheral base address
- config – Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

void FTM_Deinit(FTM_Type *base)

Gates the FTM clock.

Parameters

- base – FTM peripheral base address

void FTM_GetDefaultConfig(*ftm_config_t* *config)

Fills in the FTM configuration structure with the default settings.

The default values are:

```
config->prescale = kFTM_Prescale_Divide_1;
config->bdmMode = kFTM_BdmMode_0;
config->pwmSyncMode = kFTM_SoftwareTrigger;
config->reloadPoints = 0;
config->faultMode = kFTM_Fault_Disable;
config->faultFilterValue = 0;
config->deadTimePrescale = kFTM_Deadtime_Prescale_1;
config->deadTimeValue =  0;
config->extTriggers = 0;
config->chnlInitState = 0;
config->chnlPolarity = 0;
config->useGlobalTimeBase = false;
config->hwTriggerResetCount = false;
config->swTriggerResetCount = true;
```

Parameters

- config – Pointer to the user configuration structure.

static inline *ftm_clock_prescale_t* FTM_CalculateCounterClkDiv(FTM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

brief Calculates the counter clock prescaler.

This function calculates the values for SC[PS] bit.

param base FTM peripheral base address param counterPeriod_Hz The desired frequency in Hz which corresponding to the time when the counter reaches the mod value param srcClock_Hz FTM counter clock in Hz

return Calculated clock prescaler value, see ftm_clock_prescale_t.

status_t FTM_SetupPwm(FTM_Type *base, const *ftm_chnl_pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *ftm_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.

Call this function to configure the PWM signal period, mode, duty cycle, and edge. Use this function to configure all FTM channels that are used to output a PWM signal.

Parameters

- base – FTM 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 ftm_pwm_mode_t
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – FTM counter clock in Hz

Returns

kStatus_Success if the PWM setup was successful kStatus_Error on failure

status_t FTM_UpdatePwmDutycycle(FTM_Type *base, *ftm_chnl_t* chnlNumber, *ftm_pwm_mode_t* currentPwmMode, uint8_t dutyCyclePercent)

Updates the duty cycle of an active PWM signal.

Parameters

- base – FTM peripheral base address
- chnlNumber – The channel/channel pair number. In combined mode, this represents the channel pair number
- currentPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width; The 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 update was successful kStatus_Error on failure

void FTM_UpdateChnlEdgeLevelSelect(FTM_Type *base, *ftm_chnl_t* chnlNumber, uint8_t level)

Updates the edge level selection for a channel.

Parameters

- base – FTM 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 Kinetis SoC reference manual for details about this field.

status_t FTM_SetupPwmMode(FTM_Type *base, const *ftm_chnl_pwm_config_param_t* *chnlParams, uint8_t numOfChnls, *ftm_pwm_mode_t* mode)

Configures the PWM mode parameters.

Call this function to configure the PWM signal mode, duty cycle in ticks, and edge. Use this function to configure all FTM channels that are used to output a PWM signal. Please note that: This API is similar with FTM_SetupPwm() API, but will not set the timer period, and this API will set channel match value in timer ticks, not period percent.

Parameters

- base – FTM 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 ftm_pwm_mode_t

Returns
kStatus_Success if the PWM setup was successful kStatus_Error on failure

void FTM_ConfigSinglePWM(FTM_Type *base, const *ftm_chnl_param_t* *chnlParams, *ftm_chnl_t* chnlNumber)

Configure FTM edge aligned PWM or center aligned PWM by each channel.

This function configure PWM signal by setting channel n value register. Need to invoke FTM_SetInitialModuloValue to configure FTM period.

Parameters

- base – FTM peripheral base address
- chnlParams – PWM configuration structure pointer.
- chnlPairNumber – Channel number.

void FTM_ConfigCombinePWM(FTM_Type *base, const *ftm_chnl_param_t* *chnlParams, *ftm_chnl_t* chnlPairNumber)

Configure FTM Combine PWM, Modified Combine PWM or Asymmetrical PWM by each channel pair.

This function configure PWM signal by setting channel n value register. Need to invoke FTM_SetInitialModuloValue to configure FTM period.

Parameters

- base – FTM peripheral base address
- chnlParams – PWM configuration structure pointer.
- chnlPairNumber – Channel pair number, options are 0, 1, 2, 3.

void FTM_SetupInputCapture(FTM_Type *base, *ftm_chnl_t* chnlNumber, *ftm_input_capture_edge_t* captureMode, uint32_t filterValue)

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 FTM counter is captured into the CnV register. The user has to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only for channels 0, 1, 2, 3.

Parameters

- base – FTM peripheral base address
- chnlNumber – The channel number
- captureMode – Specifies which edge to capture
- filterValue – Filter value, specify 0 to disable filter. Available only for channels 0-3.

void FTM_SetupOutputCompare(FTM_Type *base, *ftm_chnl_t* chnlNumber, *ftm_output_compare_mode_t* compareMode, uint32_t compareValue)

Configures the FTM to generate timed pulses.

When the FTM 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 – FTM 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.

void FTM_SetupDualEdgeCapture(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, const *ftm_dual_edge_capture_param_t* *edgeParam, uint32_t filterValue)

Configures the dual edge capture mode of the FTM.

This function sets up the dual edge capture mode on a channel pair. The capture edge for the channel pair and the capture mode (one-shot or continuous) is specified in the parameter argument. The filter function is disabled if the filterVal argument passed is zero. The filter function is available only on channels 0 and 2. The user has to read the channel CnV registers separately to get the capture values.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM 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. Available only for channel pair 0 and 1.

void FTM_EnableInterrupts(FTM_Type *base, uint32_t mask)

Enables the selected FTM interrupts.

Parameters

- base – FTM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration ftm_interrupt_enable_t

void FTM_DisableInterrupts(FTM_Type *base, uint32_t mask)

Disables the selected FTM interrupts.

Parameters

- base – FTM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration ftm_interrupt_enable_t

uint32_t FTM_GetEnabledInterrupts(FTM_Type *base)

Gets the enabled FTM interrupts.

Parameters

- base – FTM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration ftm_interrupt_enable_t

uint32_t FTM_GetInstance(FTM_Type *base)

Gets the instance from the base address.

Parameters

- base – FTM peripheral base address

Returns

The FTM instance

uint32_t FTM_GetStatusFlags(FTM_Type *base)

Gets the FTM status flags.

Parameters

- base – FTM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration ftm_status_flags_t

void FTM_ClearStatusFlags(FTM_Type *base, uint32_t mask)

Clears the FTM status flags.

Parameters

- base – FTM peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration ftm_status_flags_t

static inline void FTM_SetTimerPeriod(FTM_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 FTM module as a timer. Do not mix usage of this API with FTM'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 – FTM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

static inline void FTM_SetInitialModuloValue(FTM_Type *base, uint16_t initialValue, uint16_t moduloValue)

Set initial value and modulo value for FTM.

Parameters

- base – FTM peripheral base address
- initialValue – FTM counter initial value.
- moduloValue – FTM counter modulo value.

static inline uint32_t FTM_GetCurrentTimerCount(FTM_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 – FTM peripheral base address

Returns

The current counter value in ticks

static inline void FTM_SetChannelMatchValue(FTM_Type *base, *ftm_chnl_t* chnlNumber, uint16_t value)

Set channel match value for output.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to set.
- value – Channel match value for output.

static inline uint32_t FTM_GetInputCaptureValue(FTM_Type *base, *ftm_chnl_t* chnlNumber)

Reads the captured value.

This function returns the captured value of a FTM channel configured in input capture or dual edge capture mode.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be read

Returns
The captured FTM counter value of the input modes.

static inline void FTM_StartTimer(FTM_Type *base, *ftm_clock_source_t* clockSource)

Starts the FTM counter.

Parameters

- base – FTM peripheral base address
- clockSource – FTM clock source; After the clock source is set, the counter starts running.

static inline void FTM_StopTimer(FTM_Type *base)

Stops the FTM counter.

Parameters

- base – FTM peripheral base address

static inline uint32_t FTM_GetSoftwareOutputValue(FTM_Type *base)

Get channel software output status.

Parameters

- base – FTM peripheral base address

Returns
Status of channel software output, logical OR value of ftm_channel_index_t.

static inline uint32_t FTM_GetSoftwareOutputEnable(FTM_Type *base)

Get channel software enable status.

Parameters

- base – FTM peripheral base address

Returns
Status of channel software enable, logical OR value of ftm_channel_index_t.

static inline void FTM_SetSoftwareOutputCtrl(FTM_Type *base, uint32_t chnlEnable, uint32_t chnlValue)

Enables or disables the channel software output control and set channel software output value.

Parameters

- base – FTM peripheral base address
- chnlEnable – Channels to enable or disable software output control, logical OR of enumeration ftm_channel_index_t members.
- chnlValue – Channels output value, logical OR of enumeration ftm_channel_index_t members

static inline void FTM_SetSoftwareCtrlEnable(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool value)

Enables or disables the channel software output control.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be enabled or disabled
- value – true: channel output is affected by software output control false: channel output is unaffected by software output control

static inline void FTM_SetSoftwareCtrlVal(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool value)

Sets the channel software output control value.

Parameters

- base – FTM peripheral base address.
- chnlNumber – Channel to be configured
- value – true to set 1, false to set 0

static inline void FTM_SetFaultControlEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables the fault control in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: Enable fault control for this channel pair; false: No fault control

static inline void FTM_SetDeadTimeEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables the dead time insertion in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: Insert dead time in this channel pair; false: No dead time inserted

static inline void FTM_SetComplementaryEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables complementary mode in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: enable complementary mode; false: disable complementary mode

static inline void FTM_SetInvertEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables inverting control in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: enable inverting; false: disable inverting

void FTM_SetupQuadDecode(FTM_Type *base, const *ftm_phase_params_t* *phaseAParams, const *ftm_phase_params_t* *phaseBParams, *ftm_quad_decode_mode_t* quadMode)

Configures the parameters and activates the quadrature decoder mode.

Parameters

- base – FTM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters
- quadMode – Selects encoding mode used in quadrature decoder mode

static inline uint32_t FTM_GetQuadDecoderFlags(FTM_Type *base)

Gets the FTM Quad Decoder flags.

Parameters

- base – FTM peripheral base address.

Returns

Flag mask of FTM Quad Decoder, see _ftm_quad_decoder_flags.

static inline void FTM_SetQuadDecoderModuloValue(FTM_Type *base, uint32_t startValue, uint32_t overValue)

Sets the modulo values for Quad Decoder.

The modulo values configure the minimum and maximum values that the Quad decoder counter can reach. After the counter goes over, the counter value goes to the other side and decrease/increase again.

Parameters

- base – FTM peripheral base address.
- startValue – The low limit value for Quad Decoder counter.
- overValue – The high limit value for Quad Decoder counter.

static inline uint32_t FTM_GetQuadDecoderCounterValue(FTM_Type *base)

Gets the current Quad Decoder counter value.

Parameters

- base – FTM peripheral base address.

Returns

Current quad Decoder counter value.

static inline void FTM_ClearQuadDecoderCounterValue(FTM_Type *base)

Clears the current Quad Decoder counter value.

The counter is set as the initial value.

Parameters

- base – FTM peripheral base address.

FSL_FTM_DRIVER_VERSION

FTM driver version 2.7.4.

enum _ftm_chnl

List of FTM channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kFTM_Chnl_0
FTM channel number 0

enumerator kFTM_Chnl_1
FTM channel number 1

enumerator kFTM_Chnl_2
FTM channel number 2

enumerator kFTM_Chnl_3
FTM channel number 3

enumerator kFTM_Chnl_4
FTM channel number 4

enumerator kFTM_Chnl_5
FTM channel number 5

enumerator kFTM_Chnl_6
FTM channel number 6

enumerator kFTM_Chnl_7
FTM channel number 7

enum _ftm_fault_input
List of FTM faults.

Values:

enumerator kFTM_Fault_0
FTM fault 0 input pin

enumerator kFTM_Fault_1
FTM fault 1 input pin

enumerator kFTM_Fault_2
FTM fault 2 input pin

enumerator kFTM_Fault_3
FTM fault 3 input pin

enum _ftm_pwm_mode
FTM PWM operation modes.

Values:

enumerator kFTM_EdgeAlignedPwm
Edge-aligned PWM

enumerator kFTM_CenterAlignedPwm
Center-aligned PWM

enumerator kFTM_EdgeAlignedCombinedPwm
Edge-aligned combined PWM

enumerator kFTM_CenterAlignedCombinedPwm
Center-aligned combined PWM

enumerator kFTM_ModifiedCombinedPwm
Modified combined PWM

enumerator kFTM_AsymmetricalCombinedPwm
    Asymmetrical combined PWM

enum _ftm_pwm_level_select
    FTM PWM output pulse mode: high-true, low-true or no output.

Note: kFTM_NoPwmSignal: ELSnB:ELSnA = 0:0 kFTM_LowTrue: ELSnB:ELSnA = 0:1 EPWM: Channel n output is forced low at counter overflow, forced high at channel n match. CPWM: Channel n output is forced low at channel n match when counting down, and forced high at channel n match when counting up. Combined PWM: Channel n output is forced high at beginning of period and at channel n+1 match. It is forced low at the channel n match. kFTM_HighTrue: ELSnB:ELSnA = 1:0 EPWM: Channel n output is forced high at counter overflow, forced low at channel n match. CPWM: Channel n output is forced high at channel n match when counting down, and forced low at channel n match when counting up. Combined PWM: Channel n output is forced low at beginning of period and at channel n+1 match. It is forced high at the channel n match.

Values:

enumerator kFTM_NoPwmSignal
    No PWM output on pin

enumerator kFTM_LowTrue
    Low true pulses

enumerator kFTM_HighTrue
    High true pulses

enum _ftm_output_compare_mode
    FlexTimer output compare mode.

Values:

enumerator kFTM_NoOutputSignal
    No channel output when counter reaches CnV

enumerator kFTM_ToggleOnMatch
    Toggle output

enumerator kFTM_ClearOnMatch
    Clear output

enumerator kFTM_SetOnMatch
    Set output

enum _ftm_input_capture_edge
    FlexTimer input capture edge.

Values:

enumerator kFTM_RisingEdge
    Capture on rising edge only

enumerator kFTM_FallingEdge
    Capture on falling edge only

enumerator kFTM_RiseAndFallEdge
    Capture on rising or falling edge

enum _ftm_dual_edge_capture_mode

FlexTimer dual edge capture modes.

Values:

enumerator kFTM_OneShot

One-shot capture mode

enumerator kFTM_Continuous

Continuous capture mode

enum _ftm_quad_decode_mode

FlexTimer quadrature decode modes.

Values:

enumerator kFTM_QuadPhaseEncode

Phase A and Phase B encoding mode

enumerator kFTM_QuadCountAndDir

Count and direction encoding mode

enum _ftm_phase_polarity

FlexTimer quadrature phase polarities.

Values:

enumerator kFTM_QuadPhaseNormal

Phase input signal is not inverted

enumerator kFTM_QuadPhaseInvert

Phase input signal is inverted

enum _ftm_fault_output_state

FlexTimer pre-scaler factor for the dead time insertion.

Values:

enumerator kFTM_FaultOutput_PreDefined

FTM outputs will be placed into safe values when fault events in ongoing (defined by POL bits).

enumerator kFTM_FaultOutput_TriStated

FTM outputs will be tri-stated when fault event is ongoing.

enum _ftm_deadtime_prescale

FlexTimer pre-scaler factor for the dead time insertion.

Values:

enumerator kFTM_Deadtime_Prescale_1

Divide by 1

enumerator kFTM_Deadtime_Prescale_4

Divide by 4

enumerator kFTM_Deadtime_Prescale_16

Divide by 16

enum _ftm_clock_source

FlexTimer clock source selection.

Values:

enumerator kFTM_SystemClock
System clock selected

enumerator kFTM_FixedClock
Fixed frequency clock

enumerator kFTM_ExternalClock
External clock

enum _ftm_clock_prescale
FlexTimer pre-scaler factor selection for the clock source.

Values:

enumerator kFTM_Prescale_Divide_1
Divide by 1

enumerator kFTM_Prescale_Divide_2
Divide by 2

enumerator kFTM_Prescale_Divide_4
Divide by 4

enumerator kFTM_Prescale_Divide_8
Divide by 8

enumerator kFTM_Prescale_Divide_16
Divide by 16

enumerator kFTM_Prescale_Divide_32
Divide by 32

enumerator kFTM_Prescale_Divide_64
Divide by 64

enumerator kFTM_Prescale_Divide_128
Divide by 128

enum _ftm_filter_prescale
FlexTimer filter clock prescaler selection.

Values:

enumerator kFTM_Filter_Prescale_Divide_1
Divide by 1

enumerator kFTM_Filter_Prescale_Divide_2
Divide by 2

enumerator kFTM_Filter_Prescale_Divide_3
Divide by 3

enumerator kFTM_Filter_Prescale_Divide_4
Divide by 4

enumerator kFTM_Filter_Prescale_Divide_5
Divide by 5

enumerator kFTM_Filter_Prescale_Divide_6
Divide by 6

enumerator kFTM_Filter_Prescale_Divide_7
Divide by 7

enumerator kFTM_Filter_Prescale_Divide_8
Divide by 8

enumerator kFTM_Filter_Prescale_Divide_9
Divide by 9

enumerator kFTM_Filter_Prescale_Divide_10
Divide by 10

enumerator kFTM_Filter_Prescale_Divide_11
Divide by 11

enumerator kFTM_Filter_Prescale_Divide_12
Divide by 12

enumerator kFTM_Filter_Prescale_Divide_13
Divide by 13

enumerator kFTM_Filter_Prescale_Divide_14
Divide by 14

enumerator kFTM_Filter_Prescale_Divide_15
Divide by 15

enumerator kFTM_Filter_Prescale_Divide_16
Divide by 16

enum _ftm_bdm_mode
Options for the FlexTimer behaviour in BDM Mode.

Values:

enumerator kFTM_BdmMode_0
FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers

enumerator kFTM_BdmMode_1
FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers

enumerator kFTM_BdmMode_2
FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers

enumerator kFTM_BdmMode_3
FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode

enum _ftm_fault_mode
Options for the FTM fault control mode.

Values:

enumerator kFTM_Fault_Disable
Fault control is disabled for all channels

enumerator kFTM_Fault_EvenChnls
Enabled for even channels only(0,2,4,6) with manual fault clearing

enumerator kFTM_Fault_AllChnlsMan
Enabled for all channels with manual fault clearing

enumerator kFTM_Fault_AllChnlsAuto
: Enabled for all channels with automatic fault clearing

enum _ftm_external_trigger
: FTM external trigger options.

Note: Actual available external trigger sources are SoC-specific

Values:

enumerator kFTM_Chnl0Trigger
: Generate trigger when counter equals chnl 0 CnV reg

enumerator kFTM_Chnl1Trigger
: Generate trigger when counter equals chnl 1 CnV reg

enumerator kFTM_Chnl2Trigger
: Generate trigger when counter equals chnl 2 CnV reg

enumerator kFTM_Chnl3Trigger
: Generate trigger when counter equals chnl 3 CnV reg

enumerator kFTM_Chnl4Trigger
: Generate trigger when counter equals chnl 4 CnV reg

enumerator kFTM_Chnl5Trigger
: Generate trigger when counter equals chnl 5 CnV reg

enumerator kFTM_Chnl6Trigger
: Available on certain SoC's, generate trigger when counter equals chnl 6 CnV reg

enumerator kFTM_Chnl7Trigger
: Available on certain SoC's, generate trigger when counter equals chnl 7 CnV reg

enumerator kFTM_InitTrigger
: Generate Trigger when counter is updated with CNTIN

enumerator kFTM_ReloadInitTrigger
: Available on certain SoC's, trigger on reload point

enum _ftm_pwm_sync_method
: FlexTimer PWM sync options to update registers with buffer.

Values:

enumerator kFTM_SoftwareTrigger
: Software triggers PWM sync

enumerator kFTM_HardwareTrigger_0
: Hardware trigger 0 causes PWM sync

enumerator kFTM_HardwareTrigger_1
: Hardware trigger 1 causes PWM sync

enumerator kFTM_HardwareTrigger_2
: Hardware trigger 2 causes PWM sync

enum _ftm_reload_point
: FTM options available as loading point for register reload.

Note: Actual available reload points are SoC-specific

Values:

enumerator kFTM_Chnl0Match
: Channel 0 match included as a reload point

enumerator kFTM_Chnl1Match
: Channel 1 match included as a reload point

enumerator kFTM_Chnl2Match
: Channel 2 match included as a reload point

enumerator kFTM_Chnl3Match
: Channel 3 match included as a reload point

enumerator kFTM_Chnl4Match
: Channel 4 match included as a reload point

enumerator kFTM_Chnl5Match
: Channel 5 match included as a reload point

enumerator kFTM_Chnl6Match
: Channel 6 match included as a reload point

enumerator kFTM_Chnl7Match
: Channel 7 match included as a reload point

enumerator kFTM_CntMax
: Use in up-down count mode only, reload when counter reaches the maximum value

enumerator kFTM_CntMin
: Use in up-down count mode only, reload when counter reaches the minimum value

enumerator kFTM_HalfCycMatch
: Available on certain SoC's, half cycle match reload point

enum _ftm_interrupt_enable
: List of FTM interrupts.

Note: Actual available interrupts are SoC-specific

Values:

enumerator kFTM_Chnl0InterruptEnable
: Channel 0 interrupt

enumerator kFTM_Chnl1InterruptEnable
: Channel 1 interrupt

enumerator kFTM_Chnl2InterruptEnable
: Channel 2 interrupt

enumerator kFTM_Chnl3InterruptEnable
: Channel 3 interrupt

enumerator kFTM_Chnl4InterruptEnable
: Channel 4 interrupt

enumerator kFTM_Chnl5InterruptEnable
: Channel 5 interrupt

enumerator kFTM_Chnl6InterruptEnable
: Channel 6 interrupt

enumerator kFTM_Chnl7InterruptEnable
    Channel 7 interrupt

enumerator kFTM_FaultInterruptEnable
    Fault interrupt

enumerator kFTM_TimeOverflowInterruptEnable
    Time overflow interrupt

enumerator kFTM_ReloadInterruptEnable
    Reload interrupt; Available only on certain SoC's

enum _ftm_status_flags
List of FTM flags.

Note: Actual available flags are SoC-specific

Values:

enumerator kFTM_Chnl0Flag
    Channel 0 Flag

enumerator kFTM_Chnl1Flag
    Channel 1 Flag

enumerator kFTM_Chnl2Flag
    Channel 2 Flag

enumerator kFTM_Chnl3Flag
    Channel 3 Flag

enumerator kFTM_Chnl4Flag
    Channel 4 Flag

enumerator kFTM_Chnl5Flag
    Channel 5 Flag

enumerator kFTM_Chnl6Flag
    Channel 6 Flag

enumerator kFTM_Chnl7Flag
    Channel 7 Flag

enumerator kFTM_FaultFlag
    Fault Flag

enumerator kFTM_TimeOverflowFlag
    Time overflow Flag

enumerator kFTM_ChnlTriggerFlag
    Channel trigger Flag

enumerator kFTM_ReloadFlag
    Reload Flag; Available only on certain SoC's

enum _ftm_channel_index
List of FTM channel index used in logic OR.

Values:

enumerator kFTM_Chnl0_Mask
    Channel 0 Mask

enumerator kFTM_Chnl1_Mask
Channel 1 Mask

enumerator kFTM_Chnl2_Mask
Channel 2 Mask

enumerator kFTM_Chnl3_Mask
Channel 3 Mask

enumerator kFTM_Chnl4_Mask
Channel 4 Mask

enumerator kFTM_Chnl5_Mask
Channel 5 Mask

enumerator kFTM_Chnl6_Mask
Channel 6 Mask

enumerator kFTM_Chnl7_Mask
Channel 7 Mask

List of FTM Quad Decoder flags.

Values:

enumerator kFTM_QuadDecoderCountingIncreaseFlag
Counting direction is increasing (FTM counter increment), or the direction is decreasing.

enumerator kFTM_QuadDecoderCountingOverflowOnTopFlag
Indicates if the TOF bit was set on the top or the bottom of counting.

typedef enum *_ftm_chnl* ftm_chnl_t
List of FTM channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_ftm_fault_input* ftm_fault_input_t
List of FTM faults.

typedef enum *_ftm_pwm_mode* ftm_pwm_mode_t
FTM PWM operation modes.

typedef enum *_ftm_pwm_level_select* ftm_pwm_level_select_t
FTM PWM output pulse mode: high-true, low-true or no output.

Note: kFTM_NoPwmSignal: ELSnB:ELSnA = 0:0 kFTM_LowTrue: ELSnB:ELSnA = 0:1 EPWM: Channel n output is forced low at counter overflow, forced high at channel n match. CPWM: Channel n output is forced low at channel n match when counting down, and forced high at channel n match when counting up. Combined PWM: Channel n output is forced high at beginning of period and at channel n+1 match. It is forced low at the channel n match. kFTM_HighTrue: ELSnB:ELSnA = 1:0 EPWM: Channel n output is forced high at counter overflow, forced low at channel n match. CPWM: Channel n output is forced high at channel n match when counting down, and forced low at channel n match when counting up. Combined PWM: Channel n output is forced low at beginning of period and at channel n+1 match. It is forced high at the channel n match.

typedef struct *_ftm_chnl_pwm_signal_param* ftm_chnl_pwm_signal_param_t
: Options to configure a FTM channel's PWM signal.

typedef struct *_ftm_chnl_pwm_config_param* ftm_chnl_pwm_config_param_t
: Options to configure a FTM channel using precise setting.

typedef struct *_ftm_chnl_param* ftm_chnl_param_t
: General options to configure a FTM channel using precise setting.

typedef enum *_ftm_output_compare_mode* ftm_output_compare_mode_t
: FlexTimer output compare mode.

typedef enum *_ftm_input_capture_edge* ftm_input_capture_edge_t
: FlexTimer input capture edge.

typedef enum *_ftm_dual_edge_capture_mode* ftm_dual_edge_capture_mode_t
: FlexTimer dual edge capture modes.

typedef struct *_ftm_dual_edge_capture_param* ftm_dual_edge_capture_param_t
: FlexTimer dual edge capture parameters.

typedef enum *_ftm_quad_decode_mode* ftm_quad_decode_mode_t
: FlexTimer quadrature decode modes.

typedef enum *_ftm_phase_polarity* ftm_phase_polarity_t
: FlexTimer quadrature phase polarities.

typedef struct *_ftm_phase_param* ftm_phase_params_t
: FlexTimer quadrature decode phase parameters.

typedef struct *_ftm_fault_param* ftm_fault_param_t
: Structure is used to hold the parameters to configure a FTM fault.

typedef enum *_ftm_fault_output_state* ftm_fault_output_state_t
: FlexTimer pre-scaler factor for the dead time insertion.

typedef enum *_ftm_deadtime_prescale* ftm_deadtime_prescale_t
: FlexTimer pre-scaler factor for the dead time insertion.

typedef struct *_ftm_deadtime_param* ftm_deadtime_param_t
: Options to configure FTM combined channel pair deadtime.

typedef enum *_ftm_clock_source* ftm_clock_source_t
: FlexTimer clock source selection.

typedef enum *_ftm_clock_prescale* ftm_clock_prescale_t
: FlexTimer pre-scaler factor selection for the clock source.

typedef enum *_ftm_filter_prescale* ftm_filter_prescale_t
: FlexTimer filter clock prescaler selection.

typedef enum *_ftm_bdm_mode* ftm_bdm_mode_t
: Options for the FlexTimer behaviour in BDM Mode.

typedef enum *_ftm_fault_mode* ftm_fault_mode_t
: Options for the FTM fault control mode.

typedef enum *_ftm_external_trigger* ftm_external_trigger_t
: FTM external trigger options.

Note: Actual available external trigger sources are SoC-specific typedef enum *_ftm_pwm_sync_method* ftm_pwm_sync_method_t

FlexTimer PWM sync options to update registers with buffer.

typedef enum *_ftm_reload_point* ftm_reload_point_t

FTM options available as loading point for register reload.

Note: Actual available reload points are SoC-specific

typedef enum *_ftm_interrupt_enable* ftm_interrupt_enable_t

List of FTM interrupts.

Note: Actual available interrupts are SoC-specific

typedef enum *_ftm_status_flags* ftm_status_flags_t

List of FTM flags.

Note: Actual available flags are SoC-specific

typedef enum *_ftm_channel_index* ftm_channel_index_t

List of FTM channel index used in logic OR.

typedef struct *_ftm_config* ftm_config_t

FTM configuration structure.

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

The configuration structure can be made constant so as to reside in flash.

void FTM_SetupFaultInput(FTM_Type *base, *ftm_fault_input_t* faultNumber, const *ftm_fault_param_t* *faultParams)

Sets up the working of the FTM fault inputs protection.

FTM can have up to 4 fault inputs. This function sets up fault parameters, fault level, and input filter.

Parameters

- base – FTM peripheral base address
- faultNumber – FTM fault to configure.
- faultParams – Parameters passed in to set up the fault

static inline void FTM_SetGlobalTimeBaseOutputEnable(FTM_Type *base, bool enable)

Enables or disables the FTM global time base signal generation to other FTMs.

Parameters

- base – FTM peripheral base address
- enable – true to enable, false to disable

static inline void FTM_SetOutputMask(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool mask)

Sets the FTM peripheral timer channel output mask.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be configured

- mask – true: masked, channel is forced to its inactive state; false: unmasked

static inline void FTM_SetPwmOutputEnable(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool value)

Allows users to enable an output on an FTM channel.

To enable the PWM channel output call this function with val=true. For input mode, call this function with val=false.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be configured
- value – true: enable output; false: output is disabled, used in input mode

static inline void FTM_SetSoftwareTrigger(FTM_Type *base, bool enable)

Enables or disables the FTM software trigger for PWM synchronization.

Parameters

- base – FTM peripheral base address
- enable – true: software trigger is selected, false: software trigger is not selected

static inline void FTM_SetWriteProtection(FTM_Type *base, bool enable)

Enables or disables the FTM write protection.

Parameters

- base – FTM peripheral base address
- enable – true: Write-protection is enabled, false: Write-protection is disabled

static inline void FTM_EnableDmaTransfer(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool enable)

Enable DMA transfer or not.

Note: CHnIE bit needs to be set when calling this API. The channel DMA transfer request is generated and the channel interrupt is not generated if (CHnF = 1) when DMA and CHnIE bits are set.

Parameters

- base – FTM peripheral base address.
- chnlNumber – Channel to be configured
- enable – true to enable, false to disable

static inline void FTM_SetLdok(FTM_Type *base, bool value)

Enable the LDOK bit.

This function enables loading updated values.

Parameters

- base – FTM peripheral base address
- value – true: loading updated values is enabled; false: loading updated values is disabled.

static inline void FTM_SetHalfCycReloadMatchValue(FTM_Type *base, uint32_t ticks)

Sets the half cycle relade period in units of ticks.

This function can be callled to set the half-cycle reload value when half-cycle matching is enabled as a reload point. Note: Need enable kFTM_HalfCycMatch as reload point, and when this API call after FTM_StartTimer(), the new HCR value will not be active until next reload point (need call FTM_SetLdok to set LDOK) or register synchronization.

Parameters

- base – FTM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

static inline void FTM_SetLoadFreq(FTM_Type *base, uint32_t loadfreq)

Set load frequency value.

Parameters

- base – FTM peripheral base address.
- loadfreq – PWM reload frequency, range: 0 ~ 31.

static inline void FTM_SetPairDeadTime(FTM_Type *base, const *ftm_deadtime_param_t* *config, *ftm_chnl_t* chnlPairNumber)

brief Configure deadtime for specific combined channel pair.

param base FTM peripheral base address param config Pointer to the user configuration structure. param chnlPairNumber The FTM channel pair number; options are 0, 1, 2, 3

static inline void FTM_SetPeriodDithering(FTM_Type *base, uint16_t moduloValue, uint8_t fractionalValue)

Set PWM Period Dithering. For the PWM period dithering, the register MOD_MIRROR should be used instead of the register MOD.

Parameters

- base – FTM peripheral base address.
- moduloValue – FTM counter modulo value.
- fractionalValue – The modulo fractional value used in the PWM period dithering.

static inline void FTM_SetEdgeDithering(FTM_Type *base, *ftm_chnl_t* chnlNumber, uint16_t matchValue, uint8_t fractionalValue)

Set PWM Edge Dithering. For the PWM edge dithering, the register CnV_MIRROR should be used instead of the register CnV.

Parameters

- base – FTM peripheral base address.
- chnlNumber – The channel number.
- matchValue – FTM channel n match value.
- fractionalValue – The channel n match fractional value used in the PWM edge dithering.

static inline uint32_t FTM_GetChannelInputState(FTM_Type *base, *ftm_chnl_t* chnlNumber)

Get value of channel n input after the double-sampling or the filtering.

Parameters

- base – FTM peripheral base address.
- chnlNumber – The channel number.

Returns
Channel n input state, 0 or 1.

static inline uint32_t FTM_GetChannelOutputState(FTM_Type *base, *ftm_chnl_t* chnlNumber)

Get final value of the channel n output.

Parameters

- base – FTM peripheral base address.
- chnlNumber – The channel number.

Returns
Channel n output value, 0 or 1.

struct _ftm_chnl_pwm_signal_param

#include <fsl_ftm.h> Options to configure a FTM channel's PWM signal.

Public Members

ftm_chnl_t chnlNumber

The channel/channel pair number. In combined mode, this represents the channel pair number.

ftm_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 kFTM_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure leave as 0; Should be specified as a percentage of the PWM period

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

bool enableDeadtime

Used only in combined PWM mode with enable complementary. true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

struct _ftm_chnl_pwm_config_param

#include <fsl_ftm.h> Options to configure a FTM channel using precise setting.

Public Members

ftm_chnl_t chnlNumber

The channel/channel pair number. In combined mode, this represents the channel pair number.

ftm_pwm_level_select_t level

PWM output active level select.

uint16_t dutyValue

PWM pulse width, the uint of this value is timer ticks.

uint16_t firstEdgeValue

Used only in kFTM_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure leave as 0, uint of this value is timer ticks.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

bool enableDeadtime

Used only in combined PWM mode with enable complementary. true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

struct _ftm_chnl_param

#include <fsl_ftm.h> General options to configure a FTM channel using precise setting.

Public Members

ftm_pwm_mode_t mode

PWM output mode.

ftm_pwm_level_select_t level

PWM output active level select.

uint16_t initialValue

FTM counter initial value.

uint16_t moduloValue

FTM counter modulo value.

uint16_t chnlValue

FTM channel n match value.

uint16_t combinedChnlValue

FTM combined channel n+1 match value, used only in (modified) combined PWM mode.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

bool enableDeadtime

Used only in combined PWM mode with enable complementary. true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

bool enablePulseOutput

Used only in Edge-aligned PWM and Center-aligned PWM. true: If a match in channel occurs, a trigger pulse with one FTM input clock width is generated in the channel n; false: Channel outputs will generate normal PWM outputs without generating a pulse.

bool enableDithering

Enable fractional delay to achieve fine resolution on generated PWM signals. true: Enable dithering; false: Disable dithering.

uint8_t moduloFracValue

Modulo fractional value, used in Period Dithering.

uint8_t chnlFracValue

Channel n match fractional value, used in Edge Dithering.

uint8_t combinedChnlFracValue

Combined channel n+1 match fractional value, used in Edge Dithering. It is recommended to use only one PWM Edge Dithering (channel n PWM Edge Dithering or channel n+1 PWM Edge Dithering) at a time.

struct _ftm_dual_edge_capture_param

#include <fsl_ftm.h> FlexTimer dual edge capture parameters.

Public Members

ftm_dual_edge_capture_mode_t mode

Dual Edge Capture mode

ftm_input_capture_edge_t currChanEdgeMode

Input capture edge select for channel n

ftm_input_capture_edge_t nextChanEdgeMode

Input capture edge select for channel n+1

struct _ftm_phase_param

#include <fsl_ftm.h> FlexTimer quadrature decode phase parameters.

Public Members

bool enablePhaseFilter

True: enable phase filter; false: disable filter

uint32_t phaseFilterVal

Filter value, used only if phase filter is enabled

ftm_phase_polarity_t phasePolarity

Phase polarity

struct _ftm_fault_param

#include <fsl_ftm.h> Structure is used to hold the parameters to configure a FTM fault.

Public Members

bool enableFaultInput

True: Fault input is enabled; false: Fault input is disabled

bool faultLevel

True: Fault polarity is active low; in other words, '0' indicates a fault; False: Fault polarity is active high

bool useFaultFilter

True: Use the filtered fault signal; False: Use the direct path from fault input

struct _ftm_deadtime_param

#include <fsl_ftm.h> Options to configure FTM combined channel pair deadtime.

Public Members

ftm_deadtime_prescale_t deadTimePrescale

The dead time prescalar value

uint32_t deadTimeValue

The dead time value deadTimeValue's available range is 0-1023 when register has DT-VALEX, otherwise its available range is 0-63.

struct _ftm_config

#include <fsl_ftm.h> FTM configuration structure.

This structure holds the configuration settings for the FTM peripheral. To initialize this structure to reasonable defaults, call the FTM_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

ftm_clock_prescale_t prescale

FTM clock prescale value

ftm_filter_prescale_t filterPrescale

Clock prescaler used in FTM filters

ftm_bdm_mode_t bdmMode

FTM behavior in BDM mode

uint32_t pwmSyncMode

Synchronization methods to use to update buffered registers; Multiple update modes can be used by providing an OR'ed list of options available in enumeration ftm_pwm_sync_method_t.

uint32_t reloadPoints

FTM reload points; When using this, the PWM synchronization is not required. Multiple reload points can be used by providing an OR'ed list of options available in enumeration ftm_reload_point_t.

ftm_fault_mode_t faultMode

FTM fault control mode

uint8_t faultFilterValue

Fault input filter value

ftm_fault_output_state_t faultOutputState

Fault output state

ftm_deadtime_prescale_t deadTimePrescale

The dead time prescalar value

uint32_t deadTimeValue

The dead time value deadTimeValue's available range is 0-1023 when register has DT-VALEX, otherwise its available range is 0-63.

uint32_t extTriggers

External triggers to enable. Multiple trigger sources can be enabled by providing an OR'ed list of options available in enumeration ftm_external_trigger_t.

uint8_t chnlInitState

Defines the initialization value of the channels in OUTINT register

uint8_t chnlPolarity
: Defines the output polarity of the channels in POL register

bool useGlobalTimeBase
: True: Use of an external global time base is enabled; False: disabled

bool swTriggerResetCount
: FTM counter synchronization activated by software trigger, avtive when (syncMethod & FTM_SYNC_SWSYNC_MASK) != 0U

bool hwTriggerResetCount
: FTM counter synchronization activated by hardware trigger, avtive when (syncMethod & (FTM_SYNC_TRIG0_MASK | FTM_SYNC_TRIG1_MASK | FTM_SYNC_TRIG2_MASK)) != 0U

2.16 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

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().

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

2.17 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

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

2.18 Common Driver

FSL_COMMON_DRIVER_VERSION

common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE

No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART

Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART

Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI

Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
    Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
    Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART
    Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART
    Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART
    Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO
    Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI
    Debug console based on QSCI.

MIN(a, b)
    Computes the minimum of *a* and *b*.

MAX(a, b)
    Computes the maximum of *a* and *b*.

UINT16_MAX
    Max value of uint16_t type.

UINT32_MAX
    Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(addr, val)
    Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)
    Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)
    Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)
    Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)
    Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)
    For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)
    For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)
    For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)
    Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)
: Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)
: Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)
: Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)
: Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)
: Macro to define a variable with L1 d-cache line size alignment

  Macro to define a variable with L2 cache line size alignment

  Macro to change a value to a given size aligned value (rounded up)

SDK_SIZEALIGN_UP(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK_SIZEALIGN

SDK_SIZEALIGN_DOWN(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded down)

SDK_IS_ALIGNED(var, alignbytes)
: Macro to check if a value is aligned to a given size

AT_NONCACHEABLE_SECTION(var)
: Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)
: Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_INIT(var)
: Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)
: Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_CACHE_LINE_SECTION(var)
: Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)
: Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

AT_QUICKACCESS_SECTION_CODE(func)
: Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)
: Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)
: Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

MCUX_RAMFUNC

Function attribute to place function in RAM. For example, to place function my_func in ram, use like:

```
MCUX_RAMFUNC my_func
```

RAMFUNCTION_SECTION_CODE(func)

Place function in ram.

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI

Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART

Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C

Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C

Group number for LPI2C status codes.

enumerator kStatusGroup_UART

Group number for UART status codes.

enumerator kStatusGroup_I2C

Group number for UART status codes.

enumerator kStatusGroup_LPSCI

Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART

Group number for LPUART status codes.

enumerator kStatusGroup_SPI

Group number for SPI status code.

enumerator kStatusGroup_XRDC

Group number for XRDC status code.

enumerator kStatusGroup_SEMA42

Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC

Group number for SDHC status code

enumerator kStatusGroup_SDMMC
    Group number for SDMMC status code

enumerator kStatusGroup_SAI
    Group number for SAI status code

enumerator kStatusGroup_MCG
    Group number for MCG status codes.

enumerator kStatusGroup_SCG
    Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
    Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
    Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
    Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
    Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
    Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
    Group number for I2S status codes

enumerator kStatusGroup_IUART
    Group number for IUART status codes

enumerator kStatusGroup_CSI
    Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
    Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
    Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
    Group number for POWER status codes.

enumerator kStatusGroup_ENET
    Group number for ENET status codes.

enumerator kStatusGroup_PHY
    Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
    Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
    Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
    Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
    Group number for QSPI status codes.

enumerator kStatusGroup_DMA
    Group number for DMA status codes.

enumerator kStatusGroup_EDMA
    Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
    Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
    Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
    Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
    Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
    Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
    Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
    Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
    Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
    Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
    Group number for OTP status codes.

enumerator kStatusGroup_MCAN
    Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
    Group number for CAAM status codes.

enumerator kStatusGroup_ECSPI
    Group number for ECSPI status codes.

enumerator kStatusGroup_USDHC
    Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
    Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
    Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
    Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
    Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
    Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
    Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
    Group number for Common task status codes.

enumerator kStatusGroup_MSG
    Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
    Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
    Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
    Group number for codec status codes.

enumerator kStatusGroup_ASRC
    Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
    Group number for codec status codes.

enumerator kStatusGroup_SDIOSLV
    Group number for SDIOSLV status codes.

enumerator kStatusGroup_MECC
    Group number for MECC status codes.

enumerator kStatusGroup_ENET_QOS
    Group number for ENET_QOS status codes.

enumerator kStatusGroup_LOG
    Group number for LOG status codes.

enumerator kStatusGroup_I3CBUS
    Group number for I3CBUS status codes.

enumerator kStatusGroup_QSCI
    Group number for QSCI status codes.

enumerator kStatusGroup_ELEMU
    Group number for ELEMU status codes.

enumerator kStatusGroup_QUEUEDSPI
    Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
    Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
    Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
    Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
    Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
    Group number for HOSTIF status codes.

enumerator kStatusGroup_CLIF
: Group number for CLIF status codes.

enumerator kStatusGroup_BMA
: Group number for BMA status codes.

enumerator kStatusGroup_NETC
: Group number for NETC status codes.

enumerator kStatusGroup_ELE
: Group number for ELE status codes.

enumerator kStatusGroup_GLIKEY
: Group number for GLIKEY status codes.

enumerator kStatusGroup_AON_POWER
: Group number for AON_POWER status codes.

enumerator kStatusGroup_AON_COMMON
: Group number for AON_COMMON status codes.

enumerator kStatusGroup_ENDAT3
: Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
: Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
: Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
: Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
: Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
: Group number for A-format status codes.

enumerator kStatusGroup_LPC_QSPI
: Group number for LPC QSPI status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
: Generic status for Success.

enumerator kStatus_Fail
: Generic status for Fail.

enumerator kStatus_ReadOnly
: Generic status for read only failure.

enumerator kStatus_OutOfRange
: Generic status for out of range access.

enumerator kStatus_InvalidArgument
: Generic status for invalid argument check.

enumerator kStatus_Timeout

Generic status for timeout.

enumerator kStatus_NoTransferInProgress

Generic status for no transfer in progress.

enumerator kStatus_Busy

Generic status for module is busy.

enumerator kStatus_NoData

Generic status for no data is found for the operation.

typedef int32_t status_t

Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)

Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values

The – allocated memory.

void SDK_Free(void *ptr)

Free memory.

Parameters

- ptr – The memory to be release.

void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- delayTime_us – Delay time in unit of microsecond.
- coreClock_Hz – Core clock frequency with Hz.

static inline *status_t* EnableIRQ(IRQn_Type interrupt)

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt enabled successfully
- kStatus_Fail – Failed to enable the interrupt

static inline *status_t* DisableIRQ(IRQn_Type interrupt)

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt disabled successfully
- kStatus_Fail – Failed to disable the interrupt

static inline *status_t* EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

Returns

Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

static inline bool _SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t newValue)

static inline uint32_t _SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

```
| Unused    || Major Version || Minor Version ||  Bug Fix    |
31        25  24           17  16            9  8            0
```

ARRAY_SIZE(x)

Computes the number of elements in an array.

UINT64_H(X)
: Macro to get upper 32 bits of a 64-bit value

UINT64_L(X)
: Macro to get lower 32 bits of a 64-bit value

SUPPRESS_FALL_THROUGH_WARNING()
: For switch case code block, if case section ends without "break;" statement, there wil be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, "SUPPRESS_FALL_THROUGH_WARNING();" need to be added at the end of each case section which misses "break;"statement.

MSDK_REG_SECURE_ADDR(x)
: Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)
: Convert the register address to the one used in non-secure mode.

MSDK_HAS_DWT_CYCCNT
: The chip supports DWT CYCCNT or not.

MSDK_INVALID_IRQ_HANDLER
: Invalid IRQ handler address.

2.19 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.

2.20 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)
Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster            = true;
masterConfig->debugEnable             = false;
masterConfig->ignoreAck               = false;
masterConfig->pinConfig               = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz             = 100000U;
masterConfig->busIdleTimeout_ns       = 0;
masterConfig->pinLowTimeout_ns        = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable      = false;
masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to lpi2c_master_config_t.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

Convert provided flags to status code, and clear any errors if present.

Parameters

- base – The LPI2C peripheral base address.
- status – Current status flags value that will be checked.

Return values

- kStatus_Success –
- kStatus_LPI2C_PinLowTimeout –
- kStatus_LPI2C_ArbitrationLost –
- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag

- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags().

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_LPI2C_Nak.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
: Errors to check for.

enum _lpi2c_direction
: Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write
: Master transmit.

enumerator kLPI2C_Read
: Master receive.

enum _lpi2c_master_pin_config
: LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain
: LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
: LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
: LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
: LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
: LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
: LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
: LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
: LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
: LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin
: Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
: Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
: LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow
: Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh

Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode

LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled

LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1

LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1

LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1

LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1

LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1

LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1

LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Values:

enumerator kLPI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag

Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t

Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t

LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t

LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t

LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t

LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t

LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

LPI2C master descriptor of the transfer.

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

LPI2C master handle of the transfer.

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base
The LPI2C peripheral base address.

Param handle
Pointer to the LPI2C master driver handle.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct _lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable
: Enable transfers to continue when halted in debug mode.

bool ignoreAck
: Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig
: The pin configuration option.

uint32_t baudRate_Hz
: Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns
: Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns
: Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest
: Host request options.

struct _lpi2c_match_config

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
: Data match configuration setting.

bool rxDataMatchOnly
: When set to true, received data is ignored until a successful match.

uint32_t match0
: Match value 0.

uint32_t match1
: Match value 1.

struct _lpi2c_master_transfer

#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
: Bit mask of options for the transfer. See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefaultFlag for normal transfers.

uint16_t slaveAddress
: The 7-bit slave address.

lpi2c_direction_t direction
: Either kLPI2C_Read or kLPI2C_Write.

uint32_t subaddress
: Sub address. Transferred MSB first.

size_t subaddressSize
: Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
: Pointer to data to transfer.

size_t dataSize
: Number of bytes to transfer.

struct __lpi2c__master__handle

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
: Transfer state machine current state.

uint16_t remainingBytes
: Remaining byte count in current state.

uint8_t *buf
: Buffer pointer for current state.

uint16_t commandBuffer[6]
: LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer
: Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback
: Callback function pointer.

void *userData
: Application data passed to callback.

struct hostRequest

Public Members

bool enable
: Enable host request.

lpi2c_host_request_source_t source
: Host request source.

lpi2c_host_request_polarity_t polarity
: Host request pin polarity.

2.21 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 enableRxDataS-CLStall 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.

2.22 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

2.23 LPSPI: Low Power Serial Peripheral Interface

2.24 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure lpspi_master_config_t.
- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the lpspi_master_config_t structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t  masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to lpspi_master_config_t structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi_slave_config_t.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the lpspi_slave_config_t structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t  slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to lpspi_slave_config_t structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns
The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type _lpspi_flags.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

bool LPSPI_CheckTransferArgument(LPSPI_Type *base, *lpspi_transfer_t* *transfer, bool isEdma)

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, *lpspi_master_slave_mode_t* mode)

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, *lpspi_which_pcs_t* select)

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.

- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI_SetPCSContinous(LPSPI_Type *base, bool IsContinous)

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, *lpspi_delay_type_t* whichDelay)

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.

uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, *lpspi_delay_type_t* whichDelay, uint32_t srcClock_Hz)

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler).

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nano-seconds.
- whichDelay – The desired delay to configuration, which must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.
- data – The data word to be sent.

static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The data read from the data buffer.

void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_master_transfer_callback_t* callback, void *userData)

Initializes the LPSPI master handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_master_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns

true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag
Word Complete flag

enumerator kLPSPI_FrameCompleteFlag
Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag
Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag
Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag
Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag
Data Match flag

enumerator kLPSPI_ModuleBusyFlag
Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable
LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable
Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0
PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity
LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity
LPSPI clock polarity configuration.

Values:

enumerator kLPSPI_ClockPolarityActiveHigh
CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow
CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase
LPSPI clock phase configuration.

Values:

enumerator kLPSPI_ClockPhaseFirstEdge
CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge
CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction
LPSPI data shifter direction options.

Values:

enumerator kLPSPI_MsbFirst
Data transfers start with most significant bit.

enumerator kLPSPI_LsbFirst
Data transfers start with least significant bit.

enum _lpspi_host_request_select
LPSPI Host Request select configuration.

Values:

enumerator kLPSPI_HostReqExtPin
Host Request is an ext pin.

enumerator kLPSPI_HostReqInternalTrigger
Host Request is an internal trigger.

enum _lpspi_match_config
LPSPI Match configuration options.

Values:

enumerator kLPSI_MatchDisabled
LPSPI Match Disabled.

enumerator kLPSI_1stWordEqualsM0orM1
LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0orM1
LPSPI Match Enabled.

enumerator kLPSI_1stWordEqualsM0and2ndWordEqualsM1
LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0andNxtWordEqualsM1
LPSPI Match Enabled.

enumerator kLPSI_1stWordAndM1EqualsM0andM1
: LPSPI Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1
: LPSPI Match Enabled.

enum _lpspi_pin_config
: LPSPI pin (SDO and SDI) configuration.

Values:

enumerator kLPSPI_SdiInSdoOut
: LPSPI SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut
: LPSPI SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut
: LPSPI SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut
: LPSPI SDO input, SDI output.

enum _lpspi_data_out_config
: LPSPI data output configuration.

Values:

enumerator kLpspiDataOutRetained
: Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate
: Data out is tristated when chip select is de-asserted

enum _lpspi_transfer_width
: LPSPI transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
: 1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
: 2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
: 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
: LPSPI delay type selection.

Values:

enumerator kLPSPI_PcsToSck
: PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
: Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
: Delay between transfers.

enum _lpspi_transfer_config_flag_for_master
Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterPcsContinuous
Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap
Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave
Use this enumeration for LPSPI slave transfer configFlags.

Values:

enumerator kLPSPI_SlavePcs0
LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS0 signal

enumerator kLPSPI_SlavePcs1
LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS1 signal

enumerator kLPSPI_SlavePcs2
LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS2 signal

enumerator kLPSPI_SlavePcs3
LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS3 signal

enumerator kLPSPI_SlaveByteSwap
Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

enum _lpspi_transfer_state
: LPSPI transfer state, which is used for LPSPI transactional API state machine.

    Values:

    enumerator kLPSPI_Idle
    : Nothing in the transmitter/receiver.

    enumerator kLPSPI_Busy
    : Transfer queue is not finished.

    enumerator kLPSPI_Error
    : Transfer error.

typedef enum *_lpspi_master_slave_mode* lpspi_master_slave_mode_t
: LPSPI master or slave mode configuration.

typedef enum *_lpspi_which_pcs_config* lpspi_which_pcs_t
: LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum *_lpspi_pcs_polarity_config* lpspi_pcs_polarity_config_t
: LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *_lpspi_clock_polarity* lpspi_clock_polarity_t
: LPSPI clock polarity configuration.

typedef enum *_lpspi_clock_phase* lpspi_clock_phase_t
: LPSPI clock phase configuration.

typedef enum *_lpspi_shift_direction* lpspi_shift_direction_t
: LPSPI data shifter direction options.

typedef enum *_lpspi_host_request_select* lpspi_host_request_select_t
: LPSPI Host Request select configuration.

typedef enum *_lpspi_match_config* lpspi_match_config_t
: LPSPI Match configuration options.

typedef enum *_lpspi_pin_config* lpspi_pin_config_t
: LPSPI pin (SDO and SDI) configuration.

typedef enum *_lpspi_data_out_config* lpspi_data_out_config_t
: LPSPI data output configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t
: LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t
: LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t
: LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t
: LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t
: Forward declaration of the _lpspi_master_handle typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t
: Forward declaration of the _lpspi_slave_handle typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)

Master completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)

Slave completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_lpspi_transfer* lpspi_transfer_t

LPSPI master/slave transfer structure.

volatile uint8_t g_lpspiDummyData[]

Global variable for dummy data value setting.

LPSPI_DUMMY_DATA

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

SPI_RETRY_TIMES

Retry times for waiting flag.

LPSPI_MASTER_PCS_SHIFT

LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_PCS_MASK

LPSPI master PCS shift macro , internal used.

LPSPI_SLAVE_PCS_SHIFT

LPSPI slave PCS shift macro , internal used.

LPSPI_SLAVE_PCS_MASK

LPSPI slave PCS shift macro , internal used.

struct _lpspi_master_config

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate
    Baud Rate for LPSPI.

uint32_t bitsPerFrame
    Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
    Clock polarity.

lpspi_clock_phase_t cpha
    Clock phase.

lpspi_shift_direction_t direction
    MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec
    PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec
    Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec
    After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs
    Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
    Desired PCS active high or low

lpspi_pin_config_t pinCfg
    Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
    Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay
    Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct _lpspi_slave_config
    #include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame
    Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
    Clock polarity.

lpspi_clock_phase_t cpha
    Clock phase.

lpspi_shift_direction_t direction
    MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs
: Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
: Desired PCS active high or low

lpspi_pin_config_t pinCfg
: Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
: Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct _lpspi_transfer
: *#include <fsl_lpspi.h>* LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData
: Send buffer.

uint8_t *rxData
: Receive buffer.

volatile size_t dataSize
: Transfer bytes.

uint32_t configFlags
: Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

struct _lpspi_master_handle
: *#include <fsl_lpspi.h>* LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
: Is PCS continuous in transfer.

volatile bool writeTcrInIsr
: A flag that whether should write TCR in ISR.

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile bool isTxMask
: A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame
: Number of bytes in each frame

volatile uint16_t frameSize
: Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

uint32_t txBuffIfNull
: Used if the txData is NULL.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

struct _lpspi_slave_handle
: *#include <fsl_lpspi.h>* LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
: Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

2.25 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;
```

(continues on next page)

(continued from previous page)

```
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:

enumerator kLPTMR_PrescalerClock_0
Prescaler/glitch filter clock 0 selected.

enumerator kLPTMR_PrescalerClock_1
Prescaler/glitch filter clock 1 selected.

enumerator kLPTMR_PrescalerClock_2
Prescaler/glitch filter clock 2 selected.

enumerator kLPTMR_PrescalerClock_3
Prescaler/glitch filter clock 3 selected.

enum _lptmr_interrupt_enable
List of the LPTMR interrupts.

Values:

enumerator kLPTMR_TimerInterruptEnable
Timer interrupt enable

enum _lptmr_status_flags
List of the LPTMR status flags.

Values:

enumerator kLPTMR_TimerCompareFlag
Timer compare flag

typedef enum *_lptmr_pin_select* lptmr_pin_select_t
LPTMR pin selection used in pulse counter mode.

typedef enum *_lptmr_pin_polarity* lptmr_pin_polarity_t
LPTMR pin polarity used in pulse counter mode.

typedef enum *_lptmr_timer_mode* lptmr_timer_mode_t
LPTMR timer mode selection.

typedef enum *_lptmr_prescaler_glitch_value* lptmr_prescaler_glitch_value_t
LPTMR prescaler/glitch filter values.

typedef enum *_lptmr_prescaler_clock_select* lptmr_prescaler_clock_select_t

LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

typedef enum *_lptmr_interrupt_enable* lptmr_interrupt_enable_t

List of the LPTMR interrupts.

typedef enum *_lptmr_status_flags* lptmr_status_flags_t

List of the LPTMR status flags.

typedef struct *_lptmr_config* lptmr_config_t

LPTMR config structure.

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

The configuration struct can be made constant so it resides in flash.

static inline void LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)

Enable or disable timer DMA request.

Parameters

- base – base LPTMR peripheral base address
- enable – Switcher of timer DMA feature. "true" means to enable, "false" means to disable.

struct _lptmr_config

#include <fsl_lptmr.h> LPTMR config structure.

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

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode

Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler

True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource

LPTMR clock source

lptmr_prescaler_glitch_value_t value

Prescaler or glitch filter value

2.26 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.27 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_idle_type_select
LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag
Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag
: The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag
: The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag
: TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag
: RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag
: TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag
: RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
: LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
: LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
: LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
: LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
: LPUART transmit CTS configure.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
: LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
: LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *_lpuart_config* lpuart_config_t
: LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
: LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
: LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_config
#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
RX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

uint8_t rtsWatermark
RTS watermark

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

bool enableTx
Enable TX

bool enableRx
Enable RX

bool swapTxdRxd
Swap TXD and RXD pins

struct _lpuart_transfer
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
: The byte count to be transfer.

struct _lpuart_handle
: *#include <fsl_lpuart.h>* LPUART handle structure.

Public Members

volatile size_t txDataSize
: Size of the remaining data to send.

size_t txDataSizeAll
: Size of the data to send out.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t rxDataSizeAll
: Size of the data to receive.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
: Callback function.

void *userData
: LPUART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state.

bool isSevenDataBits
: Seven data bits flag.

bool is16bitData
: 16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed21___

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 ___unnamed23___

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 ___unnamed25___

Public Members

uint8_t *volatile rxData

Address of remaining data to receive.

uint16_t *volatile rxData16

Address of remaining data to receive.

union ___unnamed27___

Public Members

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16

Start address of the receiver ring buffer.

2.28 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.

2.29 MMDVSQ: Memory-Mapped Divide and Square Root

int32_t MMDVSQ_GetDivideRemainder(MMDVSQ_Type *base, int32_t dividend, int32_t divisor, bool isUnsigned)

Performs the MMDVSQ division operation and returns the remainder.

Parameters

- base – MMDVSQ peripheral address
- dividend – Dividend value
- divisor – Divisor value
- isUnsigned – Mode of unsigned divide
  - true unsigned divide
  - false signed divide

int32_t MMDVSQ_GetDivideQuotient(MMDVSQ_Type *base, int32_t dividend, int32_t divisor, bool isUnsigned)

Performs the MMDVSQ division operation and returns the quotient.

Parameters

- base – MMDVSQ peripheral address
- dividend – Dividend value
- divisor – Divisor value
- isUnsigned – Mode of unsigned divide
  - true unsigned divide
  - false signed divide

uint16_t MMDVSQ_Sqrt(MMDVSQ_Type *base, uint32_t radicand)

Performs the MMDVSQ square root operation.

This function performs the MMDVSQ square root operation and returns the square root result of a given radicand value.

Parameters

- base – MMDVSQ peripheral address
- radicand – Radicand value

static inline *mmdvsq_execution_status_t* MMDVSQ_GetExecutionStatus(MMDVSQ_Type *base)

Gets the MMDVSQ execution status.

This function checks the current MMDVSQ execution status of the combined CSR[BUSY, DIV, SQRT] indicators.

Parameters

- base – MMDVSQ peripheral address

Returns

Current MMDVSQ execution status

static inline void MMDVSQ_SetFastStartConfig(MMDVSQ_Type *base, *mmdvsq_fast_start_select_t* mode)

Configures MMDVSQ fast start mode.

This function sets the MMDVSQ division fast start. The MMDVSQ supports two mechanisms for initiating a division operation. The default mechanism is a “fast start” where a write to the DSOR register begins the division. Alternatively, the start mechanism can begin after a write to the CSR register with CSR[SRT] set.

Parameters

- base – MMDVSQ peripheral address
- mode – Mode of Divide-Fast-Start
  - kMmdvsqDivideFastStart = 0
  - kMmdvsqDivideNormalStart = 1

static inline void MMDVSQ_SetDivideByZeroConfig(MMDVSQ_Type *base, bool isDivByZero)

Configures the MMDVSQ divide-by-zero mode.

This function configures the MMDVSQ response to divide-by-zero calculations. If both CSR[DZ] and CSR[DZE] are set, then a subsequent read of the RES register is error-terminated to signal the processor of the attempted divide-by-zero. Otherwise, the register contents are returned.

Parameters

- base – MMDVSQ peripheral address
- isDivByZero – Mode of Divide-By-Zero
  - kMmdvsqDivideByZeroDis = 0
  - kMmdvsqDivideByZeroEn = 1

FSL_MMSVSQ_DRIVER_VERSION

Version 2.0.4.

enum _mmdvsq_execution_status

MMDVSQ execution status.

Values:

enumerator kMMDVSQ_IdleSquareRoot

MMDVSQ is idle; the last calculation was a square root

enumerator kMMDVSQ_IdleDivide

MMDVSQ is idle; the last calculation was division

enumerator kMMDVSQ_BusySquareRoot

MMDVSQ is busy processing a square root calculation

enumerator kMMDVSQ_BusyDivide

MMDVSQ is busy processing a division calculation

enum _mmdvsq_fast_start_select

MMDVSQ divide fast start select.

Values:

enumerator kMMDVSQ_EnableFastStart

Division operation is initiated by a write to the DSOR register

enumerator kMMDVSQ_DisableFastStart

Division operation is initiated by a write to CSR[SRT] = 1; normal start instead fast start

typedef enum *_mmdvsq_execution_status* mmdvsq_execution_status_t

MMDVSQ execution status.

typedef enum *_mmdvsq_fast_start_select* mmdvsq_fast_start_select_t

MMDVSQ divide fast start select.

2.30 MSCAN: Scalable Controller Area Network

2.31 MSCAN Driver

void MSCAN_Init(MSCAN_Type *base, const *mscan_config_t* *config, uint32_t sourceClock_Hz)

Initializes a MsCAN instance.

This function initializes the MsCAN module with user-defined settings. This example shows how to set up the mscan_config_t parameters and how to call the MSCAN_Init function by passing in these parameters.

```
mscan_config_t mscanConfig;
mscanConfig.clkSrc              = kMSCAN_ClkSrcOsc;
mscanConfig.baudRate            = 1250000U;
mscanConfig.enableTimer         = false;
mscanConfig.enableLoopBack      = false;
mscanConfig.enableWakeup        = false;
mscanConfig.enableListen        = false;
mscanConfig.busoffrecMode       = kMSCAN_BusoffrecAuto;
mscanConfig.filterConfig.filterMode = kMSCAN_Filter32Bit;
MSCAN_Init(MSCAN, &mscanConfig, 8000000UL);
```

Parameters

- base – MsCAN peripheral base address.
- config – Pointer to the user-defined configuration structure.
- sourceClock_Hz – MsCAN Protocol Engine clock source frequency in Hz.

void MSCAN_Deinit(MSCAN_Type *base)

De-initializes a MsCAN instance.

This function disables the MsCAN module clock and sets all register values to the reset value.

Parameters

- base – MsCAN peripheral base address.

void MSCAN_GetDefaultConfig(*mscan_config_t* *config)

Gets the default configuration structure.

This function initializes the MsCAN configuration structure to default values.

Parameters

- config – Pointer to the MsCAN configuration structure.

static inline uint8_t MSCAN_GetTxBufferEmptyFlag(MSCAN_Type *base)

Get the transmit buffer empty status.

This flag indicates that the associated transmit message buffer is empty.

Parameters

- base – MsCAN peripheral base address.

static inline void MSCAN_TxBufferSelect(MSCAN_Type *base, uint8_t txBuf)

The selection of the actual transmit message buffer.

To get the next available transmit buffer, read the CANTFLG register and write its value back into the CANTBSEL register.

Parameters

- base – MsCAN peripheral base address.
- txBuf – The value read from CANTFLG.

static inline uint8_t MSCAN_GetTxBufferSelect(MSCAN_Type *base)

Get the actual transmit message buffer.

After write TFLG value back into the CANTBSEL register, read again CANBSEL to get the actual trasnsmit message buffer.

Parameters

- base – MsCAN peripheral base address.

static inline void MSCAN_TxBufferLaunch(MSCAN_Type *base, uint8_t txBuf)

Clear TFLG to schedule for transmission.

The CPU must clear the flag after a message is set up in the transmit buffer and is due for transmission.

Parameters

- base – MsCAN peripheral base address.
- txBuf – Message buffer(s) to be cleared.

static inline uint8_t MSCAN_GetTxBufferStatusFlags(MSCAN_Type *base, uint8_t mask)

Get Tx buffer status flag.

The bit is set after successful transmission.

Parameters

- base – MsCAN peripheral base address.
- mask – Message buffer(s) mask.

static inline uint8_t MSCAN_GetRxBufferFullFlag(MSCAN_Type *base)

Check Receive Buffer Full Flag.

RXF is set by the MSCAN when a new message is shifted in the receiver FIFO. This flag indicates whether the shifted buffer is loaded with a correctly received message.

Parameters

- base – MsCAN peripheral base address.

static inline void MSCAN_ClearRxBufferFullFlag(MSCAN_Type *base)

Clear Receive buffer Full flag.

After the CPU has read that message from the RxFG buffer in the receiver FIFO The RXF flag must be cleared to release the buffer.

Parameters

- base – MsCAN peripheral base address.

static inline uint8_t MSCAN_ReadRIDR0(MSCAN_Type *base)

static inline uint8_t MSCAN_ReadRIDR1(MSCAN_Type *base)

static inline uint8_t MSCAN_ReadRIDR2(MSCAN_Type *base)

static inline uint8_t MSCAN_ReadRIDR3(MSCAN_Type *base)

static inline void MSCAN_WriteTIDR0(MSCAN_Type *base, uint8_t id)

static inline void MSCAN_WriteTIDR1(MSCAN_Type *base, uint8_t id)

static inline void MSCAN_WriteTIDR2(MSCAN_Type *base, uint8_t id)

static inline void MSCAN_WriteTIDR3(MSCAN_Type *base, uint8_t id)

static inline void MSCAN_SetIDFilterMode(MSCAN_Type *base, *mscan_id_filter_mode_t* mode)

static inline void MSCAN_WriteIDAR0(MSCAN_Type *base, uint8_t *pID)

static inline void MSCAN_WriteIDAR1(MSCAN_Type *base, uint8_t *pID)

static inline void MSCAN_WriteIDMR0(MSCAN_Type *base, uint8_t *pID)

static inline void MSCAN_WriteIDMR1(MSCAN_Type *base, uint8_t *pID) void MSCAN_SetTimingConfig(MSCAN_Type *base, const *mscan_timing_config_t* *config)

Sets the MsCAN protocol timing characteristic.

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MSCAN_Init() and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling MSCAN_SetTimingConfig() overrides the baud rate set in MSCAN_Init().

Parameters

- base – MsCAN peripheral base address.
- config – Pointer to the timing configuration structure.

static inline uint8_t MSCAN_GetTxBufEmptyFlags(MSCAN_Type *base)

Gets the MsCAN Tx buffer empty flags.

This function gets MsCAN Tx buffer empty flags. It's returned as the value of the enumerators _mscan_tx_buffer_empty_flag.

Parameters

- base – MsCAN peripheral base address.

Returns

Tx buffer empty flags in the _mscan_tx_buffer_empty_flag.

static inline void MSCAN_EnableTxInterrupts(MSCAN_Type *base, uint8_t mask)

Enables MsCAN Transmitter interrupts according to the provided mask.

This function enables the MsCAN Tx empty interrupts according to the mask.

Parameters

- base – MsCAN peripheral base address.
- mask – The Tx interrupts mask to enable.

static inline void MSCAN_DisableTxInterrupts(MSCAN_Type *base, uint8_t mask)

Disables MsCAN Transmitter interrupts according to the provided mask.

This function disables the MsCAN Tx emtpy interrupts according to the mask.

Parameters

- base – MsCAN peripheral base address.
- mask – The Tx interrupts mask to disable.

static inline void MSCAN_EnableRxInterrupts(MSCAN_Type *base, uint8_t mask)

Enables MsCAN Receiver interrupts according to the provided mask.

This function enables the MsCAN Rx interrupts according to the provided mask which is a logical OR of enumeration members, see _mscan_interrupt_enable.

Parameters

- base – MsCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of _mscan_interrupt_enable.

static inline void MSCAN_DisableRxInterrupts(MSCAN_Type *base, uint8_t mask)

Disables MsCAN Receiver interrupts according to the provided mask.

This function disables the MsCAN Rx interrupts according to the provided mask which is a logical OR of enumeration members, see _mscan_interrupt_enable.

Parameters

- base – MsCAN peripheral base address.

- mask – The interrupts to disable. Logical OR of _mscan_interrupt_enable.

static inline void MSCAN_AbortTxRequest(MSCAN_Type *base, uint8_t mask)

Abort MsCAN Tx request.

This function allows abort request of queued messages.

Parameters

- base – MsCAN peripheral base address.
- mask – The Tx mask to abort.

static inline void MSCAN_Enable(MSCAN_Type *base, bool enable)

Enables or disables the MsCAN module operation.

This function enables or disables the MsCAN module.

Parameters

- base – MsCAN base pointer.
- enable – true to enable, false to disable.

status_t MSCAN_WriteTxMb(MSCAN_Type *base, *mscan_frame_t* *pTxFrame)

Writes a MsCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- base – MsCAN peripheral base address.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_MSCAN_DataLengthError – - Tx Message Buffer data length is wrong.

status_t MSCAN_ReadRxMb(MSCAN_Type *base, *mscan_frame_t* *pRxFrame)

Reads a MsCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – MsCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_MSCAN_DataLengthError – - Rx Message data length is wrong.

void MSCAN_TransferCreateHandle(MSCAN_Type *base, *mscan_handle_t* *handle, *mscan_transfer_callback_t* callback, void *userData)

Initializes the MsCAN handle.

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

Parameters

- base – MsCAN peripheral base address.
- handle – MsCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t MSCAN_TransferSendBlocking(MSCAN_Type *base, *mscan_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

Parameters

- base – MsCAN peripheral base pointer.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_MSCAN_DataLengthError – - Tx Message Buffer data length is wrong.

status_t MSCAN_TransferReceiveBlocking(MSCAN_Type *base, *mscan_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

Parameters

- base – MsCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame to be received.

Return values

- kStatus_Success – - Read Rx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_MSCAN_DataLengthError – - Rx Message data length is wrong.

status_t MSCAN_TransferSendNonBlocking(MSCAN_Type *base, *mscan_handle_t* *handle, *mscan_mb_transfer_t* *xfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – MsCAN peripheral base address.
- handle – MsCAN handle pointer.
- xfer – MsCAN Message Buffer transfer structure. See the mscan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_MSCAN_DataLengthError – - Tx Message Buffer data length is wrong.

status_t MSCAN_TransferReceiveNonBlocking(MSCAN_Type *base, *mscan_handle_t* *handle, *mscan_mb_transfer_t* *xfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – MsCAN peripheral base address.
- handle – MsCAN handle pointer.
- xfer – MsCAN Message Buffer transfer structure. See the mscan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_MSCAN_RxBusy – - Rx Message Buffer is in use.

void MSCAN_TransferAbortSend(MSCAN_Type *base, *mscan_handle_t* *handle, uint8_t mask)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – MsCAN peripheral base address.
- handle – MsCAN handle pointer.
- mask – The MsCAN Tx Message Buffer mask.

void MSCAN_TransferAbortReceive(MSCAN_Type *base, *mscan_handle_t* *handle, uint8_t mask)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – MsCAN peripheral base address.
- handle – MsCAN handle pointer.
- mask – The MsCAN Rx Message Buffer mask.

void MSCAN_TransferHandleIRQ(MSCAN_Type *base, *mscan_handle_t* *handle)

MSCAN IRQ handle function.

This function handles the MSCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – MSCAN peripheral base address.
- handle – MSCAN handle pointer.

FSL_MSCAN_DRIVER_VERSION

MsCAN driver version.

FlexCAN transfer status.

Values:

enumerator kStatus_MSCAN_TxBusy
Tx Message Buffer is Busy.

enumerator kStatus_MSCAN_TxIdle
Tx Message Buffer is Idle.

enumerator kStatus_MSCAN_TxSwitchToRx
Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus_MSCAN_RxBusy
Rx Message Buffer is Busy.

enumerator kStatus_MSCAN_RxIdle
Rx Message Buffer is Idle.

enumerator kStatus_MSCAN_RxOverflow
Rx Message Buffer is Overflowed.

enumerator kStatus_MSCAN_RxFifoBusy
Rx Message FIFO is Busy.

enumerator kStatus_MSCAN_RxFifoIdle
Rx Message FIFO is Idle.

enumerator kStatus_MSCAN_RxFifoOverflow
Rx Message FIFO is overflowed.

enumerator kStatus_MSCAN_RxFifoWarning
Rx Message FIFO is almost overflowed.

enumerator kStatus_MSCAN_ErrorStatus
FlexCAN Module Error and Status.

enumerator kStatus_MSCAN_UnHandled
UnHadled Interrupt asserted.

enumerator kStatus_MSCAN_DataLengthError
Frame data length is wrong.

enum _mscan_frame_format
MsCAN frame format.

Values:

enumerator kMSCAN_FrameFormatStandard
Standard frame format attribute.

enumerator kMSCAN_FrameFormatExtend
Extend frame format attribute.

enum _mscan_frame_type
MsCAN frame type.

Values:

enumerator kMSCAN_FrameTypeData
Data frame type attribute.

enumerator kMSCAN_FrameTypeRemote
Remote frame type attribute.

enum _mscan_clock_source
MsCAN clock source.

Values:

enumerator kMSCAN_ClkSrcOsc
MsCAN Protocol Engine clock from Oscillator.

enumerator kMSCAN_ClkSrcBus
MsCAN Protocol Engine clock from Bus Clock.

enum _mscan_busoffrec_mode
MsCAN bus-off recovery mode.

Values:

enumerator kMSCAN_BusoffrecAuto
MsCAN automatic bus-off recovery.

enumerator kMSCAN_BusoffrecUsr
MsCAN bus-off recovery upon user request.

enum _mscan_tx_buffer_empty_flag
MsCAN Tx buffer empty flag.

Values:

enumerator kMSCAN_TxBuf0Empty
MsCAN Tx Buffer 0 empty.

enumerator kMSCAN_TxBuf1Empty
MsCAN Tx Buffer 1 empty.

enumerator kMSCAN_TxBuf2Empty
MsCAN Tx Buffer 2 empty.

enumerator kMSCAN_TxBufFull
MsCAN Tx Buffer all not empty.

enum _mscan_id_filter_mode
MsCAN id filter mode.

Values:

enumerator kMSCAN_Filter32Bit
Two 32-bit acceptance filters.

enumerator kMSCAN_Filter16Bit
Four 16-bit acceptance filters.

enumerator kMSCAN_Filter8Bit
Eight 8-bit acceptance filters.

enumerator kMSCAN_FilterClose
Filter closed.

enum _mscan_interrupt_enable
MsCAN interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the MsCAN Module interrupt configurations.

Values:

enumerator kMSCAN_WakeUpInterruptEnable
Wake Up interrupt.

enumerator kMSCAN_StatusChangeInterruptEnable
Status change interrupt.

enumerator kMSCAN_RxStatusChangeInterruptEnable
Rx status change interrupt.

enumerator kMSCAN_TxStatusChangeInterruptEnable
Tx status change interrupt.

enumerator kMSCAN_OverrunInterruptEnable
Overrun interrupt.

enumerator kMSCAN_RxFullInterruptEnable
Rx buffer full interrupt.

enumerator kMSCAN_TxEmptyInterruptEnable
Tx buffer empty interrupt.

typedef enum *_mscan_frame_format* mscan_frame_format_t
MsCAN frame format.

typedef enum *_mscan_frame_type* mscan_frame_type_t
MsCAN frame type.

typedef enum *_mscan_clock_source* mscan_clock_source_t
MsCAN clock source.

typedef enum *_mscan_busoffrec_mode* mscan_busoffrec_mode_t
MsCAN bus-off recovery mode.

typedef enum *_mscan_id_filter_mode* mscan_id_filter_mode_t
MsCAN id filter mode.

typedef struct *_mscan_mb* mscan_mb_t
MsCAN message buffer structure.

typedef struct *_mscan_frame* mscan_frame_t
MsCAN frame structure.

typedef struct *_mscan_idfilter_config* mscan_idfilter_config_t
MsCAN module acceptance filter configuration structure.

typedef struct *_mscan_config* mscan_config_t
MsCAN module configuration structure.

typedef struct *_mscan_timing_config* mscan_timing_config_t
MsCAN protocol timing characteristic configuration structure.

typedef struct *_mscan_mb_transfer* mscan_mb_transfer_t
MSCAN Message Buffer transfer.

typedef struct *_mscan_handle* mscan_handle_t
MsCAN handle structure definition.

typedef void (*mscan_transfer_callback_t)(MSCAN_Type *base, *mscan_handle_t* *handle, *status_t* status, void *userData)
MsCAN transfer callback function.

The MsCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_MSCAN_ErrorStatus, the result parameter is the Content of MsCAN status register which can be used to get the working status(or error status) of MsCAN module. If the status equals to other MsCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other MsCAN Message Buffer transfer status, the result is meaningless and should be Ignored. If the status equals kStatus_MSCAN_DataLengthError, it means the received frame data length code (DLC) is wrong.

MSCAN_RX_MB_STD_MASK(id)

MsCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

MSCAN_RX_MB_EXT_MASK(id)

Extend Rx Message Buffer Mask helper macro.

struct MSCAN_IDR1Type

#include <fsl_mscan.h> MSCAN IDR1 struct.

Public Members

uint8_t EID17_15

Extended Format Identifier 17-15

uint8_t R_TEIDE

ID Extended

uint8_t R_TSRR

Substitute Remote Request

uint8_t EID20_18_OR_SID2_0

Extended Format Identifier 18-20 or standard format bit 0-2

struct MSCAN_IDR3Type

#include <fsl_mscan.h> MSCAN IDR3 struct.

Public Members

uint8_t ERTR

Remote Transmission Request

uint8_t EID6_0

Extended Format Identifier 6-0

union IDR1_3_UNION

#include <fsl_mscan.h> MSCAN idr1 and idr3 union.

Public Members

MSCAN_IDR1Type IDR1

structure for identifier 1

MSCAN_IDR3Type IDR3

structure for identifier 3

uint8_t Bytes

bytes

struct MSCAN_ExtendIDType

#include <fsl_mscan.h> MSCAN extend ID struct.

Public Members

uint32_t EID6_0
    ID[0:6]

uint32_t EID14_7
    ID[14:7]

uint32_t EID17_15
    ID[17:15]

uint32_t EID20_18
    ID[20:18]

uint32_t EID28_21
    ID[28:21]

struct MSCAN_StandardIDType
    #include <fsl_mscan.h> MSCAN standard ID struct.

Public Members

uint32_t EID2_0
    ID[0:2]

uint32_t EID10_3
    ID[10:3]

struct _mscan_mb
    #include <fsl_mscan.h> MsCAN message buffer structure.

Public Members

uint8_t EIDR0
    Extended Identifier Register 0

uint8_t EIDR1
    Extended Identifier Register 1

uint8_t EIDR2
    Extended Identifier Register 2

uint8_t EIDR3
    Extended Identifier Register 3

uint8_t EDSR[8]
    Extended Data Segment Register

uint8_t DLR
    data length field

uint8_t BPR
    Buffer Priority Register

uint8_t TSRH
    Time Stamp Register High

uint8_t TSRL
    Time Stamp Register Low

struct _mscan_frame
    #include <fsl_mscan.h> MsCAN frame structure.

Public Members

union *_mscan_frame* ID_Type
: identifier union

uint8_t DLR
: data length

uint8_t BPR
: transmit buffer priority

mscan_frame_type_t type
: remote frame or data frame

mscan_frame_format_t format
: extend frame or standard frame

uint8_t TSRH
: time stamp high byte

uint8_t TSRL
: time stamp low byte

struct _mscan_idfilter_config
: *#include <fsl_mscan.h>* MsCAN module acceptance filter configuration structure.

Public Members

mscan_id_filter_mode_t filterMode
: MSCAN Identifier Acceptance Filter Mode

uint32_t u32IDAR0
: MSCAN Identifier Acceptance Register n of First Bank

uint32_t u32IDAR1
: MSCAN Identifier Acceptance Register n of Second Bank

uint32_t u32IDMR0
: MSCAN Identifier Mask Register n of First Bank

uint32_t u32IDMR1
: MSCAN Identifier Mask Register n of Second Bank

struct _mscan_config
: *#include <fsl_mscan.h>* MsCAN module configuration structure.

Public Members

uint32_t baudRate
: MsCAN baud rate in bps.

bool enableTimer
: Enable or Disable free running timer.

bool enableWakeup
: Enable or Disable Wakeup Mode.

mscan_clock_source_t clkSrc
: Clock source for MsCAN Protocol Engine.

bool enableLoopBack
: Enable or Disable Loop Back Self Test Mode.

bool enableListen
: Enable or Disable Listen Only Mode.

mscan_busoffrec_mode_t busoffrecMode
: Bus-Off Recovery Mode.

struct _mscan_timing_config
: *#include <fsl_mscan.h>* MsCAN protocol timing characteristic configuration structure.

Public Members

uint8_t priDiv
: Baud rate prescaler.

uint8_t sJumpwidth
: Sync Jump Width.

uint8_t timeSeg1
: Time Segment 1.

uint8_t timeSeg2
: Time Segment 2.

uint8_t samp
: Number of samples per bit time.

struct _mscan_mb_transfer
: *#include <fsl_mscan.h>* MSCAN Message Buffer transfer.

Public Members

mscan_frame_t *frame
: The buffer of CAN Message to be transfer.

uint8_t mask
: The mask of Tx buffer.

struct _mscan_handle
: *#include <fsl_mscan.h>* MsCAN handle structure.

Public Members

mscan_transfer_callback_t callback
: Callback function.

void *userData
: MsCAN callback function parameter.

mscan_frame_t *volatile mbFrameBuf
: The buffer for received data from Message Buffers.

volatile uint8_t mbStateTx
: Message Buffer transfer state.

volatile uint8_t mbStateRx
: Message Buffer transfer state.

union ID_Type

Public Members

MSCAN_StandardIDType StdID
    standard format

MSCAN_ExtendIDType ExtID
    extend format

uint32_t ID
    Identifire with 32 bit format

union ___unnamed32___

Public Members

uint8_t DSR[8]
    data segment

struct _mscan_frame

struct _mscan_frame

struct ___unnamed34___

Public Members

uint32_t dataWord0
    MSCAN Frame payload word0.

uint32_t dataWord1
    MSCAN Frame payload word1.

struct ___unnamed36___

Public Members

uint8_t dataByte0
    MSCAN Frame payload byte0.

uint8_t dataByte1
    MSCAN Frame payload byte1.

uint8_t dataByte2
    MSCAN Frame payload byte2.

uint8_t dataByte3
    MSCAN Frame payload byte3.

uint8_t dataByte4
    MSCAN Frame payload byte4.

uint8_t dataByte5
    MSCAN Frame payload byte5.

uint8_t dataByte6
    MSCAN Frame payload byte6.

uint8_t dataByte7
    MSCAN Frame payload byte7.

2.32 PDB: Programmable Delay Block

void PDB_Init(PDB_Type *base, const *pdb_config_t* *config)

Initializes the PDB module.

This function initializes the PDB module. The operations included are as follows.

- Enable the clock for PDB instance.
- Configure the PDB module.
- Enable the PDB module.

Parameters

- base – PDB peripheral base address.
- config – Pointer to the configuration structure. See "pdb_config_t".

void PDB_Deinit(PDB_Type *base)

De-initializes the PDB module.

Parameters

- base – PDB peripheral base address.

void PDB_GetDefaultConfig(*pdb_config_t* *config)

Initializes the PDB user configuration structure.

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

```
config->loadValueMode = kPDB_LoadValueImmediately;
config->prescalerDivider = kPDB_PrescalerDivider1;
config->dividerMultiplicationFactor = kPDB_DividerMultiplicationFactor1;
config->triggerInputSource = kPDB_TriggerSoftware;
config->enableContinuousMode = false;
```

Parameters

- config – Pointer to configuration structure. See "pdb_config_t".

static inline void PDB_Enable(PDB_Type *base, bool enable)

Enables the PDB module.

Parameters

- base – PDB peripheral base address.
- enable – Enable the module or not.

static inline void PDB_DoSoftwareTrigger(PDB_Type *base)

Triggers the PDB counter by software.

Parameters

- base – PDB peripheral base address.

static inline void PDB_DoLoadValues(PDB_Type *base)

Loads the counter values.

This function loads the counter values from the internal buffer. See "pdb_load_value_mode_t" about PDB's load mode.

Parameters

- base – PDB peripheral base address.

static inline void PDB_EnableDMA(PDB_Type *base, bool enable)

Enables the DMA for the PDB module.

Parameters

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

static inline void PDB_EnableInterrupts(PDB_Type *base, uint32_t mask)

Enables the interrupts for the PDB module.

Parameters

- base – PDB peripheral base address.
- mask – Mask value for interrupts. See “_pdb_interrupt_enable”.

static inline void PDB_DisableInterrupts(PDB_Type *base, uint32_t mask)

Disables the interrupts for the PDB module.

Parameters

- base – PDB peripheral base address.
- mask – Mask value for interrupts. See “_pdb_interrupt_enable”.

static inline uint32_t PDB_GetStatusFlags(PDB_Type *base)

Gets the status flags of the PDB module.

Parameters

- base – PDB peripheral base address.

Returns

Mask value for asserted flags. See “_pdb_status_flags”.

static inline void PDB_ClearStatusFlags(PDB_Type *base, uint32_t mask)

Clears the status flags of the PDB module.

Parameters

- base – PDB peripheral base address.
- mask – Mask value of flags. See “_pdb_status_flags”.

static inline void PDB_SetModulusValue(PDB_Type *base, uint32_t value)

Specifies the counter period.

Parameters

- base – PDB peripheral base address.
- value – Setting value for the modulus. 16-bit is available.

static inline uint32_t PDB_GetCounterValue(PDB_Type *base)

Gets the PDB counter’s current value.

Parameters

- base – PDB peripheral base address.

Returns

PDB counter’s current value.

static inline void PDB_SetCounterDelayValue(PDB_Type *base, uint32_t value)

Sets the value for the PDB counter delay event.

Parameters

- base – PDB peripheral base address.

- value – Setting value for PDB counter delay event. 16-bit is available.

static inline void PDB_SetADCPreTriggerConfig(PDB_Type *base, *pdb_adc_trigger_channel_t* channel, *pdb_adc_pretrigger_config_t* *config)

Configures the ADC pre-trigger in the PDB module.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.
- config – Pointer to the configuration structure. See "pdb_adc_pretrigger_config_t".

static inline void PDB_SetADCPreTriggerDelayValue(PDB_Type *base, *pdb_adc_trigger_channel_t* channel, *pdb_adc_pretrigger_t* pretriggerNumber, uint32_t value)

Sets the value for the ADC pre-trigger delay event.

This function sets the value for ADC pre-trigger delay event. It specifies the delay value for the channel's corresponding pre-trigger. The pre-trigger asserts when the PDB counter is equal to the set value.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.
- pretriggerNumber – Channel group index for ADC instance.
- value – Setting value for ADC pre-trigger delay event. 16-bit is available.

static inline uint32_t PDB_GetADCPreTriggerStatusFlags(PDB_Type *base, *pdb_adc_trigger_channel_t* channel)

Gets the ADC pre-trigger's status flags.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.

Returns

Mask value for asserted flags. See "_pdb_adc_pretrigger_flags".

static inline void PDB_ClearADCPreTriggerStatusFlags(PDB_Type *base, *pdb_adc_trigger_channel_t* channel, uint32_t mask)

Clears the ADC pre-trigger status flags.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.
- mask – Mask value for flags. See "_pdb_adc_pretrigger_flags".

void PDB_SetDACTriggerConfig(PDB_Type *base, *pdb_dac_trigger_channel_t* channel, *pdb_dac_trigger_config_t* *config)

Configures the DAC trigger in the PDB module.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for DAC instance.

- config – Pointer to the configuration structure. See "pdb_dac_trigger_config_t".

static inline void PDB_SetDACTriggerIntervalValue(PDB_Type *base, *pdb_dac_trigger_channel_t* channel, uint32_t value)

Sets the value for the DAC interval event.

This function sets the value for DAC interval event. DAC interval trigger triggers the DAC module to update the buffer when the DAC interval counter is equal to the set value.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for DAC instance.
- value – Setting value for the DAC interval event.

static inline void PDB_EnablePulseOutTrigger(PDB_Type *base, *pdb_pulse_out_channel_mask_t* channelMask, bool enable)

Enables the pulse out trigger channels.

Parameters

- base – PDB peripheral base address.
- channelMask – Channel mask value for multiple pulse out trigger channel.
- enable – Whether the feature is enabled or not.

static inline void PDB_SetPulseOutTriggerDelayValue(PDB_Type *base, *pdb_pulse_out_trigger_channel_t* channel, uint32_t value1, uint32_t value2)

Sets event values for the pulse out trigger.

This function is used to set event values for the pulse output trigger. These pulse output trigger delay values specify the delay for the PDB Pulse-out. Pulse-out goes high when the PDB counter is equal to the pulse output high value (value1). Pulse-out goes low when the PDB counter is equal to the pulse output low value (value2).

Parameters

- base – PDB peripheral base address.
- channel – Channel index for pulse out trigger channel.
- value1 – Setting value for pulse out high.
- value2 – Setting value for pulse out low.

FSL_PDB_DRIVER_VERSION

PDB driver version 2.0.4.

enum _pdb_status_flags

PDB flags.

Values:

enumerator kPDB_LoadOKFlag

This flag is automatically cleared when the values in buffers are loaded into the internal registers after the LDOK bit is set or the PDBEN is cleared.

enumerator kPDB_DelayEventFlag

PDB timer delay event flag.

enum _pdb_adc_pretrigger_flags

PDB ADC PreTrigger channel flags.

Values:

enumerator kPDB_ADCPreTriggerChannel0Flag

Pre-trigger 0 flag.

enumerator kPDB_ADCPreTriggerChannel1Flag

Pre-trigger 1 flag.

enumerator kPDB_ADCPreTriggerChannel0ErrorFlag

Pre-trigger 0 Error.

enumerator kPDB_ADCPreTriggerChannel1ErrorFlag

Pre-trigger 1 Error.

enum _pdb_interrupt_enable

PDB buffer interrupts.

Values:

enumerator kPDB_SequenceErrorInterruptEnable

PDB sequence error interrupt enable.

enumerator kPDB_DelayInterruptEnable

PDB delay interrupt enable.

enum _pdb_load_value_mode

PDB load value mode.

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx_MOD, PDBx_IDLY)
- ADC trigger (PDBx_CHnDLYm)
- DAC trigger (PDBx_DACINTx)
- CMP trigger (PDBx_POyDLY)

Values:

enumerator kPDB_LoadValueImmediately

Load immediately after 1 is written to LDOK.

enumerator kPDB_LoadValueOnCounterOverflow

Load when the PDB counter overflows (reaches the MOD register value).

enumerator kPDB_LoadValueOnTriggerInput

Load a trigger input event is detected.

enumerator kPDB_LoadValueOnCounterOverflowOrTriggerInput

Load either when the PDB counter overflows or a trigger input is detected.

enum _pdb_prescaler_divider

Prescaler divider.

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

Values:

enumerator kPDB_PrescalerDivider1

Divider x1.

enumerator kPDB_PrescalerDivider2
: Divider x2.

enumerator kPDB_PrescalerDivider4
: Divider x4.

enumerator kPDB_PrescalerDivider8
: Divider x8.

enumerator kPDB_PrescalerDivider16
: Divider x16.

enumerator kPDB_PrescalerDivider32
: Divider x32.

enumerator kPDB_PrescalerDivider64
: Divider x64.

enumerator kPDB_PrescalerDivider128
: Divider x128.

enum _pdb_divider_multiplication_factor

Multiplication factor select for prescaler.

Selects the multiplication factor of the prescaler divider for the counter clock.

Values:

enumerator kPDB_DividerMultiplicationFactor1
: Multiplication factor is 1.

enumerator kPDB_DividerMultiplicationFactor10
: Multiplication factor is 10.

enumerator kPDB_DividerMultiplicationFactor20
: Multiplication factor is 20.

enumerator kPDB_DividerMultiplicationFactor40
: Multiplication factor is 40.

enum _pdb_trigger_input_source

Trigger input source.

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

Values:

enumerator kPDB_TriggerInput0
: Trigger-In 0.

enumerator kPDB_TriggerInput1
: Trigger-In 1.

enumerator kPDB_TriggerInput2
: Trigger-In 2.

enumerator kPDB_TriggerInput3
: Trigger-In 3.

enumerator kPDB_TriggerInput4
: Trigger-In 4.

enumerator kPDB_TriggerInput5
: Trigger-In 5.

enumerator kPDB_TriggerInput6
: Trigger-In 6.

enumerator kPDB_TriggerInput7
: Trigger-In 7.

enumerator kPDB_TriggerInput8
: Trigger-In 8.

enumerator kPDB_TriggerInput9
: Trigger-In 9.

enumerator kPDB_TriggerInput10
: Trigger-In 10.

enumerator kPDB_TriggerInput11
: Trigger-In 11.

enumerator kPDB_TriggerInput12
: Trigger-In 12.

enumerator kPDB_TriggerInput13
: Trigger-In 13.

enumerator kPDB_TriggerInput14
: Trigger-In 14.

enumerator kPDB_TriggerSoftware
: Trigger-In 15, software trigger.

enum _pdb_adc_trigger_channel
: List of PDB ADC trigger channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPDB_ADCTriggerChannel0
: PDB ADC trigger channel number 0

enumerator kPDB_ADCTriggerChannel1
: PDB ADC trigger channel number 1

enumerator kPDB_ADCTriggerChannel2
: PDB ADC trigger channel number 2

enumerator kPDB_ADCTriggerChannel3
: PDB ADC trigger channel number 3

enum _pdb_adc_pretrigger
: List of PDB ADC pretrigger.

Note: Actual number of available pretrigger channels is SoC dependent

Values:

enumerator kPDB_ADCPreTrigger0
PDB ADC pretrigger number 0

enumerator kPDB_ADCPreTrigger1
PDB ADC pretrigger number 1

enumerator kPDB_ADCPreTrigger2
PDB ADC pretrigger number 2

enumerator kPDB_ADCPreTrigger3
PDB ADC pretrigger number 3

enumerator kPDB_ADCPreTrigger4
PDB ADC pretrigger number 4

enumerator kPDB_ADCPreTrigger5
PDB ADC pretrigger number 5

enumerator kPDB_ADCPreTrigger6
PDB ADC pretrigger number 6

enumerator kPDB_ADCPreTrigger7
PDB ADC pretrigger number 7

enum _pdb_dac_trigger_channel
List of PDB DAC trigger channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPDB_DACTriggerChannel0
PDB DAC trigger channel number 0

enumerator kPDB_DACTriggerChannel1
PDB DAC trigger channel number 1

enum _pdb_pulse_out_trigger_channel
List of PDB pulse out trigger channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPDB_PulseOutTriggerChannel0
PDB pulse out trigger channel number 0

enumerator kPDB_PulseOutTriggerChannel1
PDB pulse out trigger channel number 1

enumerator kPDB_PulseOutTriggerChannel2
PDB pulse out trigger channel number 2

enumerator kPDB_PulseOutTriggerChannel3
PDB pulse out trigger channel number 3

enum _pdb_pulse_out_channel_mask

List of PDB pulse out trigger channels mask.

Note: Actual number of available channels mask is SoC dependent

Values:

enumerator kPDB_PulseOutChannel0Mask

PDB pulse out trigger channel number 0 mask

enumerator kPDB_PulseOutChannel1Mask

PDB pulse out trigger channel number 1 mask

enumerator kPDB_PulseOutChannel2Mask

PDB pulse out trigger channel number 2 mask

enumerator kPDB_PulseOutChannel3Mask

PDB pulse out trigger channel number 3 mask

typedef enum *_pdb_load_value_mode* pdb_load_value_mode_t

PDB load value mode.

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx_MOD, PDBx_IDLY)
- ADC trigger (PDBx_CHnDLYm)
- DAC trigger (PDBx_DACINTx)
- CMP trigger (PDBx_POyDLY)

typedef enum *_pdb_prescaler_divider* pdb_prescaler_divider_t

Prescaler divider.

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

typedef enum *_pdb_divider_multiplication_factor* pdb_divider_multiplication_factor_t

Multiplication factor select for prescaler.

Selects the multiplication factor of the prescaler divider for the counter clock.

typedef enum *_pdb_trigger_input_source* pdb_trigger_input_source_t

Trigger input source.

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

typedef enum *_pdb_adc_trigger_channel* pdb_adc_trigger_channel_t

List of PDB ADC trigger channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pdb_adc_pretrigger* pdb_adc_pretrigger_t

List of PDB ADC pretrigger.

Note: Actual number of available pretrigger channels is SoC dependent

typedef enum *_pdb_dac_trigger_channel* pdb_dac_trigger_channel_t
List of PDB DAC trigger channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pdb_pulse_out_trigger_channel* pdb_pulse_out_trigger_channel_t
List of PDB pulse out trigger channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pdb_pulse_out_channel_mask* pdb_pulse_out_channel_mask_t
List of PDB pulse out trigger channels mask.

Note: Actual number of available channels mask is SoC dependent

typedef struct *_pdb_config* pdb_config_t
PDB module configuration.

typedef struct *_pdb_adc_pretrigger_config* pdb_adc_pretrigger_config_t
PDB ADC Pre-trigger configuration.

typedef struct *_pdb_dac_trigger_config* pdb_dac_trigger_config_t
PDB DAC trigger configuration.

struct _pdb_config
#include <fsl_pdb.h> PDB module configuration.

Public Members

pdb_load_value_mode_t loadValueMode
Select the load value mode.

pdb_prescaler_divider_t prescalerDivider
Select the prescaler divider.

pdb_divider_multiplication_factor_t dividerMultiplicationFactor
Multiplication factor select for prescaler.

pdb_trigger_input_source_t triggerInputSource
Select the trigger input source.

bool enableContinuousMode
Enable the PDB operation in Continuous mode.

struct _pdb_adc_pretrigger_config
#include <fsl_pdb.h> PDB ADC Pre-trigger configuration.

Public Members

uint32_t enablePreTriggerMask
PDB Channel Pre-trigger Enable.

uint32_t enableOutputMask
PDB Channel Pre-trigger Output Select. PDB channel's corresponding pre-trigger asserts when the counter reaches the channel delay register.

uint32_t enableBackToBackOperationMask

PDB Channel pre-trigger Back-to-Back Operation Enable. Back-to-back operation enables the ADC conversions complete to trigger the next PDB channel pre-trigger and trigger output, so that the ADC conversions can be triggered on next set of configuration and results registers.

struct _pdb_dac_trigger_config

#include <fsl_pdb.h> PDB DAC trigger configuration.

Public Members

bool enableExternalTriggerInput

Enables the external trigger for DAC interval counter.

bool enableIntervalTrigger

Enables the DAC interval trigger.

2.33 PMC: Power Management Controller

static inline void PMC_GetVersionId(PMC_Type *base, *pmc_version_id_t* *versionId)

Gets the PMC version ID.

This function gets the PMC version ID, including major version number, minor version number, and a feature specification number.

Parameters

- base – PMC peripheral base address.
- versionId – Pointer to version ID structure.

void PMC_GetParam(PMC_Type *base, *pmc_param_t* *param)

Gets the PMC parameter.

This function gets the PMC parameter including the VLPO enable and the HVD enable.

Parameters

- base – PMC peripheral base address.
- param – Pointer to PMC param structure.

void PMC_ConfigureLowVoltDetect(PMC_Type *base, const *pmc_low_volt_detect_config_t* *config)

Configures the low-voltage detect setting.

This function configures the low-voltage detect setting, including the trip point voltage setting, enables or disables the interrupt, enables or disables the system reset.

Parameters

- base – PMC peripheral base address.
- config – Low-voltage detect configuration structure.

static inline bool PMC_GetLowVoltDetectFlag(PMC_Type *base)

Gets the Low-voltage Detect Flag status.

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

Parameters

- base – PMC peripheral base address.

Returns
Current low-voltage detect flag

- true: Low-voltage detected
- false: Low-voltage not detected

static inline void PMC_ClearLowVoltDetectFlag(PMC_Type *base)

Acknowledges clearing the Low-voltage Detect flag.

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

Parameters

- base – PMC peripheral base address.

void PMC_ConfigureLowVoltWarning(PMC_Type *base, const *pmc_low_volt_warning_config_t* *config)

Configures the low-voltage warning setting.

This function configures the low-voltage warning setting, including the trip point voltage setting and enabling or disabling the interrupt.

Parameters

- base – PMC peripheral base address.
- config – Low-voltage warning configuration structure.

static inline bool PMC_GetLowVoltWarningFlag(PMC_Type *base)

Gets the Low-voltage Warning Flag status.

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

Parameters

- base – PMC peripheral base address.

Returns
Current LVWF status

- true: Low-voltage Warning Flag is set.
- false: the Low-voltage Warning does not happen.

static inline void PMC_ClearLowVoltWarningFlag(PMC_Type *base)

Acknowledges the Low-voltage Warning flag.

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

- base – PMC peripheral base address.

void PMC_ConfigureHighVoltDetect(PMC_Type *base, const *pmc_high_volt_detect_config_t* *config)

Configures the high-voltage detect setting.

This function configures the high-voltage detect setting, including the trip point voltage setting, enabling or disabling the interrupt, enabling or disabling the system reset.

Parameters

- base – PMC peripheral base address.
- config – High-voltage detect configuration structure.

static inline bool PMC_GetHighVoltDetectFlag(PMC_Type *base)

Gets the High-voltage Detect Flag status.

This function reads the current HVDF status. If it returns 1, a low voltage event is detected.

Parameters

- base – PMC peripheral base address.

Returns

Current high-voltage detect flag

- true: High-voltage detected
- false: High-voltage not detected

static inline void PMC_ClearHighVoltDetectFlag(PMC_Type *base)

Acknowledges clearing the High-voltage Detect flag.

This function acknowledges the high-voltage detection errors (write 1 to clear HVDF).

Parameters

- base – PMC peripheral base address.

void PMC_ConfigureBandgapBuffer(PMC_Type *base, const *pmc_bandgap_buffer_config_t* *config)

Configures the PMC bandgap.

This function configures the PMC bandgap, including the drive select and behavior in low-power mode.

Parameters

- base – PMC peripheral base address.
- config – Pointer to the configuration structure

static inline bool PMC_GetPeriphIOIsolationFlag(PMC_Type *base)

Gets the acknowledge Peripherals and I/O pads isolation flag.

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

Parameters

- base – PMC peripheral base address.
- base – Base address for current PMC instance.

Returns

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

static inline void PMC_ClearPeriphIOIsolationFlag(PMC_Type *base)

Acknowledges the isolation flag to Peripherals and I/O pads.

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

Parameters

- base – PMC peripheral base address.

static inline bool PMC_IsRegulatorInRunRegulation(PMC_Type *base)

Gets the regulator regulation status.

This function returns the regulator to run a regulation status. It provides the current status of the internal voltage regulator.

Parameters

- base – PMC peripheral base address.
- base – Base address for current PMC instance.

Returns

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.

FSL_PMC_DRIVER_VERSION

PMC driver version.

Version 2.0.3.

enum _pmc_low_volt_detect_volt_select

Low-voltage Detect Voltage Select.

Values:

enumerator kPMC_LowVoltDetectLowTrip

Low-trip point selected (VLVD = VLVDL)

enumerator kPMC_LowVoltDetectHighTrip

High-trip point selected (VLVD = VLVDH)

enum _pmc_low_volt_warning_volt_select

Low-voltage Warning Voltage Select.

Values:

enumerator kPMC_LowVoltWarningLowTrip

Low-trip point selected (VLVW = VLVW1)

enumerator kPMC_LowVoltWarningMid1Trip

Mid 1 trip point selected (VLVW = VLVW2)

enumerator kPMC_LowVoltWarningMid2Trip

Mid 2 trip point selected (VLVW = VLVW3)

enumerator kPMC_LowVoltWarningHighTrip

High-trip point selected (VLVW = VLVW4)

enum _pmc_high_volt_detect_volt_select

High-voltage Detect Voltage Select.

Values:

enumerator kPMC_HighVoltDetectLowTrip

Low-trip point selected (VHVD = VHVDL)

enumerator kPMC_HighVoltDetectHighTrip

High-trip point selected (VHVD = VHVDH)

enum _pmc_bandgap_buffer_drive_select

Bandgap Buffer Drive Select.

Values:

enumerator kPMC_BandgapBufferDriveLow

Low-drive.

enumerator kPMC_BandgapBufferDriveHigh

High-drive.

enum _pmc_vlp_freq_option
: VLPx Option.

  Values:

  enumerator kPMC_FreqRestrict
  : Frequency is restricted in VLPx mode.

  enumerator kPMC_FreqUnrestrict
  : Frequency is unrestricted in VLPx mode.

typedef enum *_pmc_low_volt_detect_volt_select* pmc_low_volt_detect_volt_select_t
: Low-voltage Detect Voltage Select.

typedef enum *_pmc_low_volt_warning_volt_select* pmc_low_volt_warning_volt_select_t
: Low-voltage Warning Voltage Select.

typedef enum *_pmc_high_volt_detect_volt_select* pmc_high_volt_detect_volt_select_t
: High-voltage Detect Voltage Select.

typedef enum *_pmc_bandgap_buffer_drive_select* pmc_bandgap_buffer_drive_select_t
: Bandgap Buffer Drive Select.

typedef enum *_pmc_vlp_freq_option* pmc_vlp_freq_mode_t
: VLPx Option.

typedef struct *_pmc_version_id* pmc_version_id_t
: IP version ID definition.

typedef struct *_pmc_param* pmc_param_t
: IP parameter definition.

typedef struct *_pmc_low_volt_detect_config* pmc_low_volt_detect_config_t
: Low-voltage Detect Configuration Structure.

typedef struct *_pmc_low_volt_warning_config* pmc_low_volt_warning_config_t
: Low-voltage Warning Configuration Structure.

typedef struct *_pmc_high_volt_detect_config* pmc_high_volt_detect_config_t
: High-voltage Detect Configuration Structure.

typedef struct *_pmc_bandgap_buffer_config* pmc_bandgap_buffer_config_t
: Bandgap Buffer configuration.

struct _pmc_version_id
: *#include <fsl_pmc.h>* IP version ID definition.

  Public Members

  uint16_t feature
  : Feature Specification Number.

  uint8_t minor
  : Minor version number.

  uint8_t major
  : Major version number.

struct _pmc_param
: *#include <fsl_pmc.h>* IP parameter definition.

Public Members

bool vlpoEnable
VLPO enable.

bool hvdEnable
HVD enable.

struct _pmc_low_volt_detect_config
#include <fsl_pmc.h> Low-voltage Detect Configuration Structure.

Public Members

bool enableInt
Enable interrupt when Low-voltage detect

bool enableReset
Enable system reset when Low-voltage detect

pmc_low_volt_detect_volt_select_t voltSelect
Low-voltage detect trip point voltage selection

struct _pmc_low_volt_warning_config
#include <fsl_pmc.h> Low-voltage Warning Configuration Structure.

Public Members

bool enableInt
Enable interrupt when low-voltage warning

pmc_low_volt_warning_volt_select_t voltSelect
Low-voltage warning trip point voltage selection

struct _pmc_high_volt_detect_config
#include <fsl_pmc.h> High-voltage Detect Configuration Structure.

Public Members

bool enableInt
Enable interrupt when high-voltage detect

bool enableReset
Enable system reset when high-voltage detect

pmc_high_volt_detect_volt_select_t voltSelect
High-voltage detect trip point voltage selection

struct _pmc_bandgap_buffer_config
#include <fsl_pmc.h> Bandgap Buffer configuration.

Public Members

bool enable
Enable bandgap buffer.

bool enableInLowPowerMode
Enable bandgap buffer in low-power mode.

pmc_bandgap_buffer_drive_select_t drive
Bandgap buffer drive select.

2.34 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]

2.35 PWT: Pulse Width Timer

void PWT_Init(PWT_Type *base, const *pwt_config_t* *config)

Ungates the PWT clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the PWT driver.

Parameters

- base – PWT peripheral base address
- config – Pointer to the user configuration structure.

void PWT_Deinit(PWT_Type *base)

Gates the PWT clock.

Parameters

- base – PWT peripheral base address

void PWT_GetDefaultConfig(*pwt_config_t* *config)

Fills in the PWT configuration structure with the default settings.

The default values are:

```
config->clockSource = kPWT_BusClock;
config->prescale = kPWT_Prescale_Divide_1;
config->inputSelect = kPWT_InputPort_0;
config->enableFirstCounterLoad = false;
```

Parameters

- config – Pointer to the user configuration structure.

static inline void PWT_EnableInterrupts(PWT_Type *base, uint32_t mask)

Enables the selected PWT interrupts.

Parameters

- base – PWT peripheral base address

- mask – The interrupts to enable. This is a logical OR of members of the enumeration pwt_interrupt_enable_t

static inline void PWT_DisableInterrupts(PWT_Type *base, uint32_t mask)

Disables the selected PWT interrupts.

Parameters

- base – PWT peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration pwt_interrupt_enable_t

static inline uint32_t PWT_GetEnabledInterrupts(PWT_Type *base)

Gets the enabled PWT interrupts.

Parameters

- base – PWT peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration pwt_interrupt_enable_t

static inline uint32_t PWT_GetStatusFlags(PWT_Type *base)

Gets the PWT status flags.

Parameters

- base – PWT peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration pwt_status_flags_t

static inline void PWT_ClearStatusFlags(PWT_Type *base, uint32_t mask)

Clears the PWT status flags.

Parameters

- base – PWT peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration pwt_status_flags_t

static inline void PWT_StartTimer(PWT_Type *base)

Starts the PWT counter.

Parameters

- base – PWT peripheral base address

static inline void PWT_StopTimer(PWT_Type *base)

Stops the PWT counter.

Parameters

- base – PWT peripheral base address

enum _pwt_clock_source

PWT clock source selection.

Values:

enumerator kPWT_BusClock

The Bus clock is used as the clock source of PWT counter

enumerator kPWT_AlternativeClock

Alternative clock is used as the clock source of PWT counter

enum _pwt_clock_prescale
PWT prescaler factor selection for clock source.

Values:

enumerator kPWT_Prescale_Divide_1
PWT clock divided by 1

enumerator kPWT_Prescale_Divide_2
PWT clock divided by 2

enumerator kPWT_Prescale_Divide_4
PWT clock divided by 4

enumerator kPWT_Prescale_Divide_8
PWT clock divided by 8

enumerator kPWT_Prescale_Divide_16
PWT clock divided by 16

enumerator kPWT_Prescale_Divide_32
PWT clock divided by 32

enumerator kPWT_Prescale_Divide_64
PWT clock divided by 64

enumerator kPWT_Prescale_Divide_128
PWT clock divided by 128

enum _pwt_input_select
PWT input port selection.

Values:

enumerator kPWT_InputPort_0
PWT input comes from PWTIN[0]

enumerator kPWT_InputPort_1
PWT input comes from PWTIN[1]

enumerator kPWT_InputPort_2
PWT input comes from PWTIN[2]

enumerator kPWT_InputPort_3
PWT input comes from PWTIN[3]

enum _pwt_interrupt_enable
List of PWT interrupts.

Values:

enumerator kPWT_PulseWidthReadyInterruptEnable
Pulse width data ready interrupt

enumerator kPWT_CounterOverflowInterruptEnable
Counter overflow interrupt

enum _pwt_status_flags
List of PWT flags.

Values:

enumerator kPWT_CounterOverflowFlag
Counter overflow flag

enumerator kPWT_PulseWidthValidFlag

Pulse width valid flag

typedef enum *_pwt_clock_source* pwt_clock_source_t

PWT clock source selection.

typedef enum *_pwt_clock_prescale* pwt_clock_prescale_t

PWT prescaler factor selection for clock source.

typedef enum *_pwt_input_select* pwt_input_select_t

PWT input port selection.

typedef struct *_pwt_config* pwt_config_t

PWT configuration structure.

This structure holds the configuration settings for the PWT peripheral. To initialize this structure to reasonable defaults, call the PWT_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 inline uint16_t PWT_GetCurrentTimerCount(PWT_Type *base)

Reads the current counter value.

This function returns the timer counting value

Parameters

- base – PWT peripheral base address

Returns

Current 16-bit timer counter value

static inline uint16_t PWT_ReadPositivePulseWidth(PWT_Type *base)

Reads the positive pulse width.

This function reads the low and high registers and returns the 16-bit positive pulse width

Parameters

- base – PWT peripheral base address.

Returns

The 16-bit positive pulse width.

static inline uint16_t PWT_ReadNegativePulseWidth(PWT_Type *base)

Reads the negative pulse width.

This function reads the low and high registers and returns the 16-bit negative pulse width

Parameters

- base – PWT peripheral base address.

Returns

The 16-bit negative pulse width.

static inline void PWT_Reset(PWT_Type *base)

Performs a software reset on the PWT module.

Parameters

- base – PWT peripheral base address

FSL_PWT_DRIVER_VERSION

Version 2.0.2

struct _pwt_config

#include <fsl_pwt.h> PWT configuration structure.

This structure holds the configuration settings for the PWT peripheral. To initialize this structure to reasonable defaults, call the PWT_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

pwt_clock_source_t clockSource

Clock source for the counter

pwt_clock_prescale_t prescale

Pre-scaler to divide down the clock

pwt_input_select_t inputSelect

PWT Pulse input port selection

bool enableFirstCounterLoad

true: Load the first counter value to registers; false: Do not load first counter value

2.36 RCM: Reset Control Module Driver

static inline void RCM_GetVersionId(RCM_Type *base, *rcm_version_id_t* *versionId)

Gets the RCM version ID.

This function gets the RCM version ID including the major version number, the minor version number, and the feature specification number.

Parameters

- base – RCM peripheral base address.
- versionId – Pointer to the version ID structure.

static inline uint32_t RCM_GetResetSourceImplementedStatus(RCM_Type *base)

Gets the reset source implemented status.

This function gets the RCM parameter that indicates whether the corresponding reset source is implemented. Use source masks defined in the rcm_reset_source_t to get the desired source status.

This is an example.

```
uint32_t status;

To test whether the MCU is reset using Watchdog.
status = RCM_GetResetSourceImplementedStatus(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.

Returns

All reset source implemented status bit map.

static inline uint32_t RCM_GetPreviousResetSources(RCM_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 rcm_reset_source_t to get the desired source status.

This is an example.

```
uint32_t resetStatus;

To get all reset source statuses.
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;

To test whether the MCU is reset using Watchdog.
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceWdog;

To test multiple reset sources.
resetStatus = RCM_GetPreviousResetSources(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.

Returns

All reset source status bit map.

static inline uint32_t RCM_GetStickyResetSources(RCM_Type *base)

Gets the sticky reset source status.

This function gets the current reset source status that has not been cleared by software for a specific source.

This is an example.

```
uint32_t resetStatus;

To get all reset source statuses.
resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceAll;

To test whether the MCU is reset using Watchdog.
resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceWdog;

To test multiple reset sources.
resetStatus = RCM_GetStickyResetSources(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.

Returns

All reset source status bit map.

static inline void RCM_ClearStickyResetSources(RCM_Type *base, uint32_t sourceMasks)

Clears the sticky reset source status.

This function clears the sticky system reset flags indicated by source masks.

This is an example.

```
Clears multiple reset sources.
RCM_ClearStickyResetSources(kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.

- sourceMasks – reset source status bit map

void RCM_ConfigureResetPinFilter(RCM_Type *base, const *rcm_reset_pin_filter_config_t* *config)

Configures the reset pin filter.

This function sets the reset pin filter including the filter source, filter width, and so on.

Parameters

- base – RCM peripheral base address.
- config – Pointer to the configuration structure.

static inline bool RCM_GetEasyPortModePinStatus(RCM_Type *base)

Gets the EZP_MS_B pin assert status.

This function gets the easy port mode status (EZP_MS_B) pin assert status.

Parameters

- base – RCM peripheral base address.

Returns

status true - asserted, false - reasserted

static inline *rcm_boot_rom_config_t* RCM_GetBootRomSource(RCM_Type *base)

Gets the ROM boot source.

This function gets the ROM boot source during the last chip reset.

Parameters

- base – RCM peripheral base address.

Returns

The ROM boot source.

static inline void RCM_ClearBootRomSource(RCM_Type *base)

Clears the ROM boot source flag.

This function clears the ROM boot source flag.

Parameters

- base – Register base address of RCM

void RCM_SetForceBootRomSource(RCM_Type *base, *rcm_boot_rom_config_t* config)

Forces the boot from ROM.

This function forces booting from ROM during all subsequent system resets.

Parameters

- base – RCM peripheral base address.
- config – Boot configuration.

static inline void RCM_SetSystemResetInterruptConfig(RCM_Type *base, uint32_t intMask, *rcm_reset_delay_t* delay)

Sets the system reset interrupt configuration.

For a graceful shut down, the RCM 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 and the delay period. The interrupts are passed in as bit mask. See rcm_int_t for details. For example, to delay a reset for 512 LPO cycles after the WDOG timeout or loss-of-clock occurs, configure as follows: RCM_SetSystemResetInterruptConfig(kRCM_IntWatchDog | kRCM_IntLossOfClk, kRCM_ResetDelay512Lpo);

Parameters

- base – RCM peripheral base address.
- intMask – Bit mask of the system reset interrupts to enable. See rcm_interrupt_enable_t for details.
- delay – Bit mask of the system reset interrupts to enable.

FSL_RCM_DRIVER_VERSION

RCM driver version 2.0.4.

enum _rcm_reset_source

System Reset Source Name definitions.

Values:

enumerator kRCM_SourceWakeup

Low-leakage wakeup reset

enumerator kRCM_SourceLvd

Low-voltage detect reset

enumerator kRCM_SourceLoc

Loss of clock reset

enumerator kRCM_SourceLol

Loss of lock reset

enumerator kRCM_SourceWdog

Watchdog reset

enumerator kRCM_SourcePin

External pin reset

enumerator kRCM_SourcePor

Power on reset

enumerator kRCM_SourceJtag

JTAG generated reset

enumerator kRCM_SourceLockup

Core lock up reset

enumerator kRCM_SourceSw

Software reset

enumerator kRCM_SourceMdmap

MDM-AP system reset

enumerator kRCM_SourceEzpt

EzPort reset

enumerator kRCM_SourceSackerr

Parameter could get all reset flags

enumerator kRCM_SourceAll

enum _rcm_run_wait_filter_mode

Reset pin filter select in Run and Wait modes.

Values:

enumerator kRCM_FilterDisable

All filtering disabled

enumerator kRCM_FilterBusClock
Bus clock filter enabled

enumerator kRCM_FilterLpoClock
LPO clock filter enabled

enum _rcm_boot_rom_config
Boot from ROM configuration.

Values:

enumerator kRCM_BootFlash
Boot from flash

enumerator kRCM_BootRomCfg0
Boot from boot ROM due to BOOTCFG0

enumerator kRCM_BootRomFopt
Boot from boot ROM due to FOPT[7]

enumerator kRCM_BootRomBoth
Boot from boot ROM due to both BOOTCFG0 and FOPT[7]

enum _rcm_reset_delay
Maximum delay time from interrupt asserts to system reset.

Values:

enumerator kRCM_ResetDelay8Lpo
Delay 8 LPO cycles.

enumerator kRCM_ResetDelay32Lpo
Delay 32 LPO cycles.

enumerator kRCM_ResetDelay128Lpo
Delay 128 LPO cycles.

enumerator kRCM_ResetDelay512Lpo
Delay 512 LPO cycles.

enum _rcm_interrupt_enable
System reset interrupt enable bit definitions.

Values:

enumerator kRCM_IntNone
No interrupt enabled.

enumerator kRCM_IntLossOfClk
Loss of clock interrupt.

enumerator kRCM_IntLossOfLock
Loss of lock interrupt.

enumerator kRCM_IntWatchDog
Watch dog interrupt.

enumerator kRCM_IntExternalPin
External pin interrupt.

enumerator kRCM_IntGlobal
Global interrupts.

enumerator kRCM_IntCoreLockup
Core lock up interrupt

enumerator kRCM_IntSoftware
software interrupt

enumerator kRCM_IntStopModeAckErr
Stop mode ACK error interrupt.

enumerator kRCM_IntCore1
Core 1 interrupt.

enumerator kRCM_IntAll
Enable all interrupts.

typedef enum *_rcm_reset_source* rcm_reset_source_t
System Reset Source Name definitions.

typedef enum *_rcm_run_wait_filter_mode* rcm_run_wait_filter_mode_t
Reset pin filter select in Run and Wait modes.

typedef enum *_rcm_boot_rom_config* rcm_boot_rom_config_t
Boot from ROM configuration.

typedef enum *_rcm_reset_delay* rcm_reset_delay_t
Maximum delay time from interrupt asserts to system reset.

typedef enum *_rcm_interrupt_enable* rcm_interrupt_enable_t
System reset interrupt enable bit definitions.

typedef struct *_rcm_version_id* rcm_version_id_t
IP version ID definition.

typedef struct *_rcm_reset_pin_filter_config* rcm_reset_pin_filter_config_t
Reset pin filter configuration.

struct _rcm_version_id
#include <fsl_rcm.h> IP version ID definition.

Public Members

uint16_t feature
Feature Specification Number.

uint8_t minor
Minor version number.

uint8_t major
Major version number.

struct _rcm_reset_pin_filter_config
#include <fsl_rcm.h> Reset pin filter configuration.

Public Members

bool enableFilterInStop
Reset pin filter select in stop mode.

rcm_run_wait_filter_mode_t filterInRunWait
Reset pin filter in run/wait mode.

uint8_t busClockFilterCount

Reset pin bus clock filter width.

2.37 RTC: Real Time Clock

void RTC_Init(RTC_Type *base, const *rtc_config_t* *config)

Ungates the RTC clock and configures the peripheral for basic operation.

This function issues a software reset if the timer invalid flag is set.

Note: This API should be called at the beginning of the application using the RTC driver.

Parameters

- base – RTC peripheral base address
- config – Pointer to the user's RTC configuration structure.

static inline void RTC_Deinit(RTC_Type *base)

Stops the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

void RTC_GetDefaultConfig(*rtc_config_t* *config)

Fills in the RTC config struct with the default settings.

The default values are as follows.

```
config->clockOutput = false;
config->wakeupSelect = false;
config->updateMode = false;
config->supervisorAccess = false;
config->compensationInterval = 0;
config->compensationTime = 0;
```

Parameters

- config – Pointer to the user's RTC configuration structure.

status_t RTC_SetDatetime(RTC_Type *base, const *rtc_datetime_t* *datetime)

Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

- base – RTC 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 RTC kStatus_InvalidArgument: Error because the datetime format is incorrect void RTC_GetDatetime(RTC_Type *base, *rtc_datetime_t* *datetime)

Gets the RTC time and stores it in the given time structure.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t RTC_SetAlarm(RTC_Type *base, const *rtc_datetime_t* *alarmTime)

Sets the RTC alarm time.

The function 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 – RTC peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

void RTC_GetAlarm(RTC_Type *base, *rtc_datetime_t* *datetime)

Returns the RTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected RTC interrupts.

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration rtc_interrupt_enable_t

void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration rtc_interrupt_enable_t

uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled RTC interrupts.

Parameters

- base – RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration rtc_interrupt_enable_t uint32_t RTC_GetStatusFlags(RTC_Type *base)

Gets the RTC status flags.

Parameters

- base – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration rtc_status_flags_t

void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clears the RTC status flags.

Parameters

- base – RTC peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration rtc_status_flags_t

static inline void RTC_EnableLPOClock(RTC_Type *base, bool enable)

Enable/Disable RTC 1kHz LPO clock.

Note: After setting this bit, RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable RTC 1kHz LPO clock

static inline void RTC_StartTimer(RTC_Type *base)

Starts the RTC time counter.

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

- base – RTC peripheral base address

static inline void RTC_StopTimer(RTC_Type *base)

Stops the RTC time counter.

RTC's seconds register can be written to only when the timer is stopped.

Parameters

- base – RTC peripheral base address

void RTC_GetMonotonicCounter(RTC_Type *base, uint64_t *counter)

Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.

Parameters

- base – RTC peripheral base address
- counter – Pointer to variable where the value is stored.

void RTC_SetMonotonicCounter(RTC_Type *base, uint64_t counter)

Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value. The Monotonic Overflow Flag in RTC_SR is cleared due to the API.

Parameters

- base – RTC peripheral base address
- counter – Counter value

status_t RTC_IncrementMonotonicCounter(RTC_Type *base)

Increments the Monotonic Counter by one.

Increments the Monotonic Counter (registers RTC_MCLR and RTC_MCHR accordingly) by setting the monotonic counter enable (MER[MCE]) and then writing to the RTC_MCLR register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

Parameters

- base – RTC peripheral base address

Returns

kStatus_Success: success kStatus_Fail: error occurred, either time invalid or monotonic overflow flag was found

FSL_RTC_DRIVER_VERSION

Version 2.4.0

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_TimeInvalidInterruptEnable

Time invalid interrupt.

enumerator kRTC_TimeOverflowInterruptEnable

Time overflow interrupt.

enumerator kRTC_AlarmInterruptEnable

Alarm interrupt.

enumerator kRTC_MonotonicOverflowInterruptEnable

Monotonic Overflow Interrupt Enable

enumerator kRTC_SecondsInterruptEnable

Seconds interrupt.

enumerator kRTC_TestModeInterruptEnable

enumerator kRTC_FlashSecurityInterruptEnable

enumerator kRTC_TamperPinInterruptEnable

enumerator kRTC_SecurityModuleInterruptEnable

enumerator kRTC_LossOfClockInterruptEnable

enum _rtc_status_flags

List of RTC flags.

Values:

enumerator kRTC_TimeInvalidFlag

Time invalid flag

enumerator kRTC_TimeOverflowFlag

Time overflow flag

enumerator kRTC_AlarmFlag

Alarm flag

enumerator kRTC_MonotonicOverflowFlag
Monotonic Overflow Flag

enumerator kRTC_TamperInterruptDetectFlag
Tamper interrupt detect flag

enumerator kRTC_TestModeFlag

enumerator kRTC_FlashSecurityFlag

enumerator kRTC_TamperPinFlag

enumerator kRTC_SecurityTamperFlag

enumerator kRTC_LossOfClockTamperFlag

enum _rtc_osc_cap_load
List of RTC Oscillator capacitor load settings.

Values:

enumerator kRTC_Capacitor_2p
2 pF capacitor load

enumerator kRTC_Capacitor_4p
4 pF capacitor load

enumerator kRTC_Capacitor_8p
8 pF capacitor load

enumerator kRTC_Capacitor_16p
16 pF capacitor load

enum _rtc_timer_seconds_interrupt_frequency
List of RTC Timer Seconds Interrupt Frequencies.

Values:

enumerator kRTC_TimerSecondsFrequency1Hz
Timer seconds frequency is 1Hz

enumerator kRTC_TimerSecondsFrequency2Hz
Timer seconds frequency is 2Hz

enumerator kRTC_TimerSecondsFrequency4Hz
Timer seconds frequency is 4Hz

enumerator kRTC_TimerSecondsFrequency8Hz
Timer seconds frequency is 8Hz

enumerator kRTC_TimerSecondsFrequency16Hz
Timer seconds frequency is 16Hz

enumerator kRTC_TimerSecondsFrequency32Hz
Timer seconds frequency is 32Hz

enumerator kRTC_TimerSecondsFrequency64Hz
Timer seconds frequency is 64Hz

enumerator kRTC_TimerSecondsFrequency128Hz
Timer seconds frequency is 128Hz

typedef enum *_rtc_interrupt_enable* rtc_interrupt_enable_t
List of RTC interrupts.

typedef enum *_rtc_status_flags* rtc_status_flags_t
: List of RTC flags.

typedef enum *_rtc_osc_cap_load* rtc_osc_cap_load_t
: List of RTC Oscillator capacitor load settings.

typedef enum *_rtc_timer_seconds_interrupt_frequency* rtc_timer_seconds_interrupt_frequency_t
: List of RTC Timer Seconds Interrupt Frequencies.

typedef struct *_rtc_datetime* rtc_datetime_t
: Structure is used to hold the date and time.

typedef struct *_rtc_pin_config* rtc_pin_config_t
: RTC pin config structure.

typedef struct *_rtc_config* rtc_config_t
: RTC config structure.

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

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

static inline uint32_t RTC_GetTamperTimeSeconds(RTC_Type *base)
: Get the RTC tamper time seconds.

  Parameters

  - base – RTC peripheral base address

static inline void RTC_SetOscCapLoad(RTC_Type *base, uint32_t capLoad)
: This function sets the specified capacitor configuration for the RTC oscillator.

  Parameters

  - base – RTC peripheral base address
  - capLoad – Oscillator loads to enable. This is a logical OR of members of the enumeration rtc_osc_cap_load_t

static inline void RTC_Reset(RTC_Type *base)
: Performs a software reset on the RTC module.

  This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

  Parameters

  - base – RTC peripheral base address

static inline void RTC_EnableWakeUpPin(RTC_Type *base, bool enable)
: Enables or disables the RTC Wakeup Pin Operation.

  This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

  Parameters

  - base – RTC_Type base pointer.
  - enable – true to enable, false to disable.

static inline void RTC_EnableClockOutput(RTC_Type *base, bool enable)
: Enables or disables the RTC 32 kHz clock output.

  This function enables or disables the RTC 32 kHz clock output.

  Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

void RTC_SetTimerSecondsInterruptFrequency(RTC_Type *base, *rtc_timer_seconds_interrupt_frequency_t* freq)

Sets the RTC timer seconds interrupt frequency.

This function sets the RTC timer seconds interrupt frequency.

Parameters

- base – RTC peripheral base address
- freq – The timer seconds interrupt frequency. This is a member of the enumeration rtc_timer_seconds_interrupt_frequency_t

struct _rtc_datetime

#include <fsl_rtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _rtc_pin_config

#include <fsl_rtc.h> RTC pin config structure.

Public Members

bool inputLogic

true: Tamper pin input data is logic one. false: Tamper pin input data is logic zero.

bool pinActiveLow

true: Tamper pin is active low. false: Tamper pin is active high.

bool filterEnable

true: Input filter is enabled on the tamper pin. false: Input filter is disabled on the tamper pin.

bool pullSelectNegate

true: Tamper pin pull resistor direction will negate the tamper pin. false: Tamper pin pull resistor direction will assert the tamper pin.

bool pullEnable

true: Pull resistor is enabled on tamper pin. false: Pull resistor is disabled on tamper pin.

struct _rtc_config

#include <fsl_rtc.h> RTC config structure.

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

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

Public Members

bool clockOutput

true: The 32 kHz clock is not output to other peripherals; false: The 32 kHz clock is output to other peripherals

bool wakeupSelect

true: Wakeup pin outputs the 32 KHz clock; false:Wakeup pin used to wakeup the chip

bool updateMode

true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked

bool supervisorAccess

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported

uint32_t compensationInterval

Compensation interval that is written to the CIR field in RTC TCR Register

uint32_t compensationTime

Compensation time that is written to the TCR field in RTC TCR Register

2.38 SIM: System Integration Module Driver

FSL_SIM_DRIVER_VERSION

Driver version.

typedef struct *_sim_uid* sim_uid_t

Unique ID.

void SIM_GetUniqueId(*sim_uid_t* *uid)

Gets the unique identification register value.

Parameters

- uid – Pointer to the structure to save the UID value.

struct _sim_uid

#include <fsl_sim.h> Unique ID.

Public Members

uint32_t MH

UIDMH.

uint32_t ML

UIDML.

uint32_t L

UIDL.

2.39 SMC: System Mode Controller Driver

static inline void SMC_GetVersionId(SMC_Type *base, *smc_version_id_t* *versionId)

Gets the SMC version ID.

This function gets the SMC version ID, including major version number, minor version number, and feature specification number.

Parameters

- base – SMC peripheral base address.
- versionId – Pointer to the version ID structure.

void SMC_GetParam(SMC_Type *base, *smc_param_t* *param)

Gets the SMC parameter.

This function gets the SMC parameter including the enabled power mdoes.

Parameters

- base – SMC peripheral base address.
- param – Pointer to the SMC param structure.

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_AllowPowerModeVlls | kSMC_AllowPowerModeVlps). 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 status. After the application switches the power mode, it should always check the status to check whether it runs into the specified mode or not. The 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 status.

Parameters

- base – SMC peripheral base address.

Returns

Current power mode status.

void SMC_PreEnterStopModes(void)

Prepares to enter stop modes.

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

void SMC_PostExitStopModes(void)

Recovers after wake up from stop modes.

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with SMC_PreEnterStopModes.

void SMC_PreEnterWaitModes(void)

Prepares to enter wait modes.

This function should be called before entering WAIT/VLPW modes.

void SMC_PostExitWaitModes(void)

Recovers after wake up from stop modes.

This function should be called after wake up from WAIT/VLPW modes. It is used with SMC_PreEnterWaitModes.

status_t SMC_SetPowerModeRun(SMC_Type *base)

Configures the system to RUN power mode.

Parameters

- base – SMC peripheral base address.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeHsrun(SMC_Type *base)

Configures the system to HSRUN power mode.

Parameters

- base – SMC peripheral base address.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeWait(SMC_Type *base)

Configures 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)

Configures 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, bool wakeupMode)

Configures the system to VLPR power mode.

Parameters

- base – SMC peripheral base address.
- wakeupMode – Enter Normal Run mode if true, else stay in VLPR mode.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeVlpw(SMC_Type *base)

Configures the system to VLPW power mode.

Parameters

- base – SMC peripheral base address.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeVlps(SMC_Type *base)

Configures the system to VLPS power mode.

Parameters

- base – SMC peripheral base address.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeLls(SMC_Type *base, const *smc_power_mode_lls_config_t* *config)

Configures the system to LLS power mode.

Parameters

- base – SMC peripheral base address.
- config – The LLS power mode configuration structure

Returns

SMC configuration error code.

status_t SMC_SetPowerModeVlls(SMC_Type *base, const *smc_power_mode_vlls_config_t* *config)

Configures the system to VLLS power mode.

Parameters

- base – SMC peripheral base address.
- config – The VLLS power mode configuration structure.

Returns

SMC configuration error code.

FSL_SMC_DRIVER_VERSION

SMC 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 STOP

  enumerator kSMC_PowerStateVlpr
  : 0000_0100 - Current power mode is VLPR

  enumerator kSMC_PowerStateVlpw
  : 0000_1000 - Current power mode is VLPW

  enumerator kSMC_PowerStateVlps
  : 0001_0000 - Current power mode is VLPS

  enumerator kSMC_PowerStateLls
  : 0010_0000 - Current power mode is LLS

  enumerator kSMC_PowerStateVlls
  : 0100_0000 - Current power mode is VLLS

  enumerator kSMC_PowerStateHsrun
  : 1000_0000 - Current power mode is HSRUN

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_StopVlls
  : Very-low-leakage Stop mode.

enum _smc_stop_submode
: VLLS/LLS stop sub mode definition.

  Values:

enumerator kSMC_StopSub0
: Stop submode 0, for VLLS0/LLS0.

enumerator kSMC_StopSub1
: Stop submode 1, for VLLS1/LLS1.

enumerator kSMC_StopSub2
: Stop submode 2, for VLLS2/LLS2.

enumerator kSMC_StopSub3
: Stop submode 3, for VLLS3/LLS3.

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

_smc_status, SMC configuration status.

Values:

enumerator kStatus_SMC_StopAbort
: Entering Stop mode is abort

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_run_mode* smc_run_mode_t
: Run mode definition.

typedef enum *_smc_stop_mode* smc_stop_mode_t
: Stop mode definition.

typedef enum *_smc_stop_submode* smc_stop_submode_t
: VLLS/LLS stop sub mode definition.

typedef enum *_smc_partial_stop_mode* smc_partial_stop_option_t
: Partial STOP option.

typedef struct *_smc_version_id* smc_version_id_t
: IP version ID definition.

typedef struct *_smc_param* smc_param_t
: IP parameter definition.

typedef struct *_smc_power_mode_lls_config* smc_power_mode_lls_config_t
: SMC Low-Leakage Stop power mode configuration.

typedef struct *_smc_power_mode_vlls_config* smc_power_mode_vlls_config_t
: SMC Very Low-Leakage Stop power mode configuration.

struct _smc_version_id
: *#include <fsl_smc.h>* IP version ID definition.

Public Members

uint16_t feature
: Feature Specification Number.

uint8_t minor
: Minor version number.

uint8_t major
: Major version number.

struct _smc_param
: *#include <fsl_smc.h>* IP parameter definition.

Public Members

bool hsrunEnable
: HSRUN mode enable.

bool llsEnable
: LLS mode enable.

bool lls2Enable
: LLS2 mode enable.

bool vlls0Enable
: VLLS0 mode enable.

struct _smc_power_mode_lls_config
: *#include <fsl_smc.h>* SMC Low-Leakage Stop power mode configuration.

Public Members

smc_stop_submode_t subMode
: Low-leakage Stop sub-mode

bool enableLpoClock
: Enable LPO clock in LLS mode

struct _smc_power_mode_vlls_config
: *#include <fsl_smc.h>* SMC Very Low-Leakage Stop power mode configuration.

Public Members

smc_stop_submode_t subMode
: Very Low-leakage Stop sub-mode

bool enablePorDetectInVlls0
: Enable Power on reset detect in VLLS mode

bool enableRam2InVlls2
: Enable RAM2 power in VLLS2

bool enableLpoClock
: Enable LPO clock in VLLS mode

2.40 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.

2.41 Tsi_v5_driver

enum _tsi_main_clock_selection
    TSI main clock selection.

    These constants set the tsi main clock.

    Values:

    enumerator kTSI_MainClockSlection_0
        Set TSI main clock frequency to 20.72MHz

    enumerator kTSI_MainClockSlection_1
        Set TSI main clock frequency to 16.65MHz

    enumerator kTSI_MainClockSlection_2
        Set TSI main clock frequency to 13.87MHz

    enumerator kTSI_MainClockSlection_3
        Set TSI main clock frequency to 11.91MHz

enum _tsi_sensing_mode_selection
    TSI sensing mode selection.

    These constants set the tsi sensing mode.

    Values:

    enumerator kTSI_SensingModeSlection_Self
        Set TSI sensing mode to self-cap mode

    enumerator kTSI_SensingModeSlection_Mutual
        Set TSI sensing mode to mutual-cap mode

enum _tsi_dvolt_option
    TSI DVOLT settings.

    These bits indicate the comparator vp, vm and dvolt voltage.

    Values:

    enumerator kTSI_DvoltOption_0
        DVOLT value option 0, the value may differ on different platforms

    enumerator kTSI_DvoltOption_1
        DVOLT value option 1, the value may differ on different platforms

    enumerator kTSI_DvoltOption_2
        DVOLT value option 2, the value may differ on different platforms

    enumerator kTSI_DvoltOption_3
        DVOLT value option 3, the value may differ on different platforms

enum _tsi_sensitivity_xdn_option

TSI sensitivity ajustment (XDN option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

Values:

enumerator kTSI_SensitivityXdnOption_0

Adjust sensitivity in self-cap mode, 1/16

enumerator kTSI_SensitivityXdnOption_1

Adjust sensitivity in self-cap mode, 1/8

enumerator kTSI_SensitivityXdnOption_2

Adjust sensitivity in self-cap mode, 1/4

enumerator kTSI_SensitivityXdnOption_3

Adjust sensitivity in self-cap mode, 1/2

enum _tsi_shield

TSI Shield setting (S_W_SHIELD option).

These constants define the shield pin used for HW shielding functionality. One or more shield pin can be selected. The involved bitfield is not fix can change from device to device (KE16Z7 and KE17Z7 support 3 shield pins, other KE serials only support 1 shield pin).

Values:

enumerator kTSI_shieldAllOff

No pin used

enumerator kTSI_shield0On

Shield 0 pin used

enumerator kTSI_shield1On

Shield 1 pin used

enumerator kTSI_shield1and0On

Shield 0,1 pins used

enumerator kTSI_shield2On

Shield 2 pin used

enumerator kTSI_shield2and0On

Shield 2,0 pins used

enumerator kTSI_shield2and1On

Shield 2,1 pins used

enumerator kTSI_shieldAllOn

Shield 2,1,0 pins used

enum _tsi_sensitivity_ctrim_option

TSI sensitivity ajustment (CTRIM option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

Values:

enumerator kTSI_SensitivityCtrimOption_0

Adjust sensitivity in self-cap mode, 2.5p

enumerator kTSI_SensitivityCtrimOption_1
Adjust sensitivity in self-cap mode, 5.0p

enumerator kTSI_SensitivityCtrimOption_2
Adjust sensitivity in self-cap mode, 7.5p

enumerator kTSI_SensitivityCtrimOption_3
Adjust sensitivity in self-cap mode, 10.0p

enumerator kTSI_SensitivityCtrimOption_4
Adjust sensitivity in self-cap mode, 12.5p

enumerator kTSI_SensitivityCtrimOption_5
Adjust sensitivity in self-cap mode, 15.0p

enumerator kTSI_SensitivityCtrimOption_6
Adjust sensitivity in self-cap mode, 17.5p

enumerator kTSI_SensitivityCtrimOption_7
Adjust sensitivity in self-cap mode, 20.0p

enum _tsi_current_multiple_input
TSI current ajustment (Input current multiple).

These constants set the tsi input current multiple in self-cap mode.

Values:

enumerator kTSI_CurrentMultipleInputValue_0
Adjust input current multiple in self-cap mode, 1/8

enumerator kTSI_CurrentMultipleInputValue_1
Adjust input current multiple in self-cap mode, 1/4

enum _tsi_current_multiple_charge
TSI current ajustment (Charge/Discharge current multiple).

These constants set the tsi charge/discharge current multiple in self-cap mode.

Values:

enumerator kTSI_CurrentMultipleChargeValue_0
Adjust charge/discharge current multiple in self-cap mode, 1/16

enumerator kTSI_CurrentMultipleChargeValue_1
Adjust charge/discharge current multiple in self-cap mode, 1/8

enumerator kTSI_CurrentMultipleChargeValue_2
Adjust charge/discharge current multiple in self-cap mode, 1/4

enumerator kTSI_CurrentMultipleChargeValue_3
Adjust charge/discharge current multiple in self-cap mode, 1/2

enumerator kTSI_CurrentMultipleChargeValue_4
Adjust charge/discharge current multiple in self-cap mode, 1/1

enumerator kTSI_CurrentMultipleChargeValue_5
Adjust charge/discharge current multiple in self-cap mode, 2/1

enumerator kTSI_CurrentMultipleChargeValue_6
Adjust charge/discharge current multiple in self-cap mode, 4/1

enumerator kTSI_CurrentMultipleChargeValue_7
Adjust charge/discharge current multiple in self-cap mode, 8/1

enum _tsi_mutual_pre_current
TSI current used in vref generator.

These constants Choose the current used in vref generator.

Values:

enumerator kTSI_MutualPreCurrent_1uA
Vref generator current is 1uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_2uA
Vref generator current is 2uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_3uA
Vref generator current is 3uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_4uA
Vref generator current is 4uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_5uA
Vref generator current is 5uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_6uA
Vref generator current is 6uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_7uA
Vref generator current is 7uA, used in mutual-cap mode

enumerator kTSI_MutualPreCurrent_8uA
Vref generator current is 8uA, used in mutual-cap mode

enum _tsi_mutual_pre_resistor
TSI resistor used in pre-charge.

These constants Choose the resistor used in pre-charge.

Values:

enumerator kTSI_MutualPreResistor_1k
Vref generator resistor is 1k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_2k
Vref generator resistor is 2k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_3k
Vref generator resistor is 3k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_4k
Vref generator resistor is 4k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_5k
Vref generator resistor is 5k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_6k
Vref generator resistor is 6k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_7k
Vref generator resistor is 7k, used in mutual-cap mode

enumerator kTSI_MutualPreResistor_8k
Vref generator resistor is 8k, used in mutual-cap mode

enum _tsi_mutual_sense_resistor

TSI resistor used in I-sense generator.

These constants Choose the resistor used in I-sense generator.

Values:

enumerator kTSI_MutualSenseResistor_2k5
I-sense resistor is 2.5k , used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_5k
I-sense resistor is 5.0k , used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_7k5
I-sense resistor is 7.5k , used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_10k
I-sense resistor is 10.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_12k5
I-sense resistor is 12.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_15k
I-sense resistor is 15.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_17k5
I-sense resistor is 17.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_20k
I-sense resistor is 20.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_22k5
I-sense resistor is 22.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_25k
I-sense resistor is 25.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_27k5
I-sense resistor is 27.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_30k
I-sense resistor is 30.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_32k5
I-sense resistor is 32.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_35k
I-sense resistor is 35.0k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_37k5
I-sense resistor is 37.5k, used in mutual-cap mode

enumerator kTSI_MutualSenseResistor_40k
I-sense resistor is 40.0k, used in mutual-cap mode

enum _tsi_mutual_tx_channel

TSI TX channel selection in mutual-cap mode.

These constants Choose the TX channel used in mutual-cap mode.

Values:

enumerator kTSI_MutualTxChannel_0
Select channel 0 as tx0, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_1
Select channel 1 as tx1, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_2
Select channel 2 as tx2, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_3
Select channel 3 as tx3, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_4
Select channel 4 as tx4, used in mutual-cap mode

enumerator kTSI_MutualTxChannel_5
Select channel 5 as tx5, used in mutual-cap mode

enum _tsi_mutual_rx_channel
TSI RX channel selection in mutual-cap mode.

These constants Choose the RX channel used in mutual-cap mode.

Values:

enumerator kTSI_MutualRxChannel_6
Select channel 6 as rx6, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_7
Select channel 7 as rx7, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_8
Select channel 8 as rx8, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_9
Select channel 9 as rx9, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_10
Select channel 10 as rx10, used in mutual-cap mode

enumerator kTSI_MutualRxChannel_11
Select channel 11 as rx11, used in mutual-cap mode

enum _tsi_mutual_sense_boost_current
TSI sensitivity boost current settings.

These constants set the sensitivity boost current.

Values:

enumerator kTSI_MutualSenseBoostCurrent_0uA
Sensitivity boost current is 0uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_2uA
Sensitivity boost current is 2uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_4uA
Sensitivity boost current is 4uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_6uA
Sensitivity boost current is 6uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_8uA
Sensitivity boost current is 8uA , used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_10uA
Sensitivity boost current is 10uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_12uA
Sensitivity boost current is 12uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_14uA
Sensitivity boost current is 14uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_16uA
Sensitivity boost current is 16uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_18uA
Sensitivity boost current is 18uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_20uA
Sensitivity boost current is 20uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_22uA
Sensitivity boost current is 22uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_24uA
Sensitivity boost current is 24uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_26uA
Sensitivity boost current is 26uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_28uA
Sensitivity boost current is 28uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_30uA
Sensitivity boost current is 30uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_32uA
Sensitivity boost current is 32uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_34uA
Sensitivity boost current is 34uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_36uA
Sensitivity boost current is 36uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_38uA
Sensitivity boost current is 38uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_40uA
Sensitivity boost current is 40uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_42uA
Sensitivity boost current is 42uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_44uA
Sensitivity boost current is 44uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_46uA
Sensitivity boost current is 46uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_48uA
Sensitivity boost current is 48uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_50uA
Sensitivity boost current is 50uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_52uA
Sensitivity boost current is 52uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_54uA
Sensitivity boost current is 54uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_56uA
Sensitivity boost current is 56uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_58uA
Sensitivity boost current is 58uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_60uA
Sensitivity boost current is 60uA, used in mutual-cap mode

enumerator kTSI_MutualSenseBoostCurrent_62uA
Sensitivity boost current is 62uA, used in mutual-cap mode

enum _tsi_mutual_tx_drive_mode
TSI TX drive mode control.

These constants Choose the TX drive mode control setting.

Values:

enumerator kTSI_MutualTxDriveModeOption_0
TX drive mode is -5v ~ +5v, used in mutual-cap mode

enumerator kTSI_MutualTxDriveModeOption_1
TX drive mode is 0v ~ +5v, used in mutual-cap mode

enum _tsi_mutual_pmos_current_left
TSI Pmos current mirror selection on the left side.

These constants set the Pmos current mirror on the left side used in mutual-cap mode.

Values:

enumerator kTSI_MutualPmosCurrentMirrorLeft_4
Set Pmos current mirror left value as 4, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_8
Set Pmos current mirror left value as 8, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_12
Set Pmos current mirror left value as 12, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_16
Set Pmos current mirror left value as 16, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_20
Set Pmos current mirror left value as 20, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_24
Set Pmos current mirror left value as 24, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_28
Set Pmos current mirror left value as 28, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorLeft_32
Set Pmos current mirror left value as 32, used in mutual-cap mode

enum _tsi_mutual_pmos_current_right
TSI Pmos current mirror selection on the right side.

These constants set the Pmos current mirror on the right side used in mutual-cap mode.

Values:

enumerator kTSI_MutualPmosCurrentMirrorRight_1
Set Pmos current mirror right value as 1, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorRight_2
Set Pmos current mirror right value as 2, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorRight_3
Set Pmos current mirror right value as 3, used in mutual-cap mode

enumerator kTSI_MutualPmosCurrentMirrorRight_4
Set Pmos current mirror right value as 4, used in mutual-cap mode

enum _tsi_mutual_nmos_current
TSI Nmos current mirror selection.

These constants set the Nmos current mirror used in mutual-cap mode.

Values:

enumerator kTSI_MutualNmosCurrentMirror_1
Set Nmos current mirror value as 1, used in mutual-cap mode

enumerator kTSI_MutualNmosCurrentMirror_2
Set Nmos current mirror value as 2, used in mutual-cap mode

enumerator kTSI_MutualNmosCurrentMirror_3
Set Nmos current mirror value as 3, used in mutual-cap mode

enumerator kTSI_MutualNmosCurrentMirror_4
Set Nmos current mirror value as 4, used in mutual-cap mode

enum _tsi_sinc_cutoff_div
TSI SINC cutoff divider setting.

These bits set the SINC cutoff divider.

Values:

enumerator kTSI_SincCutoffDiv_1
Set SINC cutoff divider as 1

enumerator kTSI_SincCutoffDiv_2
Set SINC cutoff divider as 2

enumerator kTSI_SincCutoffDiv_4
Set SINC cutoff divider as 4

enumerator kTSI_SincCutoffDiv_8
Set SINC cutoff divider as 8

enumerator kTSI_SincCutoffDiv_16
Set SINC cutoff divider as 16

enumerator kTSI_SincCutoffDiv_32
Set SINC cutoff divider as 32

enumerator kTSI_SincCutoffDiv_64
Set SINC cutoff divider as 64

enumerator kTSI_SincCutoffDiv_128
Set SINC cutoff divider as 128

enum _tsi_sinc_filter_order

TSI SINC filter order setting.

These bits set the SINC filter order.

Values:

enumerator kTSI_SincFilterOrder_1

Use 1 order SINC filter

enumerator kTSI_SincFilterOrder_2

Use 1 order SINC filter

enum _tsi_sinc_decimation_value

TSI SINC decimation value setting.

These bits set the SINC decimation value.

Values:

enumerator kTSI_SincDecimationValue_1

The TSI_DATA[TSICH] bits is the counter value of 1 triger period.

enumerator kTSI_SincDecimationValue_2

The TSI_DATA[TSICH] bits is the counter value of 2 triger period.

enumerator kTSI_SincDecimationValue_3

The TSI_DATA[TSICH] bits is the counter value of 3 triger period.

enumerator kTSI_SincDecimationValue_4

The TSI_DATA[TSICH] bits is the counter value of 4 triger period.

enumerator kTSI_SincDecimationValue_5

The TSI_DATA[TSICH] bits is the counter value of 5 triger period.

enumerator kTSI_SincDecimationValue_6

The TSI_DATA[TSICH] bits is the counter value of 6 triger period.

enumerator kTSI_SincDecimationValue_7

The TSI_DATA[TSICH] bits is the counter value of 7 triger period.

enumerator kTSI_SincDecimationValue_8

The TSI_DATA[TSICH] bits is the counter value of 8 triger period.

enumerator kTSI_SincDecimationValue_9

The TSI_DATA[TSICH] bits is the counter value of 9 triger period.

enumerator kTSI_SincDecimationValue_10

The TSI_DATA[TSICH] bits is the counter value of 10 triger period.

enumerator kTSI_SincDecimationValue_11

The TSI_DATA[TSICH] bits is the counter value of 11 triger period.

enumerator kTSI_SincDecimationValue_12

The TSI_DATA[TSICH] bits is the counter value of 12 triger period.

enumerator kTSI_SincDecimationValue_13

The TSI_DATA[TSICH] bits is the counter value of 13 triger period.

enumerator kTSI_SincDecimationValue_14

The TSI_DATA[TSICH] bits is the counter value of 14 triger period.

enumerator kTSI_SincDecimationValue_15

The TSI_DATA[TSICH] bits is the counter value of 15 triger period.

enumerator kTSI_SincDecimationValue_16
The TSI_DATA[TSICH] bits is the counter value of 16 triger period.

enumerator kTSI_SincDecimationValue_17
The TSI_DATA[TSICH] bits is the counter value of 17 triger period.

enumerator kTSI_SincDecimationValue_18
The TSI_DATA[TSICH] bits is the counter value of 18 triger period.

enumerator kTSI_SincDecimationValue_19
The TSI_DATA[TSICH] bits is the counter value of 19 triger period.

enumerator kTSI_SincDecimationValue_20
The TSI_DATA[TSICH] bits is the counter value of 20 triger period.

enumerator kTSI_SincDecimationValue_21
The TSI_DATA[TSICH] bits is the counter value of 21 triger period.

enumerator kTSI_SincDecimationValue_22
The TSI_DATA[TSICH] bits is the counter value of 22 triger period.

enumerator kTSI_SincDecimationValue_23
The TSI_DATA[TSICH] bits is the counter value of 23 triger period.

enumerator kTSI_SincDecimationValue_24
The TSI_DATA[TSICH] bits is the counter value of 24 triger period.

enumerator kTSI_SincDecimationValue_25
The TSI_DATA[TSICH] bits is the counter value of 25 triger period.

enumerator kTSI_SincDecimationValue_26
The TSI_DATA[TSICH] bits is the counter value of 26 triger period.

enumerator kTSI_SincDecimationValue_27
The TSI_DATA[TSICH] bits is the counter value of 27 triger period.

enumerator kTSI_SincDecimationValue_28
The TSI_DATA[TSICH] bits is the counter value of 28 triger period.

enumerator kTSI_SincDecimationValue_29
The TSI_DATA[TSICH] bits is the counter value of 29 triger period.

enumerator kTSI_SincDecimationValue_30
The TSI_DATA[TSICH] bits is the counter value of 30 triger period.

enumerator kTSI_SincDecimationValue_31
The TSI_DATA[TSICH] bits is the counter value of 31 triger period.

enumerator kTSI_SincDecimationValue_32
The TSI_DATA[TSICH] bits is the counter value of 32 triger period.

enum _tsi_ssc_charge_num
TSI SSC output bit0's period setting(SSC0[CHARGE_NUM])

These bits set the SSC output bit0's period setting.

Values:

enumerator kTSI_SscChargeNumValue_1
The SSC output bit 0's period will be 1 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_2
The SSC output bit 0's period will be 2 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_3
: The SSC output bit 0's period will be 3 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_4
: The SSC output bit 0's period will be 4 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_5
: The SSC output bit 0's period will be 5 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_6
: The SSC output bit 0's period will be 6 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_7
: The SSC output bit 0's period will be 7 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_8
: The SSC output bit 0's period will be 8 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_9
: The SSC output bit 0's period will be 9 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_10
: The SSC output bit 0's period will be 10 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_11
: The SSC output bit 0's period will be 11 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_12
: The SSC output bit 0's period will be 12 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_13
: The SSC output bit 0's period will be 13 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_14
: The SSC output bit 0's period will be 14 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_15
: The SSC output bit 0's period will be 15 clock cycle of system clock.

enumerator kTSI_SscChargeNumValue_16
: The SSC output bit 0's period will be 16 clock cycle of system clock.

enum _tsi_ssc_nocharge_num

TSI SSC output bit1's period setting(SSC0[BASE_NOCHARGE_NUM])

These bits set the SSC output bit1's period setting.

Values:

enumerator kTSI_SscNoChargeNumValue_1
: The SSC output bit 1's basic period will be 1 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_2
: The SSC output bit 1's basic period will be 2 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_3
: The SSC output bit 1's basic period will be 3 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_4
: The SSC output bit 1's basic period will be 4 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_5
: The SSC output bit 1's basic period will be 5 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_6
The SSC output bit 1's basic period will be 6 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_7
The SSC output bit 1's basic period will be 7 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_8
The SSC output bit 1's basic period will be 8 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_9
The SSC output bit 1's basic period will be 9 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_10
The SSC output bit 1's basic period will be 10 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_11
The SSC output bit 1's basic period will be 11 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_12
The SSC output bit 1's basic period will be 12 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_13
The SSC output bit 1's basic period will be 13 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_14
The SSC output bit 1's basic period will be 14 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_15
The SSC output bit 1's basic period will be 15 clock cycle of system clock.

enumerator kTSI_SscNoChargeNumValue_16
The SSC output bit 1's basic period will be 16 clock cycle of system clock.

enum _tsi_ssc_prbs_outsel
TSI SSC outsel choosing the length of the PRBS (Pseudo-RandomBinarySequence) method setting(SSC0[TSI_SSC0_PRBS_OUTSEL])

These bits set the SSC PRBS length.

Values:

enumerator kTSI_SscPrbsOutsel_2
The length of the PRBS is 2.

enumerator kTSI_SscPrbsOutsel_3
The length of the PRBS is 3.

enumerator kTSI_SscPrbsOutsel_4
The length of the PRBS is 4.

enumerator kTSI_SscPrbsOutsel_5
The length of the PRBS is 5.

enumerator kTSI_SscPrbsOutsel_6
The length of the PRBS is 6.

enumerator kTSI_SscPrbsOutsel_7
The length of the PRBS is 7.

enumerator kTSI_SscPrbsOutsel_8
The length of the PRBS is 8.

enumerator kTSI_SscPrbsOutsel_9
The length of the PRBS is 9.

enumerator kTSI_SscPrbsOutsel_10
    The length of the PRBS is 10.

enumerator kTSI_SscPrbsOutsel_11
    The length of the PRBS is 11.

enumerator kTSI_SscPrbsOutsel_12
    The length of the PRBS is 12.

enumerator kTSI_SscPrbsOutsel_13
    The length of the PRBS is 13.

enumerator kTSI_SscPrbsOutsel_14
    The length of the PRBS is 14.

enumerator kTSI_SscPrbsOutsel_15
    The length of the PRBS is 15.

enum _tsi_status_flags
    TSI status flags.

    Values:

    enumerator kTSI_EndOfScanFlag
        End-Of-Scan flag

    enumerator kTSI_OutOfRangeFlag
        Out-Of-Range flag

enum _tsi_interrupt_enable
    TSI feature interrupt source.

    Values:

    enumerator kTSI_GlobalInterruptEnable
        TSI module global interrupt

    enumerator kTSI_OutOfRangeInterruptEnable
        Out-Of-Range interrupt

    enumerator kTSI_EndOfScanInterruptEnable
        End-Of-Scan interrupt

enum _tsi_ssc_mode
    TSI SSC mode selection.

    These constants set the SSC mode.

    Values:

    enumerator kTSI_ssc_prbs_method
        Using PRBS method generating SSC output bit.

    enumerator kTSI_ssc_up_down_counter
        Using up-down counter generating SSC output bit.

    enumerator kTSI_ssc_disable
        SSC function is disabled.

enum _tsi_ssc_prescaler
    TSI main clock selection.

    These constants set select the divider ratio for the clock used for generating the SSC output bit.

    Values:

enumerator kTSI_ssc_div_by_1
: Set SSC divider to 00000000 div1(2^0)

enumerator kTSI_ssc_div_by_2
: Set SSC divider to 00000001 div2(2^1)

enumerator kTSI_ssc_div_by_4
: Set SSC divider to 00000011 div4(2^2)

enumerator kTSI_ssc_div_by_8
: Set SSC divider to 00000111 div8(2^3)

enumerator kTSI_ssc_div_by_16
: Set SSC divider to 00001111 div16(2^4)

enumerator kTSI_ssc_div_by_32
: Set SSC divider to 00011111 div32(2^5)

enumerator kTSI_ssc_div_by_64
: Set SSC divider to 00111111 div64(2^6)

enumerator kTSI_ssc_div_by_128
: Set SSC divider to 01111111 div128(2^7)

enumerator kTSI_ssc_div_by_256
: Set SSC divider to 11111111 div256(2^8)

typedef enum *_tsi_main_clock_selection* tsi_main_clock_selection_t

TSI main clock selection.

These constants set the tsi main clock.

typedef enum *_tsi_sensing_mode_selection* tsi_sensing_mode_selection_t

TSI sensing mode selection.

These constants set the tsi sensing mode.

typedef enum *_tsi_dvolt_option* tsi_dvolt_option_t

TSI DVOLT settings.

These bits indicate the comparator vp, vm and dvolt voltage.

typedef enum *_tsi_sensitivity_xdn_option* tsi_sensitivity_xdn_option_t

TSI sensitivity ajustment (XDN option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

typedef enum *_tsi_shield* tsi_shield_t

TSI Shield setting (S_W_SHIELD option).

These constants define the shield pin used for HW shielding functionality. One or more shield pin can be selected. The involved bitfield is not fix can change from device to device (KE16Z7 and KE17Z7 support 3 shield pins, other KE serials only support 1 shield pin).

typedef enum *_tsi_sensitivity_ctrim_option* tsi_sensitivity_ctrim_option_t

TSI sensitivity ajustment (CTRIM option).

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

typedef enum *_tsi_current_multiple_input* tsi_current_multiple_input_t

TSI current ajustment (Input current multiple).

These constants set the tsi input current multiple in self-cap mode.

typedef enum *_tsi_current_multiple_charge* tsi_current_multiple_charge_t

TSI current ajustment (Charge/Discharge current multiple).

These constants set the tsi charge/discharge current multiple in self-cap mode.

typedef enum *_tsi_mutual_pre_current* tsi_mutual_pre_current_t

TSI current used in vref generator.

These constants Choose the current used in vref generator.

typedef enum *_tsi_mutual_pre_resistor* tsi_mutual_pre_resistor_t

TSI resistor used in pre-charge.

These constants Choose the resistor used in pre-charge.

typedef enum *_tsi_mutual_sense_resistor* tsi_mutual_sense_resistor_t

TSI resistor used in I-sense generator.

These constants Choose the resistor used in I-sense generator.

typedef enum *_tsi_mutual_tx_channel* tsi_mutual_tx_channel_t

TSI TX channel selection in mutual-cap mode.

These constants Choose the TX channel used in mutual-cap mode.

typedef enum *_tsi_mutual_rx_channel* tsi_mutual_rx_channel_t

TSI RX channel selection in mutual-cap mode.

These constants Choose the RX channel used in mutual-cap mode.

typedef enum *_tsi_mutual_sense_boost_current* tsi_mutual_sense_boost_current_t

TSI sensitivity boost current settings.

These constants set the sensitivity boost current.

typedef enum *_tsi_mutual_tx_drive_mode* tsi_mutual_tx_drive_mode_t

TSI TX drive mode control.

These constants Choose the TX drive mode control setting.

typedef enum *_tsi_mutual_pmos_current_left* tsi_mutual_pmos_current_left_t

TSI Pmos current mirror selection on the left side.

These constants set the Pmos current mirror on the left side used in mutual-cap mode.

typedef enum *_tsi_mutual_pmos_current_right* tsi_mutual_pmos_current_right_t

TSI Pmos current mirror selection on the right side.

These constants set the Pmos current mirror on the right side used in mutual-cap mode.

typedef enum *_tsi_mutual_nmos_current* tsi_mutual_nmos_current_t

TSI Nmos current mirror selection.

These constants set the Nmos current mirror used in mutual-cap mode.

typedef enum *_tsi_sinc_cutoff_div* tsi_sinc_cutoff_div_t

TSI SINC cutoff divider setting.

These bits set the SINC cutoff divider.

typedef enum *_tsi_sinc_filter_order* tsi_sinc_filter_order_t

TSI SINC filter order setting.

These bits set the SINC filter order.

typedef enum *_tsi_sinc_decimation_value* tsi_sinc_decimation_value_t
: TSI SINC decimation value setting.

  These bits set the SINC decimation value.

typedef enum *_tsi_ssc_charge_num* tsi_ssc_charge_num_t
: TSI SSC output bit0's period setting(SSC0[CHARGE_NUM])

  These bits set the SSC output bit0's period setting.

typedef enum *_tsi_ssc_nocharge_num* tsi_ssc_nocharge_num_t
: TSI SSC output bit1's period setting(SSC0[BASE_NOCHARGE_NUM])

  These bits set the SSC output bit1's period setting.

typedef enum *_tsi_ssc_prbs_outsel* tsi_ssc_prbs_outsel_t
: TSI SSC outsel choosing the length of the PRBS (Pseudo-RandomBinarySequence) method setting(SSC0[TSI_SSC0_PRBS_OUTSEL])

  These bits set the SSC PRBS length.

typedef enum *_tsi_status_flags* tsi_status_flags_t
: TSI status flags.

typedef enum *_tsi_interrupt_enable* tsi_interrupt_enable_t
: TSI feature interrupt source.

typedef enum *_tsi_ssc_mode* tsi_ssc_mode_t
: TSI SSC mode selection.

  These constants set the SSC mode.

typedef enum *_tsi_ssc_prescaler* tsi_ssc_prescaler_t
: TSI main clock selection.

  These constants set select the divider ratio for the clock used for generating the SSC output bit.

typedef struct *_tsi_calibration_data* tsi_calibration_data_t
: TSI calibration data storage.

typedef struct *_tsi_common_config* tsi_common_config_t
: TSI common configuration structure.

  This structure contains the common settings for TSI self-cap or mutual-cap mode, configurations including the TSI module main clock, sensing mode, DVOLT options, SINC and SSC configurations.

typedef struct *_tsi_selfCap_config* tsi_selfCap_config_t
: TSI configuration structure for self-cap mode.

  This structure contains the settings for the most common TSI self-cap configurations including the TSI module charge currents, sensitivity configuration and so on.

typedef struct *_tsi_mutualCap_config* tsi_mutualCap_config_t
: TSI configuration structure for mutual-cap mode.

  This structure contains the settings for the most common TSI mutual-cap configurations including the TSI module generator settings, sensitivity related current settings and so on.

const *clock_ip_name_t* s_tsiClock[]

const IRQn_Type s_TsiIRQ[]

TSI_Type *const s_tsiBases[]

uint32_t TSI_GetInstance(TSI_Type *base)

Get the TSI instance from peripheral base address.

Parameters

- base – TSI peripheral base address.

Returns

TSI instance.

void TSI_InitSelfCapMode(TSI_Type *base, const *tsi_selfCap_config_t* *config)

Initialize hardware to Self-cap mode.

Initialize the peripheral to the targeted state specified by parameter config, such as sets sensitivity adjustment, current settings.

Parameters

- base – TSI peripheral base address.
- config – Pointer to TSI self-cap configuration structure.

Returns

none

void TSI_InitMutualCapMode(TSI_Type *base, const *tsi_mutualCap_config_t* *config)

Initialize hardware to Mutual-cap mode.

Initialize the peripheral to the targeted state specified by parameter config, such as sets Vref generator setting, sensitivity boost settings, Pmos/Nmos settings.

Parameters

- base – TSI peripheral base address.
- config – Pointer to TSI mutual-cap configuration structure.

Returns

none

void TSI_Deinit(TSI_Type *base)

De-initialize hardware.

De-initialize the peripheral to default state.

Parameters

- base – TSI peripheral base address.

Returns

none

void TSI_GetSelfCapModeDefaultConfig(*tsi_selfCap_config_t* *userConfig)

Get TSI self-cap mode user configure structure. This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to a value:

```
userConfig->commonConfig.mainClock      = kTSI_MainClockSlection_0;
userConfig->commonConfig.mode           = kTSI_SensingModeSlection_Self;
userConfig->commonConfig.dvolt          = kTSI_DvoltOption_2;
userConfig->commonConfig.cutoff         = kTSI_SincCutoffDiv_1;
userConfig->commonConfig.order          = kTSI_SincFilterOrder_1;
userConfig->commonConfig.decimation     = kTSI_SincDecimationValue_8;
userConfig->commonConfig.chargeNum      = kTSI_SscChargeNumValue_3;
userConfig->commonConfig.prbsOutsel     = kTSI_SscPrbsOutsel_2;
userConfig->commonConfig.noChargeNum    = kTSI_SscNoChargeNumValue_2;
userConfig->commonConfig.ssc_mode       = kTSI_ssc_prbs_method;
userConfig->commonConfig.ssc_prescaler = kTSI_ssc_div_by_1;
```

(continues on next page)

(continued from previous page)

```
userConfig->enableSensitivity          = true;
userConfig->enableShield               = false;
userConfig->xdn                        = kTSI_SensitivityXdnOption_1;
userConfig->ctrim                      = kTSI_SensitivityCtrimOption_7;
userConfig->inputCurrent               = kTSI_CurrentMultipleInputValue_0;
userConfig->chargeCurrent              = kTSI_CurrentMultipleChargeValue_1;
```

Parameters

- userConfig – Pointer to TSI user configure structure.

void TSI_GetMutualCapModeDefaultConfig(*tsi_mutualCap_config_t* *userConfig)

Get TSI mutual-cap mode default user configure structure. This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to a value:

```
userConfig->commonConfig.mainClock      = kTSI_MainClockSlection_1;
userConfig->commonConfig.mode           = kTSI_SensingModeSlection_Mutual;
userConfig->commonConfig.dvolt          = kTSI_DvoltOption_0;
userConfig->commonConfig.cutoff         = kTSI_SincCutoffDiv_1;
userConfig->commonConfig.order          = kTSI_SincFilterOrder_1;
userConfig->commonConfig.decimation     = kTSI_SincDecimationValue_8;
userConfig->commonConfig.chargeNum      = kTSI_SscChargeNumValue_4;
userConfig->commonConfig.prbsOutsel     = kTSI_SscPrbsOutsel_2;
userConfig->commonConfig.noChargeNum    = kTSI_SscNoChargeNumValue_5;
userConfig->commonConfig.ssc_mode       = kTSI_ssc_prbs_method;
userConfig->commonConfig.ssc_prescaler = kTSI_ssc_div_by_1;
userConfig->preCurrent                  = kTSI_MutualPreCurrent_4uA;
userConfig->preResistor                 = kTSI_MutualPreResistor_4k;
userConfig->senseResistor               = kTSI_MutualSenseResistor_10k;
userConfig->boostCurrent                = kTSI_MutualSenseBoostCurrent_0uA;
userConfig->txDriveMode                 = kTSI_MutualTxDriveModeOption_0;
userConfig->pmosLeftCurrent             = kTSI_MutualPmosCurrentMirrorLeft_32;
userConfig->pmosRightCurrent            = kTSI_MutualPmosCurrentMirrorRight_1;
userConfig->enableNmosMirror            = true;
userConfig->nmosCurrent                 = kTSI_MutualNmosCurrentMirror_1;
```

Parameters

- userConfig – Pointer to TSI user configure structure.

void TSI_SelfCapCalibrate(TSI_Type *base, *tsi_calibration_data_t* *calBuff)

Hardware base counter value for calibration.

Calibrate the peripheral to fetch the initial counter value of the enabled channels. This API is mostly used at initial application setup, it shall be called after the TSI_Init API, then user can use the calibrated counter values to setup applications(such as to determine

under which counter value we can confirm a touch event occurs).

Note: This API is mainly used for self-cap mode;

Note: The calibration work in mutual-cap mode shall be done in applications due to different board layout.

Parameters

- base – TSI peripheral base address.
- calBuff – Data buffer that store the calibrated counter value.

Returns
none

void TSI_EnableInterrupts(TSI_Type *base, uint32_t mask)
Enables TSI interrupt requests.

Parameters

- base – TSI peripheral base address.
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kTSI_GlobalInterruptEnable
  - kTSI_EndOfScanInterruptEnable
  - kTSI_OutOfRangeInterruptEnable

void TSI_DisableInterrupts(TSI_Type *base, uint32_t mask)
Disables TSI interrupt requests.

Parameters

- base – TSI peripheral base address.
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kTSI_GlobalInterruptEnable
  - kTSI_EndOfScanInterruptEnable
  - kTSI_OutOfRangeInterruptEnable

static inline uint32_t TSI_GetStatusFlags(TSI_Type *base)
Get interrupt flag. This function get tsi interrupt flags.

Parameters

- base – TSI peripheral base address.

Returns
The mask of these status flags combination.

void TSI_ClearStatusFlags(TSI_Type *base, uint32_t mask)
Clear interrupt flag.

This function clear tsi interrupt flag, automatically cleared flags can not be cleared by this function.

Parameters

- base – TSI peripheral base address.
- mask – The status flags to clear.

static inline uint32_t TSI_GetScanTriggerMode(TSI_Type *base)
Get TSI scan trigger mode.

Parameters

- base – TSI peripheral base address.

Returns
Scan trigger mode.

static inline bool TSI_IsScanInProgress(TSI_Type *base)

Get scan in progress flag.

Parameters

- base – TSI peripheral base address.

Returns

True - scan is in progress. False - scan is not in progress.

static inline void TSI_EnableModule(TSI_Type *base, bool enable)

Enables the TSI Module or not.

Parameters

- base – TSI peripheral base address.
- enable – Choose whether to enable or disable module;
  - true Enable TSI module;
  - false Disable TSI module;

Returns

none.

static inline void TSI_EnableLowPower(TSI_Type *base, bool enable)

Sets the TSI low power STOP mode enable or not. This enables TSI module function in low power modes.

Parameters

- base – TSI peripheral base address.
- enable – Choose to enable or disable STOP mode.
  - true Enable module in STOP mode;
  - false Disable module in STOP mode;

Returns

none.

static inline void TSI_EnableHardwareTriggerScan(TSI_Type *base, bool enable)

Enable the hardware trigger scan or not.

Parameters

- base – TSI peripheral base address.
- enable – Choose to enable hardware trigger or software trigger scan.
  - true Enable hardware trigger scan;
  - false Enable software trigger scan;

Returns

none.

static inline void TSI_StartSoftwareTrigger(TSI_Type *base)

Start one sotware trigger measurement (trigger a new measurement).

Parameters

- base – TSI peripheral base address.

Returns

none.

static inline void TSI_SetSelfCapMeasuredChannel(TSI_Type *base, uint8_t channel)

Set the measured channel number for self-cap mode.

Note: This API can only be used in self-cap mode!

Parameters

- base – TSI peripheral base address.
- channel – Channel number 0 … 24.

Returns

none.

static inline uint8_t TSI_GetSelfCapMeasuredChannel(TSI_Type *base)

Get the current measured channel number, in self-cap mode.

Note: This API can only be used in self-cap mode!

Parameters

- base – TSI peripheral base address.

Returns

uint8_t Channel number 0 … 24.

static inline void TSI_EnableDmaTransfer(TSI_Type *base, bool enable)

Enable DMA transfer or not.

Parameters

- base – TSI peripheral base address.
- enable – Choose to enable DMA transfer or not.
  - true Enable DMA transfer;
  - false Disable DMA transfer;

Returns

none.

static inline void TSI_EnableEndOfScanDmaTransferOnly(TSI_Type *base, bool enable)

Decide whether to enable End of Scan DMA transfer request only.

Parameters

- base – TSI peripheral base address.
- enable – Choose whether to enable End of Scan DMA transfer request only.
  - true Enable End of Scan DMA transfer request only;
  - false Both End-of-Scan and Out-of-Range can generate DMA transfer request.

Returns

none.

static inline uint16_t TSI_GetCounter(TSI_Type *base)

Gets the conversion counter value.

Parameters

- base – TSI peripheral base address.

Returns
Accumulated scan counter value ticked by the reference clock.

static inline void TSI_SetLowThreshold(TSI_Type *base, uint16_t low_threshold)
Set the TSI wake-up channel low threshold.

Parameters

- base – TSI peripheral base address.
- low_threshold – Low counter threshold.

Returns
none.

static inline void TSI_SetHighThreshold(TSI_Type *base, uint16_t high_threshold)
Set the TSI wake-up channel high threshold.

Parameters

- base – TSI peripheral base address.
- high_threshold – High counter threshold.

Returns
none.

static inline void TSI_SetMainClock(TSI_Type *base, *tsi_main_clock_selection_t* mainClock)
Set the main clock of the TSI module.

Parameters

- base – TSI peripheral base address.
- mainClock – clock option value.

Returns
none.

static inline void TSI_SetSensingMode(TSI_Type *base, *tsi_sensing_mode_selection_t* mode)
Set the sensing mode of the TSI module.

Parameters

- base – TSI peripheral base address.
- mode – Mode value.

Returns
none.

static inline *tsi_sensing_mode_selection_t* TSI_GetSensingMode(TSI_Type *base)
Get the sensing mode of the TSI module.

Parameters

- base – TSI peripheral base address.

Returns
Currently selected sensing mode.

static inline void TSI_SetDvolt(TSI_Type *base, *tsi_dvolt_option_t* dvolt)
Set the DVOLT settings.

Parameters

- base – TSI peripheral base address.
- dvolt – The voltage rails.

Returns
none.

static inline void TSI_EnableNoiseCancellation(TSI_Type *base, bool enableCancellation)

Enable self-cap mode noise cancellation function or not.

Parameters

- base – TSI peripheral base address.
- enableCancellation – Choose whether to enable noise cancellation in self-cap mode
  - true Enable noise cancellation;
  - false Disable noise cancellation;

Returns
none.

static inline void TSI_SetMutualCapTxChannel(TSI_Type *base, *tsi_mutual_tx_channel_t* txChannel)

Set the mutual-cap mode TX channel.

Parameters

- base – TSI peripheral base address.
- txChannel – Mutual-cap mode TX channel number

Returns
none.

static inline *tsi_mutual_tx_channel_t* TSI_GetTxMutualCapMeasuredChannel(TSI_Type *base)

Get the current measured TX channel number, in mutual-cap mode.

Note: This API can only be used in mutual-cap mode!

Parameters

- base – TSI peripheral base address;

Returns
Tx Channel number 0 … 5;

static inline void TSI_SetMutualCapRxChannel(TSI_Type *base, *tsi_mutual_rx_channel_t* rxChannel)

Set the mutual-cap mode RX channel.

Parameters

- base – TSI peripheral base address.
- rxChannel – Mutual-cap mode RX channel number

Returns
none.

static inline *tsi_mutual_rx_channel_t* TSI_GetRxMutualCapMeasuredChannel(TSI_Type *base)

Get the current measured RX channel number, in mutual-cap mode.

Note: This API can only be used in mutual-cap mode!

Parameters

- base – TSI peripheral base address;

Returns
Rx Channel number 6 … 11;

static inline void TSI_SetSscMode(TSI_Type *base, *tsi_ssc_mode_t* mode)
Set the SSC clock mode of the TSI module.

Parameters

- base – TSI peripheral base address.
- mode – SSC mode option value.

Returns
none.

static inline void TSI_SetSscPrescaler(TSI_Type *base, *tsi_ssc_prescaler_t* prescaler)
Set the SSC prescaler of the TSI module.

Parameters

- base – TSI peripheral base address.
- prescaler – SSC prescaler option value.

Returns
none.

static inline void TSI_SetUsedTxChannel(TSI_Type *base, *tsi_mutual_tx_channel_t* txChannel)
Set used mutual-cap TX channel.

Parameters

- base – TSI peripheral base address.
- txChannel – Mutual-cap mode TX channel number

Returns
none.

static inline void TSI_ClearUsedTxChannel(TSI_Type *base, *tsi_mutual_tx_channel_t* txChannel)
Clear used mutual-cap TX channel.

Parameters

- base – TSI peripheral base address.
- txChannel – Mutual-cap mode TX channel number

Returns
none.

FSL_TSI_DRIVER_VERSION
TSI driver version.

ALL_FLAGS_MASK
TSI status flags macro collection.

struct _tsi_calibration_data
#include <fsl_tsi_v5.h> TSI calibration data storage.

Public Members

uint16_t calibratedData[1]
TSI calibration data storage buffer

struct _tsi_common_config

#include <fsl_tsi_v5.h> TSI common configuration structure.

This structure contains the common settings for TSI self-cap or mutual-cap mode, configurations including the TSI module main clock, sensing mode, DVOLT options, SINC and SSC configurations.

Public Members

tsi_main_clock_selection_t mainClock

Set main clock.

tsi_sensing_mode_selection_t mode

Choose sensing mode.

tsi_dvolt_option_t dvolt

DVOLT option value.

tsi_sinc_cutoff_div_t cutoff

Cutoff divider.

tsi_sinc_filter_order_t order

SINC filter order.

tsi_sinc_decimation_value_t decimation

SINC decimation value.

tsi_ssc_charge_num_t chargeNum

SSC High Width (t1), SSC output bit0's period setting.

tsi_ssc_prbs_outsel_t prbsOutsel

SSC High Random Width (t2), length of PRBS(Pseudo-RandomBinarySequence),SSC output bit2's period setting.

tsi_ssc_nocharge_num_t noChargeNum

SSC Low Width (t3), SSC output bit1's period setting.

tsi_ssc_mode_t ssc_mode

Clock mode selection (basic - from main clock by divider,advanced - using SSC(Switching Speed Clock) by three configurable intervals.

tsi_ssc_prescaler_t ssc_prescaler

Set clock divider for basic mode.

struct _tsi_selfCap_config

#include <fsl_tsi_v5.h> TSI configuration structure for self-cap mode.

This structure contains the settings for the most common TSI self-cap configurations including the TSI module charge currents, sensitivity configuration and so on.

Public Members

tsi_common_config_t commonConfig

Common settings.

bool enableSensitivity

Enable sensitivity boost of self-cap or not.

tsi_shield_t enableShield

Enable shield of self-cap mode or not.

tsi_sensitivity_xdn_option_t xdn
    Sensitivity XDN option.

tsi_sensitivity_ctrim_option_t ctrim
    Sensitivity CTRIM option.

tsi_current_multiple_input_t inputCurrent
    Input current multiple.

tsi_current_multiple_charge_t chargeCurrent
    Charge/Discharge current multiple.

struct _tsi_mutualCap_config

#include <fsl_tsi_v5.h> TSI configuration structure for mutual-cap mode.

This structure contains the settings for the most common TSI mutual-cap configurations including the TSI module generator settings, sensitivity related current settings and so on.

Public Members

tsi_common_config_t commonConfig
    Common settings.

tsi_mutual_pre_current_t preCurrent
    Vref generator current.

tsi_mutual_pre_resistor_t preResistor
    Vref generator resistor.

tsi_mutual_sense_resistor_t senseResistor
    I-sense generator resistor.

tsi_mutual_sense_boost_current_t boostCurrent
    Sensitivity boost current setting.

tsi_mutual_tx_drive_mode_t txDriveMode
    TX drive mode control setting.

tsi_mutual_pmos_current_left_t pmosLeftCurrent
    Pmos current mirror on the left side.

tsi_mutual_pmos_current_right_t pmosRightCurrent
    Pmos current mirror on the right side.

bool enableNmosMirror
    Enable Nmos current mirror setting or not.

tsi_mutual_nmos_current_t nmosCurrent
    Nmos current mirror setting.

2.42 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

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)
FM-STR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")      /* entering a new virtual directory */

```

(continues on next page)

(continued from previous page)

```
/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR␣
↪tsaType,
        FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
        FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*

- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
        FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↪
        FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
        FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
        FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_xxx and FMSTR_PIPE_SIZE_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
        FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the nGranularity value equal to the nLength value, all data are considered as one chunk which is either written successfully as a whole or not at all. The nGranularity value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
        FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

FMSTR_TSA_TI By default, this is defined as FMSTR_SIZE.	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
FMSTR_TSA_TS By default, this is defined as FMSTR_SIZE.	Data type used to hold a memory block size, as used in the TSA descriptors.

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE Generally, this is a pointer to a void type.	Pipe handle that identifies the open-pipe object.
FMSTR_PIPE_P Generally, this is an unsigned 8-bit or 16-bit type.	Integer type required to hold at least 7 bits of data.
FMSTR_PIPE_SI This is used to store the data buffer sizes.	Integer type required to hold at least 16 bits of data.
FMSTR_PPIPEF See *FMSTR_PipeOpen* for more details.	Pointer to the pipe handler function.

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.

Chapter 4

RTOS

4.1 FreeRTOS

4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

FreeRTOS kernel for MCUXpresso SDK Readme

FreeRTOS kernel for MCUXpresso SDK ChangeLog

FreeRTOS kernel Readme

4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

4.1.5 corejson

JSON parser.

Readme

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

4.1.7 corepkcs11

PKCS #11 key management library.

Readme

4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme