# MCUXpresso SDK API Reference Manual

## NXP Semiconductors

Document Number: MCUXSDKAPIRM
Rev. 0
May 2019

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Chapter 1
# Introduction

The MCUXpresso Software Development Kit (MCUXpresso SDK) is a collection of software enablement for NXP Microcontrollers that includes peripheral drivers, multicore support and integrated RTOS support for FreeRTOS™ . In addition to the base enablement, the MCUXpresso SDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support provided by MCUXpresso SDK. The MCUXpresso SDK Web Builder is available to provide access to all MCUXpresso SDK packages. See the *MCUXpresso Software Development Kit (SDK) Release Notes* (document MCUXSDKRN) in the Supported Devices section at MCUXpresso-SDK: Software Development Kit for MCUXpresso for details.

The MCUXpresso SDK is built with the following runtime software components:

- Arm® and DSP standard libraries, and CMSIS-compliant device header files which provide direct access to the peripheral registers.
- Peripheral drivers that provide stateless, high-performance, ease-of-use APIs. Communication drivers provide higher-level transactional APIs for a higher-performance option.
- RTOS wrapper driver built on top of MCUXpresso SDK peripheral drivers and leverage native RTOS services to better comply to the RTOS cases.
- Real time operation systems (RTOS) for FreeRTOS OS.
- Stacks and middleware in source or object formats including:
  - CMSIS-DSP, a suite of common signal processing functions.
  - The MCUXpresso SDK comes complete with software examples demonstrating the usage of the peripheral drivers, RTOS wrapper drivers, middleware, and RTOSes.

  All demo applications and driver examples are provided with projects for the following toolchains:
  - IAR Embedded Workbench
  - GNU Arm Embedded Toolchain

The peripheral drivers and RTOS driver wrappers can be used across multiple devices within the product family without modification. The configuration items for each driver are encapsulated into C language data structures. Device-specific configuration information is provided as part of the MCUXpresso SDK and need not be modified by the user. If necessary, the user is able to modify the peripheral driver and RTOS wrapper driver configuration during runtime. The driver examples demonstrate how to configure the drivers by passing the proper configuration data to the APIs. The folder structure is organized to reduce the total number of includes required to compile a project.

The rest of this document describes the API references in detail for the peripheral drivers and RTOS wrapper drivers. For the latest version of this and other MCUXpresso SDK documents, see the mcuxpresso.nxp.com/apidoc/.

| Deliverable | Location |
|---|---|
| Demo Applications | <install_dir>/boards/<board_name>/demo_-apps |
| Driver Examples | <install_dir>/boards/<board_name>/driver_-examples |
| Documentation | <install_dir>/docs |
| Middleware | <install_dir>/middleware |
| Drivers | <install_dir>/<device_name>/drivers/ |
| CMSIS Standard Arm Cortex-M Headers, math and DSP Libraries | <install_dir>/CMSIS |
| Device Startup and Linker | <install_dir>/<device_name>/<toolchain>/ |
| MCUXpresso SDK Utilities | <install_dir>/devices/<device_name>/utilities |
| RTOS Kernel Code | <install_dir>/rtos |

Table 2: MCUXpresso SDK Folder Structure

# Chapter 2
# Driver errors status

- kStatus_CSI_NoEmptyBuffer = 2900
- kStatus_CSI_NoFullBuffer = 2901
- kStatus_CSI_QueueFull = 2902
- #kStatus_DCP_Again = 6700
- kStatus_EDMA_QueueFull = 5100
- kStatus_EDMA_Busy = 5101
- kStatus_ENET_RxFrameError = 4000
- kStatus_ENET_RxFrameFail = 4001
- kStatus_ENET_RxFrameEmpty = 4002
- kStatus_ENET_TxFrameOverLen = 4003
- kStatus_ENET_TxFrameBusy = 4004
- kStatus_ENET_TxFrameFail = 4005
- kStatus_ENET_PtpTsRingFull = 4006
- kStatus_ENET_PtpTsRingEmpty = 4007
- kStatus_FLEXCAN_TxBusy = 5300
- kStatus_FLEXCAN_TxIdle = 5301
- kStatus_FLEXCAN_TxSwitchToRx = 5302
- kStatus_FLEXCAN_RxBusy = 5303
- kStatus_FLEXCAN_RxIdle = 5304
- kStatus_FLEXCAN_RxOverflow = 5305
- kStatus_FLEXCAN_RxFifoBusy = 5306
- kStatus_FLEXCAN_RxFifoIdle = 5307
- kStatus_FLEXCAN_RxFifoOverflow = 5308
- kStatus_FLEXCAN_RxFifoWarning = 5309
- kStatus_FLEXCAN_ErrorStatus = 5310
- kStatus_FLEXCAN_UnHandled = 5311
- kStatus_FLEXIO_CAMERA_RxBusy = 5500
- kStatus_FLEXIO_CAMERA_RxIdle = 5501
- kStatus_FLEXIO_I2C_Busy = 800
- kStatus_FLEXIO_I2C_Idle = 801
- kStatus_FLEXIO_I2C_Nak = 802
- kStatus_FLEXIO_I2S_Idle = 2300
- kStatus_FLEXIO_I2S_TxBusy = 2301
- kStatus_FLEXIO_I2S_RxBusy = 2302
- kStatus_FLEXIO_I2S_Error = 2303
- kStatus_FLEXIO_I2S_QueueFull = 2304
- kStatus_FLEXIO_MCULCD_Idle = 2400
- kStatus_FLEXIO_MCULCD_Busy = 2401

- kStatus_FLEXIO_MCULCD_Error = 2302
- kStatus_FLEXIO_SPI_Busy = 501
- kStatus_FLEXIO_SPI_Idle = 502
- kStatus_FLEXIO_SPI_Error = 503
- kStatus_FLEXIO_UART_TxBusy = 700
- kStatus_FLEXIO_UART_RxBusy = 701
- kStatus_FLEXIO_UART_TxIdle = 702
- kStatus_FLEXIO_UART_RxIdle = 703
- kStatus_FLEXIO_UART_ERROR = 704
- kStatus_FLEXIO_UART_RxRingBufferOverrun = 705
- kStatus_FLEXIO_UART_RxHardwareOverrun = 706
- #kStatus_FLEXSPI_Idle = 7000
- kStatus_FLEXSPI_Busy = 7001
- kStatus_FLEXSPI_SequenceExecutionTimeout = 7002
- kStatus_FLEXSPI_IpCommandSequenceError = 7003
- kStatus_FLEXSPI_IpCommandGrantTimeout = 7004
- kStatus_LPI2C_Busy = 900
- kStatus_LPI2C_Idle = 901
- kStatus_LPI2C_Nak = 902
- kStatus_LPI2C_FifoError = 903
- kStatus_LPI2C_BitError = 904
- kStatus_LPI2C_ArbitrationLost = 905
- kStatus_LPI2C_PinLowTimeout = 906
- kStatus_LPI2C_NoTransferInProgress = 907
- kStatus_LPI2C_DmaRequestFail = 908
- kStatus_LPI2C_Timeout = 909
- kStatus_LPSPI_Busy = 400
- kStatus_LPSPI_Error = 401
- kStatus_LPSPI_Idle = 402
- kStatus_LPSPI_OutOfRange = 403
- kStatus_LPUART_TxBusy = 1300
- kStatus_LPUART_RxBusy = 1301
- kStatus_LPUART_TxIdle = 1302
- kStatus_LPUART_RxIdle = 1303
- kStatus_LPUART_TxWatermarkTooLarge = 1304
- kStatus_LPUART_RxWatermarkTooLarge = 1305
- kStatus_LPUART_FlagCannotClearManually = 1306
- kStatus_LPUART_Error = 1307
- kStatus_LPUART_RxRingBufferOverrun = 1308
- kStatus_LPUART_RxHardwareOverrun = 1309
- kStatus_LPUART_NoiseError = 1310
- kStatus_LPUART_FramingError = 1311
- kStatus_LPUART_ParityError = 1312
- kStatus_LPUART_BaudrateNotSupport = 1313
- kStatus_LPUART_IdleLineDetected = 1314

- kStatus_SAI_TxBusy = 1900
- kStatus_SAI_RxBusy = 1901
- kStatus_SAI_TxError = 1902
- kStatus_SAI_RxError = 1903
- kStatus_SAI_QueueFull = 1904
- kStatus_SAI_TxIdle = 1905
- kStatus_SAI_RxIdle = 1906
- #kStatus_SEMC_InvalidDeviceType = 10000
- #kStatus_SEMC_IpCommandExecutionError = 10001
- #kStatus_SEMC_AxiCommandExecutionError = 10002
- #kStatus_SEMC_InvalidMemorySize = 10003
- #kStatus_SEMC_InvalidIpcmdDataSize = 10004
- #kStatus_SEMC_InvalidAddressPortWidth = 10005
- #kStatus_SEMC_InvalidDataPortWidth = 10006
- #kStatus_SEMC_InvalidSwPinmuxSelection = 10007
- #kStatus_SEMC_InvalidBurstLength = 10008
- #kStatus_SEMC_InvalidColumnAddressBitWidth = 10009
- #kStatus_SEMC_InvalidBaseAddress = 10010
- #kStatus_SEMC_InvalidTimerSetting = 10011
- kStatus_SPDIF_RxDPLLLocked = 7500
- kStatus_SPDIF_TxFIFOError = 7501
- kStatus_SPDIF_TxFIFOResync = 7502
- kStatus_SPDIF_RxCnew = 7503
- kStatus_SPDIF_ValidatyNoGood = 7504
- kStatus_SPDIF_RxIllegalSymbol = 7505
- kStatus_SPDIF_RxParityBitError = 7506
- kStatus_SPDIF_UChannelOverrun = 7507
- kStatus_SPDIF_QChannelOverrun = 7508
- kStatus_SPDIF_UQChannelSync = 7509
- kStatus_SPDIF_UQChannelFrameError = 7510
- kStatus_SPDIF_RxFIFOError = 7511
- kStatus_SPDIF_RxFIFOResync = 7512
- kStatus_SPDIF_LockLoss = 7513
- kStatus_SPDIF_TxIdle = 7514
- kStatus_SPDIF_RxIdle = 7515
- kStatus_SPDIF_QueueFull = 7516
- kStatus_DMAMGR_ChannelOccupied = 5200
- kStatus_DMAMGR_ChannelNotUsed = 5201
- kStatus_DMAMGR_NoFreeChannel = 5202
- kStatus_NOTIFIER_ErrorNotificationBefore = 9800
- kStatus_NOTIFIER_ErrorNotificationAfter = 9801

# Chapter 3
# Architectural Overview

This chapter provides the architectural overview for the MCUXpresso Software Development Kit (MCUXpresso SDK). It describes each layer within the architecture and its associated components.

**Overview**

The MCUXpresso SDK architecture consists of five key components listed below.

1. The Arm Cortex Microcontroller Software Interface Standard (CMSIS) CORE compliance device-specific header files, SOC Header, and CMSIS math/DSP libraries.
2. Peripheral Drivers
3. Real-time Operating Systems (RTOS)
4. Stacks and Middleware that integrate with the MCUXpresso SDK
5. Demo Applications based on the MCUXpresso SDK

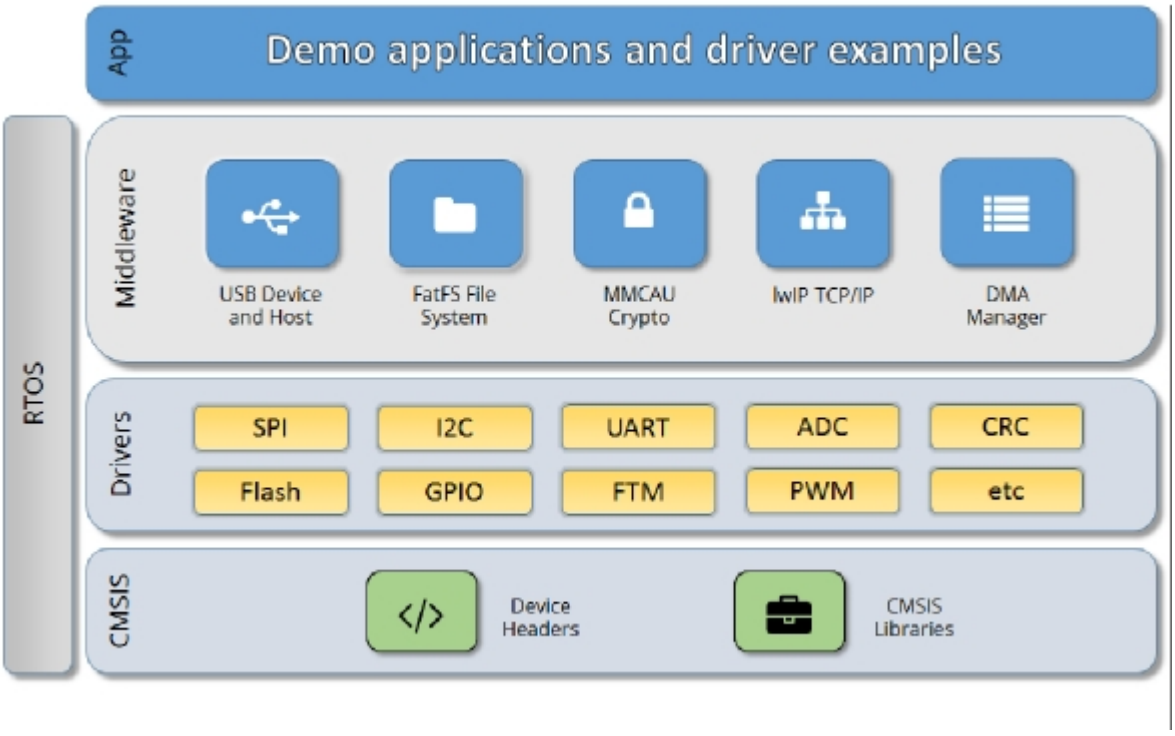


Figure 1: MCUXpresso SDK Block Diagram

**MCU header files**

Each supported MCU device in the MCUXpresso SDK has an overall System-on Chip (SoC) memory-

mapped header file. This header file contains the memory map and register base address for each peripheral and the IRQ vector table with associated vector numbers. The overall SoC header file provides access to the peripheral registers through pointers and predefined bit masks. In addition to the overall SoC memory-mapped header file, the MCUXpresso SDK includes a feature header file for each device. The feature header file allows NXP to deliver a single software driver for a given peripheral. The feature file ensures that the driver is properly compiled for the target SOC.

**CMSIS Support**

Along with the SoC header files and peripheral extension header files, the MCUXpresso SDK also includes common CMSIS header files for the Arm Cortex-M core and the math and DSP libraries from the latest CMSIS release. The CMSIS DSP library source code is also included for reference.

**MCUXpresso SDK Peripheral Drivers**

The MCUXpresso SDK peripheral drivers mainly consist of low-level functional APIs for the MCU product family on-chip peripherals and also of high-level transactional APIs for some bus drivers/DMA driver/eDMA driver to quickly enable the peripherals and perform transfers.

All MCUXpresso SDK peripheral drivers only depend on the CMSIS headers, device feature files, fsl_common.h, and fsl_clock.h files so that users can easily pull selected drivers and their dependencies into projects. With the exception of the clock/power-relevant peripherals, each peripheral has its own driver. Peripheral drivers handle the peripheral clock gating/ungating inside the drivers during initialization and deinitialization respectively.

Low-level functional APIs provide common peripheral functionality, abstracting the hardware peripheral register accesses into a set of stateless basic functional operations. These APIs primarily focus on the control, configuration, and function of basic peripheral operations. The APIs hide the register access details and various MCU peripheral instantiation differences so that the application can be abstracted from the low-level hardware details. The API prototypes are intentionally similar to help ensure easy portability across supported MCUXpresso SDK devices.

Transactional APIs provide a quick method for customers to utilize higher-level functionality of the peripherals. The transactional APIs utilize interrupts and perform asynchronous operations without user intervention. Transactional APIs operate on high-level logic that requires data storage for internal operation context handling. However, the Peripheral Drivers do not allocate this memory space. Rather, the user passes in the memory to the driver for internal driver operation. Transactional APIs ensure the NVIC is enabled properly inside the drivers. The transactional APIs do not meet all customer needs, but provide a baseline for development of custom user APIs.

Note that the transactional drivers never disable an NVIC after use. This is due to the shared nature of interrupt vectors on devices. It is up to the user to ensure that NVIC interrupts are properly disabled after usage is complete.

**Interrupt handling for transactional APIs**

A double weak mechanism is introduced for drivers with transactional API. The double weak indicates two levels of weak vector entries. See the examples below:

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

```
LDR     R0, =SPI0_DriverIRQHandler
BX      R0
```

The first level of the weak implementation are the functions defined in the vector table. In the devices/<-DEVICE_NAME>/<TOOLCHAIN>/startup_<DEVICE_NAME>.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0_DriverIRQHandler) jumps to itself (B). The MCUXpresso SDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0_-DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

The reason for implementing the double weak functions is to provide a better user experience when using the transactional APIs. For drivers with a transactional function, call the transactional APIs and the drivers complete the interrupt-driven flow. Users are not required to redefine the vector entries out of the box. At the same time, if users are not satisfied by the second layer weak function implemented in the MCU-Xpresso SDK drivers, users can redefine the first layer weak function and implement their own interrupt handler functions to suit their implementation.

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0_UART1_IRQHandler according to the use case requirements.

**Feature Header Files**

The peripheral drivers are designed to be reusable regardless of the peripheral functional differences from one MCU device to another. An overall Peripheral Feature Header File is provided for the MCUXpresso SDK-supported MCU device to define the features or configuration differences for each sub-family device.

**Application**

See the *Getting Started with MCUXpresso SDK* document (MCUXSDKGSUG).

# Chapter 4
# Trademarks

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: http://www.nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

# Chapter 5
# ADC: 12-bit Analog to Digital Converter Driver

## 5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit Analog to Digital Converter (ADC) module of MCUXpresso SDK devices.

## 5.2 Typical use case

### 5.2.1 Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/fsl_adc

### 5.2.2 Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/fsl_adc

## Data Structures

- struct adc_config_t
  *Converter configuration.* *More...*
- struct adc_offest_config_t
  *Converter Offset configuration.* *More...*
- struct adc_hardware_compare_config_t
  *ADC hardware compare configuration.* *More...*
- struct adc_channel_config_t
  *ADC channel conversion configuration.* *More...*

## Macros

- #define FSL_ADC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *ADC driver version.*

## Enumerations

- enum adc_status_flags_t {
  kADC_ConversionActiveFlag = ADC_GS_ADACT_MASK,
  kADC_CalibrationFailedFlag = ADC_GS_CALF_MASK,
  kADC_AsynchronousWakeupInterruptFlag }
  *Converter's status flags.*

**Typical use case**

- enum adc_reference_voltage_source_t { kADC_ReferenceVoltageSourceAlt0 = 0U }
  *Reference voltage source.*
- enum adc_sample_period_mode_t {
  kADC_SamplePeriod2or12Clocks = 0U,
  kADC_SamplePeriod4or16Clocks = 1U,
  kADC_SamplePeriod6or20Clocks = 2U,
  kADC_SamplePeriod8or24Clocks = 3U,
  kADC_SamplePeriodLong12Clcoks = kADC_SamplePeriod2or12Clocks,
  kADC_SamplePeriodLong16Clcoks = kADC_SamplePeriod4or16Clocks,
  kADC_SamplePeriodLong20Clcoks = kADC_SamplePeriod6or20Clocks,
  kADC_SamplePeriodLong24Clcoks = kADC_SamplePeriod8or24Clocks,
  kADC_SamplePeriodShort2Clocks = kADC_SamplePeriod2or12Clocks,
  kADC_SamplePeriodShort4Clocks = kADC_SamplePeriod4or16Clocks,
  kADC_SamplePeriodShort6Clocks = kADC_SamplePeriod6or20Clocks,
  kADC_SamplePeriodShort8Clocks = kADC_SamplePeriod8or24Clocks }
  *Sample time duration.*
- enum adc_clock_source_t {
  kADC_ClockSourceIPG = 0U,
  kADC_ClockSourceIPGDiv2 = 1U,
  kADC_ClockSourceAD = 3U }
  *Clock source.*
- enum adc_clock_driver_t {
  kADC_ClockDriver1 = 0U,
  kADC_ClockDriver2 = 1U,
  kADC_ClockDriver4 = 2U,
  kADC_ClockDriver8 = 3U }
  *Clock divider for the converter.*
- enum adc_resolution_t {
  kADC_Resolution8Bit = 0U,
  kADC_Resolution10Bit = 1U,
  kADC_Resolution12Bit = 2U }
  *Converter's resolution.*
- enum adc_hardware_compare_mode_t {
  kADC_HardwareCompareMode0 = 0U,
  kADC_HardwareCompareMode1 = 1U,
  kADC_HardwareCompareMode2 = 2U,
  kADC_HardwareCompareMode3 = 3U }
  *Converter hardware compare mode.*
- enum adc_hardware_average_mode_t {
  kADC_HardwareAverageCount4 = 0U,
  kADC_HardwareAverageCount8 = 1U,
  kADC_HardwareAverageCount16 = 2U,
  kADC_HardwareAverageCount32 = 3U,
  kADC_HardwareAverageDiasable = 4U }
  *Converter hardware average mode.*

## Variables

- bool adc_config_t::enableOverWrite
  *Enable the overwriting.*
- bool adc_config_t::enableContinuousConversion
  *Enable the continuous conversion mode.*
- bool adc_config_t::enableHighSpeed
  *Enable the high-speed mode.*
- bool adc_config_t::enableLowPower
  *Enable the low power mode.*
- bool adc_config_t::enableLongSample
  *Enable the long sample mode.*
- bool adc_config_t::enableAsynchronousClockOutput
  *Enable the asynchronous clock output.*
- adc_reference_voltage_source_t adc_config_t::referenceVoltageSource
  *Select the reference voltage source.*
- adc_sample_period_mode_t adc_config_t::samplePeriodMode
  *Select the sample period in long sample mode or short mode.*
- adc_clock_source_t adc_config_t::clockSource
  *Select the input clock source to generate the internal clock ADCK.*
- adc_clock_driver_t adc_config_t::clockDriver
  *Select the divide ratio used by the ADC to generate the internal clock ADCK.*
- adc_resolution_t adc_config_t::resolution
  *Select the ADC resolution mode.*
- bool adc_offest_config_t::enableSigned
  *if false,The offset value is added with the raw result.*
- uint32_t adc_offest_config_t::offsetValue
  *User configurable offset value(0-4095).*
- adc_hardware_compare_mode_t adc_hardware_compare_config_t::hardwareCompareMode
  *Select the hardware compare mode.*
- uint16_t adc_hardware_compare_config_t::value1
  *Setting value1(0-4095) for hardware compare mode.*
- uint16_t adc_hardware_compare_config_t::value2
  *Setting value2(0-4095) for hardware compare mode.*
- uint32_t adc_channel_config_t::channelNumber
  *Setting the conversion channel number.*
- bool adc_channel_config_t::enableInterruptOnConversionCompleted
  *Generate an interrupt request once the conversion is completed.*

## Initialization

- void ADC_Init (ADC_Type *base, const adc_config_t *config)
  *Initialize the ADC module.*
- void ADC_Deinit (ADC_Type *base)
  *De-initializes the ADC module.*
- void ADC_GetDefaultConfig (adc_config_t *config)
  *Gets an available pre-defined settings for the converter's configuration.*
- void ADC_SetChannelConfig (ADC_Type *base, uint32_t channelGroup, const adc_channel_config_t *config)
  *Configures the conversion channel.*
- static uint32_t ADC_GetChannelConversionValue (ADC_Type *base, uint32_t channelGroup)
  *Gets the conversion value.*

- static uint32_t ADC_GetChannelStatusFlags (ADC_Type *base, uint32_t channelGroup)
  *Gets the status flags of channel.*
- status_t ADC_DoAutoCalibration (ADC_Type *base)
  *Automates the hardware calibration.*
- void ADC_SetOffsetConfig (ADC_Type *base, const adc_offest_config_t *config)
  *Set user defined offset.*
- static void ADC_EnableDMA (ADC_Type *base, bool enable)
  *Enables generating the DMA trigger when the conversion is complete.*
- static void ADC_EnableHardwareTrigger (ADC_Type *base, bool enable)
  *Enables the hardware trigger mode.*
- void ADC_SetHardwareCompareConfig (ADC_Type *base, const adc_hardware_compare_config-_t *config)
  *Configures the hardware compare mode.*
- void ADC_SetHardwareAverageConfig (ADC_Type *base, adc_hardware_average_mode_t mode)
  *Configures the hardware average mode.*
- static uint32_t ADC_GetStatusFlags (ADC_Type *base)
  *Gets the converter's status flags.*
- void ADC_ClearStatusFlags (ADC_Type *base, uint32_t mask)
  *Clears the converter's status falgs.*

## 5.3 Data Structure Documentation

### 5.3.1 struct adc_config_t

### Data Fields

- bool enableOverWrite
  *Enable the overwriting.*
- bool enableContinuousConversion
  *Enable the continuous conversion mode.*
- bool enableHighSpeed
  *Enable the high-speed mode.*
- bool enableLowPower
  *Enable the low power mode.*
- bool enableLongSample
  *Enable the long sample mode.*
- bool enableAsynchronousClockOutput
  *Enable the asynchronous clock output.*
- adc_reference_voltage_source_t referenceVoltageSource
  *Select the reference voltage source.*
- adc_sample_period_mode_t samplePeriodMode
  *Select the sample period in long sample mode or short mode.*
- adc_clock_source_t clockSource
  *Select the input clock source to generate the internal clock ADCK.*
- adc_clock_driver_t clockDriver
  *Select the divide ratio used by the ADC to generate the internal clock ADCK.*
- adc_resolution_t resolution
  *Select the ADC resolution mode.*

### 5.3.2 struct adc_offest_config_t

#### Data Fields

- bool enableSigned
  *if false,The offset value is added with the raw result.*
- uint32_t offsetValue
  *User configurable offset value(0-4095).*

### 5.3.3 struct adc_hardware_compare_config_t

In kADC_HardwareCompareMode0, compare true if the result is less than the value1. In kADC_HardwareCompareMode1, compare true if the result is greater than or equal to value1. In kADC_HardwareCompareMode2, Value1 <= Value2, compare true if the result is less than value1 Or the result is Greater than value2. Value1 > Value2, compare true if the result is less than value1 And the result is Greater than value2. In kADC_HardwareCompareMode3, Value1 <= Value2, compare true if the result is greater than or equal to value1 And the result is less than or equal to value2. Value1 > Value2, compare true if the result is greater than or equal to value1 Or the result is less than or equal to value2.

#### Data Fields

- adc_hardware_compare_mode_t hardwareCompareMode
  *Select the hardware compare mode.*
- uint16_t value1
  *Setting value1(0-4095) for hardware compare mode.*
- uint16_t value2
  *Setting value2(0-4095) for hardware compare mode.*

### 5.3.4 struct adc_channel_config_t

#### Data Fields

- uint32_t channelNumber
  *Setting the conversion channel number.*
- bool enableInterruptOnConversionCompleted
  *Generate an interrupt request once the conversion is completed.*

## 5.4 Macro Definition Documentation

### 5.4.1 #define FSL_ADC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Version 2.0.2.

## 5.5 Enumeration Type Documentation

### 5.5.1 enum adc_status_flags_t

Enumerator

***kADC_ConversionActiveFlag*** Conversion is active,not support w1c.
***kADC_CalibrationFailedFlag*** Calibration is failed,support w1c.
***kADC_AsynchronousWakeupInterruptFlag*** Asynchronous wakeup interrupt occurred, support w1c.

### 5.5.2 enum adc_reference_voltage_source_t

Enumerator

***kADC_ReferenceVoltageSourceAlt0*** For external pins pair of VrefH and VrefL.

### 5.5.3 enum adc_sample_period_mode_t

Enumerator

***kADC_SamplePeriod2or12Clocks*** Long sample 12 clocks or short sample 2 clocks.
***kADC_SamplePeriod4or16Clocks*** Long sample 16 clocks or short sample 4 clocks.
***kADC_SamplePeriod6or20Clocks*** Long sample 20 clocks or short sample 6 clocks.
***kADC_SamplePeriod8or24Clocks*** Long sample 24 clocks or short sample 8 clocks.
***kADC_SamplePeriodLong12Clcoks*** Long sample 12 clocks.
***kADC_SamplePeriodLong16Clcoks*** Long sample 16 clocks.
***kADC_SamplePeriodLong20Clcoks*** Long sample 20 clocks.
***kADC_SamplePeriodLong24Clcoks*** Long sample 24 clocks.
***kADC_SamplePeriodShort2Clocks*** Short sample 2 clocks.
***kADC_SamplePeriodShort4Clocks*** Short sample 4 clocks.
***kADC_SamplePeriodShort6Clocks*** Short sample 6 clocks.
***kADC_SamplePeriodShort8Clocks*** Short sample 8 clocks.

### 5.5.4 enum adc_clock_source_t

Enumerator

***kADC_ClockSourceIPG*** Select IPG clock to generate ADCK.
***kADC_ClockSourceIPGDiv2*** Select IPG clock divided by 2 to generate ADCK.
***kADC_ClockSourceAD*** Select Asynchronous clock to generate ADCK.

### 5.5.5 enum adc_clock_driver_t

Enumerator

***kADC_ClockDriver1*** For divider 1 from the input clock to the module.
***kADC_ClockDriver2*** For divider 2 from the input clock to the module.
***kADC_ClockDriver4*** For divider 4 from the input clock to the module.
***kADC_ClockDriver8*** For divider 8 from the input clock to the module.

### 5.5.6 enum adc_resolution_t

Enumerator

***kADC_Resolution8Bit*** Single End 8-bit resolution.
***kADC_Resolution10Bit*** Single End 10-bit resolution.
***kADC_Resolution12Bit*** Single End 12-bit resolution.

### 5.5.7 enum adc_hardware_compare_mode_t

Enumerator

***kADC_HardwareCompareMode0*** Compare true if the result is less than the value1.
***kADC_HardwareCompareMode1*** Compare true if the result is greater than or equal to value1.
***kADC_HardwareCompareMode2*** Value1 <= Value2, compare true if the result is less than value1 Or the result is Greater than value2. Value1 > Value2, compare true if the result is less than value1 And the result is greater than value2
***kADC_HardwareCompareMode3*** Value1 <= Value2, compare true if the result is greater than or equal to value1 And the result is less than or equal to value2. Value1 > Value2, compare true if the result is greater than or equal to value1 Or the result is less than or equal to value2.

### 5.5.8 enum adc_hardware_average_mode_t

Enumerator

***kADC_HardwareAverageCount4*** For hardware average with 4 samples.
***kADC_HardwareAverageCount8*** For hardware average with 8 samples.
***kADC_HardwareAverageCount16*** For hardware average with 16 samples.
***kADC_HardwareAverageCount32*** For hardware average with 32 samples.
***kADC_HardwareAverageDiasable*** Disable the hardware average function.

## 5.6 Function Documentation

### 5.6.1 void ADC_Init ( ADC_Type * *base,* const adc_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *config* | Pointer to "adc_config_t" structure. |

### 5.6.2 void ADC_Deinit ( ADC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |

### 5.6.3 void ADC_GetDefaultConfig ( adc_config_t * *config* )

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

```
*  config->enableAsynchronousClockOutput = true;
*  config->enableOverWrite =               false;
*  config->enableContinuousConversion =    false;
*  config->enableHighSpeed =               false;
*  config->enableLowPower =                false;
*  config->enableLongSample =              false;
*  config->referenceVoltageSource =        kADC_ReferenceVoltageSourceAlt0;
*  config->samplePeriodMode =              kADC_SamplePeriod2or12Clocks;
*  config->clockSource =                   kADC_ClockSourceAD;
*  config->clockDriver =                   kADC_ClockDriver1;
*  config->resolution =                    kADC_Resolution12Bit;
*
```

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *config* | Pointer to the configuration structure. |

### 5.6.4 void ADC_SetChannelConfig ( ADC_Type * *base*, uint32_t *channelGroup*, const adc_channel_config_t * *config* )

This operation triggers the conversion when in software trigger mode. When 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 has more than one group of status and control registers, one for each conversion. The channel group parameter indicates which group of registers are used, for

example 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 point, only one of the channel groups is actively controlling ADC conversions. The channel group 0 is used for both software and hardware trigger modes. 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 appropriate 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. Therefore, writing to these channel groups does not initiate a new conversion. Updating the 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* | ADC peripheral base address. |
| *channelGroup* | Channel group index. |
| *config* | Pointer to the "adc_channel_config_t" structure for the conversion channel. |

### 5.6.5 static uint32_t ADC_GetChannelConversionValue ( ADC_Type * *base,* uint32_t *channelGroup* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *channelGroup* | Channel group index. |

Returns

Conversion value.

### 5.6.6 static uint32_t ADC_GetChannelStatusFlags ( ADC_Type * *base,* uint32_t *channelGroup* ) [inline], [static]

A conversion is completed when the result of the conversion is transferred into the data result registers. (provided the compare function & hardware averaging is disabled), this is indicated by the setting of COCOn. If hardware averaging is enabled, COCOn sets only, if the last of the selected number of conversions is complete. If the compare function is enabled, COCOn sets and conversion result data is transferred only if the compare condition is true. If both hardware averaging and compare functions are enabled, then COCOn sets only if the last of the selected number of conversions is complete and the compare condition is true.

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *channelGroup* | Channel group index. |

Returns

Status flags of channel.return 0 means COCO flag is 0,return 1 means COCOflag is 1.

### 5.6.7 status_t ADC_DoAutoCalibration ( ADC_Type * *base* )

This auto calibration helps to adjust the plus/minus side gain automatically. Execute the calibration before using the converter. Note that the software trigger should be used during calibration.

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |

Returns

Execution status.

Return values

| | |
|---|---|
| *kStatus_Success* | Calibration is done successfully. |
| *kStatus_Fail* | Calibration has failed. |

### 5.6.8 void ADC_SetOffsetConfig ( ADC_Type * *base,* const adc_offest_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *config* | Pointer to "adc_offest_config_t" structure. |

### 5.6.9 static void ADC_EnableDMA ( ADC_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *enable* | Switcher of the DMA feature. "true" means enabled, "false" means not enabled. |

### 5.6.10 static void ADC_EnableHardwareTrigger ( ADC_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *enable* | Switcher of the trigger mode. "true" means hardware tirgger mode,"false" means software mode. |

### 5.6.11 void ADC_SetHardwareCompareConfig ( ADC_Type * *base,* const adc_hardware_compare_config_t * *config* )

The hardware compare mode provides a way to process the conversion result automatically by using hardware. Only the result in the compare range is available. To compare the range, see "adc_hardware_-compare_mode_t" or the appopriate reference manual for more information.

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *Pointer* | to "adc_hardware_compare_config_t" structure. |

### 5.6.12 void ADC_SetHardwareAverageConfig ( ADC_Type * *base,* adc_hardware_average_mode_t *mode* )

The hardware average mode provides a way to process the conversion result automatically by using hardware. The multiple conversion results are accumulated and averaged internally making them easier to read.

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *mode* | Setting the hardware average mode. See "adc_hardware_average_mode_t". |

### 5.6.13 static uint32_t ADC_GetStatusFlags ( ADC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |

Returns

Flags' mask if indicated flags are asserted. See "adc_status_flags_t".

### 5.6.14 void ADC_ClearStatusFlags ( ADC_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | ADC peripheral base address. |
| *mask* | Mask value for the cleared flags. See "adc_status_flags_t". |

## 5.7 Variable Documentation

### 5.7.1 bool adc_config_t::enableOverWrite

### 5.7.2 bool adc_config_t::enableContinuousConversion

### 5.7.3 bool adc_config_t::enableHighSpeed

### 5.7.4 bool adc_config_t::enableLowPower

### 5.7.5 bool adc_config_t::enableLongSample

### 5.7.6 bool adc_config_t::enableAsynchronousClockOutput

### 5.7.7 adc_reference_voltage_source_t adc_config_t::referenceVoltageSource

### 5.7.8 adc_sample_period_mode_t adc_config_t::samplePeriodMode

### 5.7.9 adc_clock_source_t adc_config_t::clockSource

### 5.7.10 adc_clock_driver_t adc_config_t::clockDriver

### 5.7.11 adc_resolution_t adc_config_t::resolution

### 5.7.12 bool adc_offest_config_t::enableSigned

if true,The offset value is subtracted from the raw converted value.

### 5.7.13 uint32_t adc_offest_config_t::offsetValue

### 5.7.14 adc_hardware_compare_mode_t adc_hardware_compare_config_t::hardwareCompareMode

See "adc_hardware_compare_mode_t".

### 5.7.15 uint16_t adc_hardware_compare_config_t::value1

### 5.7.16 uint16_t adc_hardware_compare_config_t::value2

### 5.7.17 uint32_t adc_channel_config_t::channelNumber

The available range is 0-31. See channel connection information for each chip in Reference Manual document.

### 5.7.18 bool adc_channel_config_t::enableInterruptOnConversionCompleted

# Chapter 6
# ADC_ETC: ADC External Trigger Control

## 6.1 Overview

The MCUXpresso SDK provides a peripheral driver for the ADC_ETC module of MCUXpresso SDK devices.

## 6.2 Typical use case

### 6.2.1 Software trigger Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/adc_etc

### 6.2.2 Hardware trigger Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/adc_etc

## Data Structures

- struct adc_etc_config_t
  *ADC_ETC configuration. More...*
- struct adc_etc_trigger_chain_config_t
  *ADC_ETC trigger chain configuration. More...*
- struct adc_etc_trigger_config_t
  *ADC_ETC trigger configuration. More...*

## Macros

- #define FSL_ADC_ETC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *ADC_ETC driver version.*
- #define ADC_ETC_DMA_CTRL_TRGn_REQ_MASK 0xFF0000U
  *The mask of status flags cleared by writing 1.*

## Enumerations

- enum _adc_etc_status_flag_mask
  *ADC_ETC customized status flags mask.*
- enum adc_etc_external_trigger_source_t
  *External triggers sources.*
- enum adc_etc_interrupt_enable_t
  *Interrupt enable/disable mask.*

### Initialization

- void ADC_ETC_Init (ADC_ETC_Type ∗base, const adc_etc_config_t ∗config)
  *Initialize the ADC_ETC module.*
- void ADC_ETC_Deinit (ADC_ETC_Type ∗base)
  *De-Initialize the ADC_ETC module.*

## 6.3 Data Structure Documentation

### 6.3.1 struct adc_etc_config_t

### 6.3.2 struct adc_etc_trigger_chain_config_t

### 6.3.3 struct adc_etc_trigger_config_t

## 6.4 Macro Definition Documentation

### 6.4.1 #define FSL_ADC_ETC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Version 2.0.1.

### 6.4.2 #define ADC_ETC_DMA_CTRL_TRGn_REQ_MASK 0xFF0000U

## 6.5 Function Documentation

### 6.5.1 void ADC_ETC_Init ( ADC_ETC_Type ∗ *base,* const adc_etc_config_t ∗ *config* )

Parameters

| | |
|---|---|
| *base* | ADC_ETC peripheral base address. |
| *config* | Pointer to "adc_etc_config_t" structure. |

### 6.5.2 void ADC_ETC_Deinit ( ADC_ETC_Type ∗ *base* )

Parameters

| | |
|---|---|
| *base* | ADC_ETC peripheral base address. |

# Chapter 7
# AIPSTZ: AHB to IP Bridge

## 7.1 Overview

The MCUXpresso SDK provides a driver for the AHB-to-IP Bridge (AIPSTZ) of MCUXpresso SDK devices.

## Enumerations

- enum aipstz_master_privilege_level_t {
  kAIPSTZ_MasterBufferedWriteEnable = (1U << 3),
  kAIPSTZ_MasterTrustedForReadEnable = (1U << 2),
  kAIPSTZ_MasterTrustedForWriteEnable = (1U << 1),
  kAIPSTZ_MasterForceUserModeEnable = 1U }
  *List of AIPSTZ privilege configuration.*
- enum aipstz_master_t
  *List of AIPSTZ masters.*
- enum aipstz_peripheral_access_control_t
  *List of AIPSTZ peripheral access control configuration.*
- enum aipstz_peripheral_t
  *List of AIPSTZ peripherals.*

## Driver version

- #define FSL_AIPSTZ_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *Version 2.0.0.*

## Initialization and deinitialization

- void AIPSTZ_SetMasterPriviledgeLevel (AIPSTZ_Type *base, aipstz_master_t master, uint32_t privilegeConfig)
  *Configure the privilege level for master.*
- void AIPSTZ_SetPeripheralAccessControl (AIPSTZ_Type *base, aipstz_peripheral_t peripheral, uint32_t accessControl)
  *Configure the access for peripheral.*

## 7.2 Enumeration Type Documentation

### 7.2.1 enum aipstz_master_privilege_level_t

Enumerator

***kAIPSTZ_MasterBufferedWriteEnable*** Write accesses from this master are allowed to be buffered.

***kAIPSTZ_MasterTrustedForReadEnable*** This master is trusted for read accesses.
***kAIPSTZ_MasterTrustedForWriteEnable*** This master is trusted for write accesses.
***kAIPSTZ_MasterForceUserModeEnable*** Accesses from this master are forced to user-mode.

### 7.2.2 enum aipstz_master_t

Organized by width for the 8-15 bits and shift for lower 8 bits.

### 7.2.3 enum aipstz_peripheral_access_control_t

### 7.2.4 enum aipstz_peripheral_t

Organized by register offset for higher 32 bits, width for the 8-15 bits and shift for lower 8 bits.

## 7.3 Function Documentation

### 7.3.1 void AIPSTZ_SetMasterPriviledgeLevel ( AIPSTZ_Type * *base,* aipstz_master_t *master,* uint32_t *privilegeConfig* )

Parameters

| | |
|---|---|
| *base* | AIPSTZ peripheral base pointer |
| *master* | Masters for AIPSTZ. |
| *privilegeConfig* | Configuration is ORed from . |

### 7.3.2 void AIPSTZ_SetPeripheralAccessControl ( AIPSTZ_Type * *base,* aipstz_peripheral_t *peripheral,* uint32_t *accessControl* )

Parameters

| | |
|---|---|
| *base* | AIPSTZ peripheral base pointer |
| *master* | Peripheral for AIPSTZ. |

| | |
|---|---|
| *accessControl* | Configuration is ORed from . |

# Chapter 8
# AOI: Crossbar AND/OR/INVERT Driver

## 8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar AND/OR/INVERT (AOI) block of MCUXpresso SDK devices.

The AOI module supports a configurable number of event outputs, where each event output represents a user-programmed combinational boolean function based on four event inputs. The key features of this module include:

- Four dedicated inputs for each event output
- User-programmable combinational boolean function evaluation for each event output
- Memory-mapped device connected to a slave peripheral (IPS) bus
- Configurable number of event outputs

## 8.2 Function groups

### 8.2.1 AOI Initialization

To initialize the AOI driver, call the AOI_Init() function and pass a baseaddr pointer.

See the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/aoi## AOI Get Set Operation {#AOIOperation} The AOI module provides a universal boolean function generator using a four-term sum of products expression with each product term containing true or complement values of the four selected event inputs (A, B, C, D). The AOI is a highly programmable module for creating combinational boolean outputs for use as hardware triggers. Each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. To configure the selected AOI module event, call the API of the AOI_SetEventLogicConfig() function. To get the current event state configure, call the API of AOI_GetEventLogicConfig() function. The AOI module does not support any special modes of operation. See the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/aoi# Typical use case {#AOIUsercase} The AOI module is designed to be integrated in conjuction with one or more inter-peripheral crossbar switch (XBAR) modules. A crossbar switch is typically used to select the 4∗n AOI inputs from among available peripheral outputs and GPIO signals. The n EVENTn outputs from the AOI module are typically used as additional inputs to a second crossbar switch, adding to it the ability to connect to its outputs an arbitrary 4-input boolean function of its other inputs.

This is an example to initialize and configure the AOI driver for a possible use case. Because the AOI module function is directly connected with an XBAR (Inter-peripheral crossbar) module, other peripheral drivers (PIT, CMP, and XBAR) are used to show full functionality of AOI module.

For example: Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/aoi

**Function groups**

## Data Structures

- struct aoi_event_config_t
  *AOI event configuration structure. More...*

## Macros

- #define AOI AOI0
  *AOI peripheral address.*

## Enumerations

- enum aoi_input_config_t {
  kAOI_LogicZero = 0x0U,
  kAOI_InputSignal = 0x1U,
  kAOI_InvInputSignal = 0x2U,
  kAOI_LogicOne = 0x3U }
  *AOI input configurations.*
- enum aoi_event_t {
  kAOI_Event0 = 0x0U,
  kAOI_Event1 = 0x1U,
  kAOI_Event2 = 0x2U,
  kAOI_Event3 = 0x3U }
  *AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).*

## Driver version

- #define FSL_AOI_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *Version 2.0.0.*

## AOI Initialization

- void AOI_Init (AOI_Type ∗base)
  *Initializes an AOI instance for operation.*
- void AOI_Deinit (AOI_Type ∗base)
  *Deinitializes an AOI instance for operation.*

## AOI Get Set Operation

- void AOI_GetEventLogicConfig (AOI_Type ∗base, aoi_event_t event, aoi_event_config_t ∗config)
  *Gets the Boolean evaluation associated.*
- void AOI_SetEventLogicConfig (AOI_Type ∗base, aoi_event_t event, const aoi_event_config_t ∗eventConfig)
  *Configures an AOI event.*

## 8.3 Data Structure Documentation

### 8.3.1 struct aoi_event_config_t

Defines structure _aoi_event_config and use the AOI_SetEventLogicConfig() function to make whole event configuration.

#### Data Fields

- aoi_input_config_t PT0AC
  *Product term 0 input A.*
- aoi_input_config_t PT0BC
  *Product term 0 input B.*
- aoi_input_config_t PT0CC
  *Product term 0 input C.*
- aoi_input_config_t PT0DC
  *Product term 0 input D.*
- aoi_input_config_t PT1AC
  *Product term 1 input A.*
- aoi_input_config_t PT1BC
  *Product term 1 input B.*
- aoi_input_config_t PT1CC
  *Product term 1 input C.*
- aoi_input_config_t PT1DC
  *Product term 1 input D.*
- aoi_input_config_t PT2AC
  *Product term 2 input A.*
- aoi_input_config_t PT2BC
  *Product term 2 input B.*
- aoi_input_config_t PT2CC
  *Product term 2 input C.*
- aoi_input_config_t PT2DC
  *Product term 2 input D.*
- aoi_input_config_t PT3AC
  *Product term 3 input A.*
- aoi_input_config_t PT3BC
  *Product term 3 input B.*
- aoi_input_config_t PT3CC
  *Product term 3 input C.*
- aoi_input_config_t PT3DC
  *Product term 3 input D.*

## 8.4 Macro Definition Documentation

### 8.4.1 #define FSL_AOI_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

## 8.5 Enumeration Type Documentation

### 8.5.1 enum aoi_input_config_t

The selection item represents the Boolean evaluations.

Enumerator

***kAOI_LogicZero*** Forces the input to logical zero.
***kAOI_InputSignal*** Passes the input signal.
***kAOI_InvInputSignal*** Inverts the input signal.
***kAOI_LogicOne*** Forces the input to logical one.

### 8.5.2 enum aoi_event_t

Enumerator

***kAOI_Event0*** Event 0 index.
***kAOI_Event1*** Event 1 index.
***kAOI_Event2*** Event 2 index.
***kAOI_Event3*** Event 3 index.

## 8.6 Function Documentation

### 8.6.1 void AOI_Init ( AOI_Type * *base* )

This function un-gates the AOI clock.

Parameters

| | |
|---|---|
| *base* | AOI peripheral address. |

### 8.6.2 void AOI_Deinit ( AOI_Type * *base* )

This function shutdowns AOI module.

Parameters

| | |
|---|---|
| *base* | AOI peripheral address. |

### 8.6.3 void AOI_GetEventLogicConfig ( AOI_Type * *base*, aoi_event_t *event*, aoi_event_config_t * *config* )

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;

AOI_GetEventLogicConfig(AOI, kAOI_Event0, &demoEventLogicStruct);
```

Parameters

| | |
|---|---|
| *base* | AOI peripheral address. |
| *event* | Index of the event which will be set of type aoi_event_t. |
| *config* | Selected input configuration . |

### 8.6.4 void AOI_SetEventLogicConfig ( AOI_Type * *base*, aoi_event_t *event*, const aoi_event_config_t * *eventConfig* )

This function configures an AOI event according to the aoiEventConfig structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event.

Example:

```
aoi_event_config_t demoEventLogicStruct;

demoEventLogicStruct.PT0AC = kAOI_InvInputSignal;
demoEventLogicStruct.PT0BC = kAOI_InputSignal;
demoEventLogicStruct.PT0CC = kAOI_LogicOne;
demoEventLogicStruct.PT0DC = kAOI_LogicOne;

demoEventLogicStruct.PT1AC = kAOI_LogicZero;
demoEventLogicStruct.PT1BC = kAOI_LogicOne;
demoEventLogicStruct.PT1CC = kAOI_LogicOne;
demoEventLogicStruct.PT1DC = kAOI_LogicOne;

demoEventLogicStruct.PT2AC = kAOI_LogicZero;
demoEventLogicStruct.PT2BC = kAOI_LogicOne;
demoEventLogicStruct.PT2CC = kAOI_LogicOne;
demoEventLogicStruct.PT2DC = kAOI_LogicOne;

demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
demoEventLogicStruct.PT3DC = kAOI_LogicOne;

AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

Parameters

| | |
|---|---|
| *base* | AOI peripheral address. |
| *event* | Event which will be configured of type aoi_event_t. |
| *eventConfig* | Pointer to type aoi_event_config_t structure. The user is responsible for filling out the members of this structure and passing the pointer to this function. |

# Chapter 9
# BEE: Bus Encryption Engine

## 9.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Bus Encryption Engine (BEE) module.

The BEE module is implemented as an on-the-fly decryption engine. The main features of the BEE module are:

- Standard AXI interconnection
- On-the-fly AES-128 decryption, supporting ECB and CTR mode
- Aliased memory space support. Address remapping for up to two individual regions
- Independent AES Key management for those two individual regions
- Bus access pattern optimization with the aid of local store and forward buffer
- Non-secured access filtering based on security label of the access
- Illegal access check and filtering.

The known hardware limitations of the BEE module are as follows:

- Only supports 128 bits data width AXI interconnection
- Only supports 16-byte burst access size. For a single transaction, the minimum supported access size is limited to 4 bytes.
- Granularity of the address bias is 128 KB per step

## 9.2 BEE Driver Initialization and Configuration

The function BEE_Init() initializes the BEE to default values. The function BEE_GetDefaultConfig() loads default values to the BEE configuration structure. The default values are described below.

See the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/bee. The function BEE_Deinit() performs a hardware reset of BEE module and disables clocks. Configuration and keys from software for both regions are cleared.

## 9.3 Enable & Disable BEE

The function BEE_Enable() enables decryption using BEE. The function BEE_Disable() disables decryption using BEE.

## 9.4 Set BEE region config and key

The function BEE_SetConfig() sets BEE settings according to given configuration structure. The structure is described below.

See the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/bee. The function BEE_SetRegionKey() loads given AES key to BEE register for the given region. The key must

be 32-bit aligned and stored in little-endian format. Note that eFuse BEE_KEYx_SEL must be set accordingly to be able to load and use the key loaded in BEE registers. Otherwise, the key cannot be loaded and BEE uses the key from OTPMK or SW_GP2.

The function BEE_SetRegionNonce() loads given AES nonce (used only for AES CTR mode) to BEE register for the given region. The nonce must be 32-bit aligned and stored in little-endian format.

## 9.5 Status

Provides functions to get and clear the BEE status.

The function BEE_GetStatusFlags() returns status of BEE peripheral. The function BEE_ClearStatusFlags() clears the BEE status flags.

### 9.5.1 BEE example

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/bee

## Data Structures

- struct bee_region_config_t
  *BEE region configuration structure. More...*

## Enumerations

- enum bee_aes_mode_t {
  kBEE_AesEcbMode = 0U,
  kBEE_AesCtrMode = 1U }
- enum bee_region_t {
  kBEE_Region0 = 0U,
  kBEE_Region1 = 1U }
- enum bee_ac_prot_enable {
  kBEE_AccessProtDisabled = 0U,
  kBEE_AccessProtEnabled = 1U }
- enum bee_endian_swap_enable {
  kBEE_EndianSwapDisabled = 1U,
  kBEE_EndianSwapEnabled = 0U }
- enum bee_security_level {
  kBEE_SecurityLevel0 = 0U,
  kBEE_SecurityLevel1 = 1U,
  kBEE_SecurityLevel2 = 2U,
  kBEE_SecurityLevel3 = 3U }
- enum bee_status_flags_t {

kBEE_DisableAbortFlag = 1U,
kBEE_Reg0ReadSecViolation = 2U,
kBEE_ReadIllegalAccess = 4U,
kBEE_Reg1ReadSecViolation = 8U,
kBEE_Reg0AccessViolation = 16U,
kBEE_Reg1AccessViolation = 32U,
kBEE_IdleFlag = BEE_STATUS_BEE_IDLE_MASK }

## Functions

- void BEE_Init (BEE_Type *base)
  *Resets BEE module to factory default values.*
- void BEE_Deinit (BEE_Type *base)
  *Resets BEE module, clears keys for both regions and disables clock to the BEE.*
- static void BEE_Enable (BEE_Type *base)
  *Enables BEE decryption.*
- static void BEE_Disable (BEE_Type *base)
  *Disables BEE decryption.*
- void BEE_GetDefaultConfig (bee_region_config_t *config)
  *Loads default values to the BEE region configuration structure.*
- void BEE_SetConfig (BEE_Type *base, const bee_region_config_t *config)
  *Sets BEE configuration.*
- status_t BEE_SetRegionKey (BEE_Type *base, bee_region_t region, const uint8_t *key, size_t keySize)
  *Loads the AES key for selected region into BEE key registers.*
- status_t BEE_SetRegionNonce (BEE_Type *base, bee_region_t region, const uint8_t *nonce, size_t nonceSize)
  *Loads the nonce for selected region into BEE nonce registers.*
- uint32_t BEE_GetStatusFlags (BEE_Type *base)
  *Gets the BEE status flags.*
- void BEE_ClearStatusFlags (BEE_Type *base, uint32_t mask)
  *Clears the BEE status flags.*

## Variables

- bee_aes_mode_t bee_region_config_t::region0Mode
  *AES mode used for encryption/decryption for region 0.*
- bee_aes_mode_t bee_region_config_t::region1Mode
  *AES mode used for encryption/decryption for region 1.*
- uint32_t bee_region_config_t::region0AddrOffset
  *Region 0 address offset.*
- uint32_t bee_region_config_t::region1AddrOffset
  *Region 1 address offset.*
- bee_security_level bee_region_config_t::region0SecLevel
  *Region 0 security level.*
- bee_security_level bee_region_config_t::region1SecLevel
  *Region 1 security level.*
- uint32_t bee_region_config_t::region1Bot
  *Region 1 bottom address.*
- uint32_t bee_region_config_t::region1Top

*Region 1 top address.*
- bee_ac_prot_enable bee_region_config_t::accessPermission
  *Access permission control enable/disable.*
- bee_endian_swap_enable bee_region_config_t::endianSwapEn
  *Endian swap enable/disable.*

### Driver version

- #define FSL_BEE_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *BEE driver version.*

## 9.6 Data Structure Documentation

### 9.6.1 struct bee_region_config_t

### Data Fields

- bee_aes_mode_t region0Mode
  *AES mode used for encryption/decryption for region 0.*
- bee_aes_mode_t region1Mode
  *AES mode used for encryption/decryption for region 1.*
- uint32_t region0AddrOffset
  *Region 0 address offset.*
- uint32_t region1AddrOffset
  *Region 1 address offset.*
- bee_security_level region0SecLevel
  *Region 0 security level.*
- bee_security_level region1SecLevel
  *Region 1 security level.*
- uint32_t region1Bot
  *Region 1 bottom address.*
- uint32_t region1Top
  *Region 1 top address.*
- bee_ac_prot_enable accessPermission
  *Access permission control enable/disable.*
- bee_endian_swap_enable endianSwapEn
  *Endian swap enable/disable.*

## 9.7 Macro Definition Documentation

### 9.7.1 #define FSL_BEE_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Version 2.0.1.

Current version: 2.0.1

Change log:

- Version 2.0.1
  - Initial version

## 9.8 Enumeration Type Documentation

### 9.8.1 enum bee_aes_mode_t

Enumerator

***kBEE_AesEcbMode*** AES ECB Mode.
***kBEE_AesCtrMode*** AES CTR Mode.

### 9.8.2 enum bee_region_t

Enumerator

***kBEE_Region0*** BEE region 0.
***kBEE_Region1*** BEE region 1.

### 9.8.3 enum bee_ac_prot_enable

Enumerator

***kBEE_AccessProtDisabled*** BEE access permission control disabled.
***kBEE_AccessProtEnabled*** BEE access permission control enabled.

### 9.8.4 enum bee_endian_swap_enable

Enumerator

***kBEE_EndianSwapDisabled*** BEE endian swap disabled.
***kBEE_EndianSwapEnabled*** BEE endian swap enabled.

### 9.8.5 enum bee_security_level

Enumerator

***kBEE_SecurityLevel0*** BEE security level 0.
***kBEE_SecurityLevel1*** BEE security level 1.
***kBEE_SecurityLevel2*** BEE security level 2.
***kBEE_SecurityLevel3*** BEE security level 3.

### 9.8.6 enum bee_status_flags_t

Enumerator

***kBEE_DisableAbortFlag*** Disable abort flag.
***kBEE_Reg0ReadSecViolation*** Region-0 read channel security violation.
***kBEE_ReadIllegalAccess*** Read channel illegal access detected.
***kBEE_Reg1ReadSecViolation*** Region-1 read channel security violation.
***kBEE_Reg0AccessViolation*** Protected region-0 access violation.
***kBEE_Reg1AccessViolation*** Protected region-1 access violation.
***kBEE_IdleFlag*** Idle flag.

## 9.9 Function Documentation

### 9.9.1 void BEE_Init ( BEE_Type * *base* )

This function performs hardware reset of BEE module. Attributes and keys from software for both regions are cleared.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |

### 9.9.2 void BEE_Deinit ( BEE_Type * *base* )

This function performs hardware reset of BEE module and disables clocks. Attributes and keys from software for both regions are cleared.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |

### 9.9.3 static void BEE_Enable ( BEE_Type * *base* ) [inline], [static]

This function enables decryption using BEE.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |

### 9.9.4 static void BEE_Disable ( BEE_Type * *base* ) [inline], [static]

This function disables decryption using BEE.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |

### 9.9.5 void BEE_GetDefaultConfig ( bee_region_config_t * *config* )

Loads default values to the BEE region configuration structure. The default values are as follows:

```
*   config->region0Mode = kBEE_AesCtrMode;
*   config->region1Mode = kBEE_AesCtrMode;
*   config->region0AddrOffset = 0U;
*   config->region1AddrOffset = 0U;
*   config->region0SecLevel = kBEE_SecurityLevel3;
*   config->region1SecLevel = kBEE_SecurityLevel3;
*   config->region1Bot = 0U;
*   config->region1Top = 0U;
*   config->accessPermission = kBEE_AccessProtDisabled;
*   config->endianSwapEn = kBEE_EndianSwapEnabled;
*
```

Parameters

| | |
|---|---|
| *config* | Configuration structure for BEE peripheral. |

### 9.9.6 void BEE_SetConfig ( BEE_Type * *base*, const bee_region_config_t * *config* )

This function sets BEE peripheral and BEE region settings accorging to given configuration structure.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |

| | |
|---|---|
| *config* | Configuration structure for BEE. |

### 9.9.7 status_t BEE_SetRegionKey ( BEE_Type * *base*, bee_region_t *region*, const uint8_t * *key*, size_t *keySize* )

This function loads given AES key to BEE register for the given region. The key must be 32-bit aligned and stored in little-endian format.

Please note, that eFuse BEE_KEYx_SEL must be set accordingly to be able to load and use key loaded in BEE registers. Otherwise, key cannot loaded and BEE will use key from OTPMK or SW_GP2.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |
| *region* | Selection of the BEE region to be configured. |
| *key* | AES key (in little-endian format). |
| *keySize* | Size of AES key. |

### 9.9.8 status_t BEE_SetRegionNonce ( BEE_Type * *base*, bee_region_t *region*, const uint8_t * *nonce*, size_t *nonceSize* )

This function loads given nonce(only AES CTR mode) to BEE register for the given region. The nonce must be 32-bit aligned and stored in little-endian format.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |
| *region* | Selection of the BEE region to be configured. |
| *nonce* | AES nonce (in little-endian format). |
| *nonceSize* | Size of AES nonce. |

### 9.9.9 uint32_t BEE_GetStatusFlags ( BEE_Type * *base* )

This function returns status of BEE peripheral.

Parameters

| | |
|---|---|
| *base* | BEE peripheral address. |

Returns

The status flags. This is the logical OR of members of the enumeration bee_status_flags_t

### 9.9.10 void BEE_ClearStatusFlags ( BEE_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | BEE peripheral base address. |
| *mask* | The status flags to clear. This is a logical OR of members of the enumeration bee_status_flags_t |

# Chapter 10
# CACHE: CACHE Memory Controller

## 10.1 Overview

The MCUXpresso SDK provides a peripheral driver for the CACHE Controller of MCUXpresso SDK devices.

The CACHE driver is created to help the user more easily operate the cache memory. The APIs for basic operations are including the following three levels:

1L. The L1 cache driver API. This level provides the level 1 caches controller drivers. The L1 caches are mainly integrated in the Core memory system, Cortex-M7 L1 caches, etc. For our Cortex-M4 series platforms, the L1 cache is the local memory controller (LMEM) which is not integrated in the Cortex-M4 processer memory system.

2L. The L2 cache driver API. This level provides the level 2 cache controller drivers. The L2 cache could be integrated in the CORE memory system or an external L2 cache memory, PL310, etc.

3L. The combined cache driver API. This level provides many APIs for combined L1 and L2 cache maintain operations. This is provided for MCUXpresso SDK drivers (DMA, ENET, USDHC, etc) which should do the cache maintenance in their transactional APIs.

## 10.2 Function groups

### L1 CACHE Operation {#L1CACHE MaintainOperation}

The L1 CACHE has both code cache and data cache. This function group provides independent two groups API for both code cache and data cache. There are Enable/Disable APIs for code cache and data cache control and cache maintenance operations as Invalidate/Clean/CleanInvalidate by all and by address range.

### L2 CACHE Operation {#L2CACHE MaintainOperation}

The L2 CACHE does not divide the cache to data and code. Instead, this function group provides one group cache maintenance operations as Enable/Disable/Invalidate/Clean/CleanInvalidate by all and by address range. Except the maintenance operation APIs, the L2 CACHE has it's initialization/configure API. The user can use the default configure parameter by calling L2CACHE_GetDefaultConfig() or changing the parameters as they wish. Then, call L2CACHE_Init to do the L2 CACHE initialization. After initialization, the L2 cache can then be enabled.

Note: For the core external l2 Cache, the SoC usually has the control bit to select the SRAM to use as L2 Cache or normal SRAM. Make sure this selection is right when you use the L2 CACHE feature.

### Driver version

- #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *cache driver version 2.0.1.*

### Control for cortex-m7 L1 cache

- static void L1CACHE_EnableICache (void)
  *Enables cortex-m7 L1 instruction cache.*
- static void L1CACHE_DisableICache (void)
  *Disables cortex-m7 L1 instruction cache.*
- static void L1CACHE_InvalidateICache (void)
  *Invalidate cortex-m7 L1 instruction cache.*
- void L1CACHE_InvalidateICacheByRange (uint32_t address, uint32_t size_byte)
  *Invalidate cortex-m7 L1 instruction cache by range.*
- static void L1CACHE_EnableDCache (void)
  *Enables cortex-m7 L1 data cache.*
- static void L1CACHE_DisableDCache (void)
  *Disables cortex-m7 L1 data cache.*
- static void L1CACHE_InvalidateDCache (void)
  *Invalidates cortex-m7 L1 data cache.*
- static void L1CACHE_CleanDCache (void)
  *Cleans cortex-m7 L1 data cache.*
- static void L1CACHE_CleanInvalidateDCache (void)
  *Cleans and Invalidates cortex-m7 L1 data cache.*
- static void L1CACHE_InvalidateDCacheByRange (uint32_t address, uint32_t size_byte)
  *Invalidates cortex-m7 L1 data cache by range.*
- static void L1CACHE_CleanDCacheByRange (uint32_t address, uint32_t size_byte)
  *Cleans cortex-m7 L1 data cache by range.*
- static void L1CACHE_CleanInvalidateDCacheByRange (uint32_t address, uint32_t size_byte)
  *Cleans and Invalidates cortex-m7 L1 data cache by range.*

### Unified Cache Control for all caches (cortex-m7 L1 cache + l2 pl310)

Mainly used for many drivers for easy cache operation.

- void ICACHE_InvalidateByRange (uint32_t address, uint32_t size_byte)
  *Invalidates all instruction caches by range.*
- void DCACHE_InvalidateByRange (uint32_t address, uint32_t size_byte)
  *Invalidates all data caches by range.*
- void DCACHE_CleanByRange (uint32_t address, uint32_t size_byte)
  *Cleans all data caches by range.*
- void DCACHE_CleanInvalidateByRange (uint32_t address, uint32_t size_byte)
  *Cleans and Invalidates all data caches by range.*

## 10.3 Macro Definition Documentation

### 10.3.1 #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

## 10.4 Function Documentation

### 10.4.1 void L1CACHE_InvalidateICacheByRange ( uint32_t *address,* uint32_t *size_byte* )

Parameters

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

Note

The start address and size_byte should be 32-byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 I-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

### 10.4.2 static void L1CACHE_InvalidateDCacheByRange ( uint32_t *address,* uint32_t *size_byte* ) [inline], [static]

Parameters

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

Note

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

### 10.4.3 static void L1CACHE_CleanDCacheByRange ( uint32_t *address,* uint32_t *size_byte* ) [inline], [static]

Parameters

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

Note

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

### 10.4.4 static void L1CACHE_CleanInvalidateDCacheByRange ( uint32_t *address,* uint32_t *size_byte* ) [inline], [static]

Parameters

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

Note

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

### 10.4.5 void ICACHE_InvalidateByRange ( uint32_t *address,* uint32_t *size_byte* )

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

Parameters

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

Note

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

### 10.4.6 void DCACHE_InvalidateByRange ( uint32_t *address,* uint32_t *size_byte* )

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

Parameters

| | |
|---|---|
| *address* | The physical address. |

| | |
|---|---|
| *size_byte* | size of the memory to be invalidated. |

Note

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

### 10.4.7 void DCACHE_CleanByRange ( uint32_t *address,* uint32_t *size_byte* )

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

Parameters

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

Note

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

### 10.4.8 void DCACHE_CleanInvalidateByRange ( uint32_t *address,* uint32_t *size_byte* )

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

Parameters

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

Note

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

# Chapter 11
# CMP: Analog Comparator Driver

## 11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Analog Comparator (CMP) module of MCUXpresso SDK devices.

The CMP driver is a basic comparator with advanced features. The APIs for the basic comparator enable the CMP to compare the two voltages of the two input channels and create the output of the comparator result. The APIs for advanced features can be used as the plug-in functions based on the basic comparator. They can process the comparator's output with hardware support.

## 11.2 Typical use case

### 11.2.1 Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/cmp

### 11.2.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/cmp

## Data Structures

- struct cmp_config_t
  *Configures the comparator. More...*
- struct cmp_filter_config_t
  *Configures the filter. More...*
- struct cmp_dac_config_t
  *Configures the internal DAC. More...*

## Enumerations

- enum _cmp_interrupt_enable {
  kCMP_OutputRisingInterruptEnable = CMP_SCR_IER_MASK,
  kCMP_OutputFallingInterruptEnable = CMP_SCR_IEF_MASK }
  *Interrupt enable/disable mask.*
- enum _cmp_status_flags {
  kCMP_OutputRisingEventFlag = CMP_SCR_CFR_MASK,
  kCMP_OutputFallingEventFlag = CMP_SCR_CFF_MASK,
  kCMP_OutputAssertEventFlag = CMP_SCR_COUT_MASK }
  *Status flags' mask.*

**Typical use case**

- enum cmp_hysteresis_mode_t {
  kCMP_HysteresisLevel0 = 0U,
  kCMP_HysteresisLevel1 = 1U,
  kCMP_HysteresisLevel2 = 2U,
  kCMP_HysteresisLevel3 = 3U }
  *CMP Hysteresis mode.*
- enum cmp_reference_voltage_source_t {
  kCMP_VrefSourceVin1 = 0U,
  kCMP_VrefSourceVin2 = 1U }
  *CMP Voltage Reference source.*

## Driver version

- #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *CMP driver version 2.0.1.*

## Initialization

- void CMP_Init (CMP_Type *base, const cmp_config_t *config)
  *Initializes the CMP.*
- void CMP_Deinit (CMP_Type *base)
  *De-initializes the CMP module.*
- static void CMP_Enable (CMP_Type *base, bool enable)
  *Enables/disables the CMP module.*
- void CMP_GetDefaultConfig (cmp_config_t *config)
  *Initializes the CMP user configuration structure.*
- void CMP_SetInputChannels (CMP_Type *base, uint8_t positiveChannel, uint8_t negativeChannel)
  *Sets the input channels for the comparator.*

## Advanced Features

- void CMP_EnableDMA (CMP_Type *base, bool enable)
  *Enables/disables the DMA request for rising/falling events.*
- static void CMP_EnableWindowMode (CMP_Type *base, bool enable)
  *Enables/disables the window mode.*
- void CMP_SetFilterConfig (CMP_Type *base, const cmp_filter_config_t *config)
  *Configures the filter.*
- void CMP_SetDACConfig (CMP_Type *base, const cmp_dac_config_t *config)
  *Configures the internal DAC.*
- void CMP_EnableInterrupts (CMP_Type *base, uint32_t mask)
  *Enables the interrupts.*
- void CMP_DisableInterrupts (CMP_Type *base, uint32_t mask)
  *Disables the interrupts.*

## Results

- uint32_t CMP_GetStatusFlags (CMP_Type *base)
  *Gets the status flags.*
- void CMP_ClearStatusFlags (CMP_Type *base, uint32_t mask)
  *Clears the status flags.*

## 11.3 Data Structure Documentation

### 11.3.1 struct cmp_config_t

#### Data Fields

- bool enableCmp
  *Enable the CMP module.*
- cmp_hysteresis_mode_t hysteresisMode
  *CMP Hysteresis mode.*
- bool enableHighSpeed
  *Enable High-speed (HS) comparison mode.*
- bool enableInvertOutput
  *Enable the inverted comparator output.*
- bool useUnfilteredOutput
  *Set the compare output(COUT) to equal COUTA(true) or COUT(false).*
- bool enablePinOut
  *The comparator output is available on the associated pin.*

##### 11.3.1.0.0.1 Field Documentation

###### 11.3.1.0.0.1.1 bool cmp_config_t::enableCmp

###### 11.3.1.0.0.1.2 cmp_hysteresis_mode_t cmp_config_t::hysteresisMode

###### 11.3.1.0.0.1.3 bool cmp_config_t::enableHighSpeed

###### 11.3.1.0.0.1.4 bool cmp_config_t::enableInvertOutput

###### 11.3.1.0.0.1.5 bool cmp_config_t::useUnfilteredOutput

###### 11.3.1.0.0.1.6 bool cmp_config_t::enablePinOut

### 11.3.2 struct cmp_filter_config_t

#### Data Fields

- bool enableSample
  *Using the external SAMPLE as a sampling clock input or using a divided bus clock.*
- uint8_t filterCount
  *Filter Sample Count.*
- uint8_t filterPeriod
  *Filter Sample Period.*

#### 11.3.2.0.0.2 Field Documentation

##### 11.3.2.0.0.2.1 bool cmp_filter_config_t::enableSample

##### 11.3.2.0.0.2.2 uint8_t cmp_filter_config_t::filterCount

Available range is 1-7; 0 disables the filter.

##### 11.3.2.0.0.2.3 uint8_t cmp_filter_config_t::filterPeriod

The divider to the bus clock. Available range is 0-255.

### 11.3.3 struct cmp_dac_config_t

### Data Fields

- cmp_reference_voltage_source_t referenceVoltageSource
  *Supply voltage reference source.*
- uint8_t DACValue
  *Value for the DAC Output Voltage.*

#### 11.3.3.0.0.3 Field Documentation

##### 11.3.3.0.0.3.1 cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource

##### 11.3.3.0.0.3.2 uint8_t cmp_dac_config_t::DACValue

Available range is 0-63.

## 11.4 Macro Definition Documentation

### 11.4.1 #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

## 11.5 Enumeration Type Documentation

### 11.5.1 enum _cmp_interrupt_enable

Enumerator

***kCMP_OutputRisingInterruptEnable*** Comparator interrupt enable rising.
***kCMP_OutputFallingInterruptEnable*** Comparator interrupt enable falling.

### 11.5.2 enum _cmp_status_flags

Enumerator

***kCMP_OutputRisingEventFlag*** Rising-edge on the comparison output has occurred.

***kCMP_OutputFallingEventFlag*** Falling-edge on the comparison output has occurred.
***kCMP_OutputAssertEventFlag*** Return the current value of the analog comparator output.

### 11.5.3 enum cmp_hysteresis_mode_t

Enumerator

***kCMP_HysteresisLevel0*** Hysteresis level 0.
***kCMP_HysteresisLevel1*** Hysteresis level 1.
***kCMP_HysteresisLevel2*** Hysteresis level 2.
***kCMP_HysteresisLevel3*** Hysteresis level 3.

### 11.5.4 enum cmp_reference_voltage_source_t

Enumerator

***kCMP_VrefSourceVin1*** Vin1 is selected as a resistor ladder network supply reference Vin.
***kCMP_VrefSourceVin2*** Vin2 is selected as a resistor ladder network supply reference Vin.

## 11.6 Function Documentation

### 11.6.1 void CMP_Init ( CMP_Type * *base,* const cmp_config_t * *config* )

This function initializes the CMP module. The operations included are as follows.

- Enabling the clock for CMP module.
- Configuring the comparator.
- Enabling the CMP module. Note that for some devices, multiple CMP instances share the same clock gate. In this case, to enable the clock for any instance enables all CMPs. See the appropriate MCU reference manual for the clock assignment of the CMP.

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *config* | Pointer to the configuration structure. |

### 11.6.2 void CMP_Deinit ( CMP_Type * *base* )

This function de-initializes the CMP module. The operations included are as follows.

- Disabling the CMP module.
- Disabling the clock for CMP module.

**Function Documentation**

This function disables the clock for the CMP. Note that for some devices, multiple CMP instances share the same clock gate. In this case, before disabling the clock for the CMP, ensure that all the CMP instances are not used.

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |

### 11.6.3 static void CMP_Enable ( CMP_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *enable* | Enables or disables the module. |

### 11.6.4 void CMP_GetDefaultConfig ( cmp_config_t * *config* )

This function initializes the user configuration structure to these default values.

```
*   config->enableCmp           = true;
*   config->hysteresisMode      = kCMP_HysteresisLevel0;
*   config->enableHighSpeed     = false;
*   config->enableInvertOutput  = false;
*   config->useUnfilteredOutput = false;
*   config->enablePinOut        = false;
*   config->enableTriggerMode   = false;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to the configuration structure. |

### 11.6.5 void CMP_SetInputChannels ( CMP_Type * *base*, uint8_t *positiveChannel*, uint8_t *negativeChannel* )

This function sets the input channels for the comparator. Note that two input channels cannot be set the same way in the application. When the user selects the same input from the analog mux to the positive and negative port, the comparator is disabled automatically.

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *positive-Channel* | Positive side input channel number. Available range is 0-7. |
| *negative-Channel* | Negative side input channel number. Available range is 0-7. |

### 11.6.6 void CMP_EnableDMA ( CMP_Type * *base*, bool *enable* )

This function enables/disables the DMA request for rising/falling events. Either event triggers the generation of the DMA request from CMP if the DMA feature is enabled. Both events are ignored for generating the DMA request from the CMP if the DMA is disabled.

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *enable* | Enables or disables the feature. |

### 11.6.7 static void CMP_EnableWindowMode ( CMP_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *enable* | Enables or disables the feature. |

### 11.6.8 void CMP_SetFilterConfig ( CMP_Type * *base*, const cmp_filter_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |

| | |
|---|---|
| *config* | Pointer to the configuration structure. |

### 11.6.9 void CMP_SetDACConfig ( CMP_Type * *base,* const cmp_dac_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *config* | Pointer to the configuration structure. "NULL" disables the feature. |

### 11.6.10 void CMP_EnableInterrupts ( CMP_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *mask* | Mask value for interrupts. See "_cmp_interrupt_enable". |

### 11.6.11 void CMP_DisableInterrupts ( CMP_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *mask* | Mask value for interrupts. See "_cmp_interrupt_enable". |

### 11.6.12 uint32_t CMP_GetStatusFlags ( CMP_Type * *base* )

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |

Returns

Mask value for the asserted flags. See "_cmp_status_flags".

### 11.6.13 void CMP_ClearStatusFlags ( CMP_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | CMP peripheral base address. |
| *mask* | Mask value for the flags. See "_cmp_status_flags". |

# Chapter 12
# CSI: CMOS Sensor Interface

## 12.1 Overview

The MCUXpresso SDK provides a driver for the CMOS Sensor Interface (CSI)

The CSI enables the chip to connect directly to external CMOS image sensors. The CSI driver provides functional APIs and transactional APIs for the CSI module. The functional APIs implement the basic functions, so the user can construct them for a special use case. The transactional APIs provide a queue mechanism in order for the user to submit an empty frame buffer and get a fully-filled frame buffer easily.

## 12.2 Frame Buffer Queue

The CSI transactional functions maintain a frame buffer queue. The queue size is defined by the macro CSI_DRIVER_QUEUE_SIZE. The queue size is 4 by default, but the user can override it by redefining the macro value in the project setting.

To use transactional APIs, first call CSI_TransferCreateHandle to create a handle to save the CSI driver state. This function initializes the frame buffer queue to empty status.

After the handle is created, the function CSI_TransferSubmitEmptyBuffer can be used to submit the empty frame buffer to the queue. If the queue does not have room to save the new empty frame buffers, this function returns with an error. It is not necessary to check the queue rooms before submitting an empty frame buffer. After this step, the application can call CSI_TransferStart to start the transfer. There must be at least two empty buffers in the queue, otherwise this function returns an error. The incoming frames are saved to the empty buffers one by one, and a callback is provided when every frame completed. To get the fully-filled frame buffer, call the function CSI_TransferGetFullBuffer. This function returns an error if the frame buffer queue does not have full buffers. Therefore, it is not necessary to check the full buffer number in the queue before this function.

To stop the transfer, call the function CSI_TransferStop at anytime. If the queue has some full frame buffers, the application can still read them out after this stop function.

During the transfer, if all empty buffers are fully-filled, the CSI module will be stopped silently. Be aware that the stop here is different with the stop by CSI_TransferStop. When the application submits new buffers to the queue and the queue has more than two empty buffers, the CSI module starts automatically to same frame to the empty buffer.

## 12.3 Fragment Mode

The frame buffer queue mechanism needs large memory, it is not suitable for some special case, for example, no SDRAM used. Fragment mode is designed for this purpose, it needs two types of buffers:

1. DMA buffer. It could be as small as (camera frame width x 2 x 2) bytes, CSI DMA writes the input data to this buffer.

2. Frame buffer. The input data is copied to this buffer at last. What is more, user could define a window (in other words, region of interest), only image in this window will be copied to the frame buffer. If input data is YUV422 format, user can only save Y component optionally.

Limitations:

1. Fragment mode could not be used together with frame buffer queue mode.
2. In fragment mode, user should pay attention to the system payload. When the payload is high, the image capture might be broken.

## 12.4 Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/csi

## Data Structures

- struct csi_config_t
  *Configuration to initialize the CSI module. More...*
- struct csi_handle_t
  *CSI handle structure. More...*

## Macros

- #define CSI_DRIVER_QUEUE_SIZE 4U
  *Size of the frame buffer queue used in CSI transactional function.*
- #define CSI_DRIVER_FRAG_MODE 0U
  *Enable fragment capture function or not.*

## Typedefs

- typedef void(∗ csi_transfer_callback_t )(CSI_Type ∗base, csi_handle_t ∗handle, status_t status, void ∗userData)
  *CSI transfer callback function.*

## Enumerations

- enum _csi_status {
  kStatus_CSI_NoEmptyBuffer = MAKE_STATUS(kStatusGroup_CSI, 0),
  kStatus_CSI_NoFullBuffer = MAKE_STATUS(kStatusGroup_CSI, 1),
  kStatus_CSI_QueueFull = MAKE_STATUS(kStatusGroup_CSI, 2),
  kStatus_CSI_FrameDone = MAKE_STATUS(kStatusGroup_CSI, 3) }
  *Error codes for the CSI driver.*
- enum csi_work_mode_t {
  kCSI_GatedClockMode = CSI_CSICR1_GCLK_MODE(1U),
  kCSI_NonGatedClockMode = 0U,
  kCSI_CCIR656ProgressiveMode = CSI_CSICR1_CCIR_EN(1U) }
  *CSI work mode.*
- enum csi_data_bus_t { kCSI_DataBus8Bit }
  *CSI data bus witdh.*

- enum _csi_polarity_flags {
  kCSI_HsyncActiveLow = 0U,
  kCSI_HsyncActiveHigh = CSI_CSICR1_HSYNC_POL_MASK,
  kCSI_DataLatchOnRisingEdge = CSI_CSICR1_REDGE_MASK,
  kCSI_DataLatchOnFallingEdge = 0U,
  kCSI_VsyncActiveHigh = 0U,
  kCSI_VsyncActiveLow = CSI_CSICR1_SOF_POL_MASK }
  *CSI signal polarity.*
- enum csi_fifo_t {
  kCSI_RxFifo = (1U << 0U),
  kCSI_StatFifo = (1U << 1U),
  kCSI_AllFifo = 0x01 | 0x02 }
  *The CSI FIFO, used for FIFO operation.*
- enum _csi_interrupt_enable {
  kCSI_EndOfFrameInterruptEnable = CSI_CSICR1_EOF_INT_EN_MASK,
  kCSI_ChangeOfFieldInterruptEnable = CSI_CSICR1_COF_INT_EN_MASK,
  kCSI_StatFifoOverrunInterruptEnable = CSI_CSICR1_SF_OR_INTEN_MASK,
  kCSI_RxFifoOverrunInterruptEnable = CSI_CSICR1_RF_OR_INTEN_MASK,
  kCSI_StatFifoDmaDoneInterruptEnable,
  kCSI_StatFifoFullInterruptEnable = CSI_CSICR1_STATFF_INTEN_MASK,
  kCSI_RxBuffer1DmaDoneInterruptEnable,
  kCSI_RxBuffer0DmaDoneInterruptEnable,
  kCSI_RxFifoFullInterruptEnable = CSI_CSICR1_RXFF_INTEN_MASK,
  kCSI_StartOfFrameInterruptEnable = CSI_CSICR1_SOF_INTEN_MASK,
  kCSI_EccErrorInterruptEnable = CSI_CSICR3_ECC_INT_EN_MASK,
  kCSI_AhbResErrorInterruptEnable = CSI_CSICR3_HRESP_ERR_EN_MASK,
  kCSI_BaseAddrChangeErrorInterruptEnable,
  kCSI_Field0DoneInterruptEnable = CSI_CSICR18_FIELD0_DONE_IE_MASK << 6U,
  kCSI_Field1DoneInterruptEnable = CSI_CSICR18_DMA_FIELD1_DONE_IE_MASK << 6U }
  *CSI feature interrupt source.*
- enum _csi_flags {

kCSI_RxFifoDataReadyFlag = CSI_CSISR_DRDY_MASK,
kCSI_EccErrorFlag = CSI_CSISR_ECC_INT_MASK,
kCSI_AhbResErrorFlag = CSI_CSISR_HRESP_ERR_INT_MASK,
kCSI_ChangeOfFieldFlag = CSI_CSISR_COF_INT_MASK,
kCSI_Field0PresentFlag = CSI_CSISR_F1_INT_MASK,
kCSI_Field1PresentFlag = CSI_CSISR_F2_INT_MASK,
kCSI_StartOfFrameFlag = CSI_CSISR_SOF_INT_MASK,
kCSI_EndOfFrameFlag = CSI_CSISR_EOF_INT_MASK,
kCSI_RxFifoFullFlag = CSI_CSISR_RxFF_INT_MASK,
kCSI_RxBuffer1DmaDoneFlag = CSI_CSISR_DMA_TSF_DONE_FB2_MASK,
kCSI_RxBuffer0DmaDoneFlag = CSI_CSISR_DMA_TSF_DONE_FB1_MASK,
kCSI_StatFifoFullFlag = CSI_CSISR_STATFF_INT_MASK,
kCSI_StatFifoDmaDoneFlag = CSI_CSISR_DMA_TSF_DONE_SFF_MASK,
kCSI_StatFifoOverrunFlag = CSI_CSISR_SF_OR_INT_MASK,
kCSI_RxFifoOverrunFlag = CSI_CSISR_RF_OR_INT_MASK,
kCSI_Field0DoneFlag = CSI_CSISR_DMA_FIELD0_DONE_MASK,
kCSI_Field1DoneFlag = CSI_CSISR_DMA_FIELD1_DONE_MASK,
kCSI_BaseAddrChangeErrorFlag = CSI_CSISR_BASEADDR_CHHANGE_ERROR_MASK }
*CSI status flags.*

## Driver version

- #define **FSL_CSI_DRIVER_VERSION** (MAKE_VERSION(2, 0, 3))

## Initialization and deinitialization

- status_t CSI_Init (CSI_Type ∗base, const csi_config_t ∗config)
  *Initialize the CSI.*
- void CSI_Deinit (CSI_Type ∗base)
  *De-initialize the CSI.*
- void CSI_Reset (CSI_Type ∗base)
  *Reset the CSI.*
- void CSI_GetDefaultConfig (csi_config_t ∗config)
  *Get the default configuration for to initialize the CSI.*

## Module operation

- void CSI_ClearFifo (CSI_Type ∗base, csi_fifo_t fifo)
  *Clear the CSI FIFO.*
- void CSI_ReflashFifoDma (CSI_Type ∗base, csi_fifo_t fifo)
  *Reflash the CSI FIFO DMA.*
- void CSI_EnableFifoDmaRequest (CSI_Type ∗base, csi_fifo_t fifo, bool enable)
  *Enable or disable the CSI FIFO DMA request.*
- static void CSI_Start (CSI_Type ∗base)
  *Start to receive data.*
- static void CSI_Stop (CSI_Type ∗base)
  *Stop to receiving data.*
- void CSI_SetRxBufferAddr (CSI_Type ∗base, uint8_t index, uint32_t addr)
  *Set the RX frame buffer address.*

### Interrupts

- void CSI_EnableInterrupts (CSI_Type *base, uint32_t mask)
  *Enables CSI interrupt requests.*
- void CSI_DisableInterrupts (CSI_Type *base, uint32_t mask)
  *Disable CSI interrupt requests.*

### Status

- static uint32_t CSI_GetStatusFlags (CSI_Type *base)
  *Gets the CSI status flags.*
- static void CSI_ClearStatusFlags (CSI_Type *base, uint32_t statusMask)
  *Clears the CSI status flag.*

### Transactional

- status_t CSI_TransferCreateHandle (CSI_Type *base, csi_handle_t *handle, csi_transfer_callback_t callback, void *userData)
  *Initializes the CSI handle.*
- status_t CSI_TransferStart (CSI_Type *base, csi_handle_t *handle)
  *Start the transfer using transactional functions.*
- status_t CSI_TransferStop (CSI_Type *base, csi_handle_t *handle)
  *Stop the transfer using transactional functions.*
- status_t CSI_TransferSubmitEmptyBuffer (CSI_Type *base, csi_handle_t *handle, uint32_t frameBuffer)
  *Submit empty frame buffer to queue.*
- status_t CSI_TransferGetFullBuffer (CSI_Type *base, csi_handle_t *handle, uint32_t *frameBuffer)
  *Get one full frame buffer from queue.*
- void CSI_TransferHandleIRQ (CSI_Type *base, csi_handle_t *handle)
  *CSI IRQ handle function.*

## 12.5 Data Structure Documentation

### 12.5.1 struct csi_config_t

### Data Fields

- uint16_t width
  *Pixels of the input frame.*
- uint16_t height
  *Lines of the input frame.*
- uint32_t polarityFlags
  *Timing signal polarity flags, OR'ed value of _csi_polarity_flags.*
- uint8_t bytesPerPixel
  *Bytes per pixel, valid values are:*
- uint16_t linePitch_Bytes
  *Frame buffer line pitch, must be 8-byte aligned.*
- csi_work_mode_t workMode
  *CSI work mode.*

- csi_data_bus_t dataBus
  *Data bus width.*
- bool useExtVsync
  *In CCIR656 progressive mode, set true to use external VSYNC signal, set false to use internal VSYNC signal decoded from SOF.*

#### 12.5.1.0.0.4 Field Documentation

##### 12.5.1.0.0.4.1 uint16_t csi_config_t::width

##### 12.5.1.0.0.4.2 uint16_t csi_config_t::height

##### 12.5.1.0.0.4.3 uint32_t csi_config_t::polarityFlags

##### 12.5.1.0.0.4.4 uint8_t csi_config_t::bytesPerPixel

- 2: Used for RGB565, YUV422, and so on.
- 3: Used for packed RGB888, packed YUV444, and so on.
- 4: Used for XRGB8888, XYUV444, and so on.

##### 12.5.1.0.0.4.5 uint16_t csi_config_t::linePitch_Bytes

##### 12.5.1.0.0.4.6 csi_work_mode_t csi_config_t::workMode

##### 12.5.1.0.0.4.7 csi_data_bus_t csi_config_t::dataBus

##### 12.5.1.0.0.4.8 bool csi_config_t::useExtVsync

### 12.5.2 struct _csi_handle

Please see the user guide for the details of the CSI driver queue mechanism.

### Data Fields

- uint32_t frameBufferQueue [CSI_DRIVER_ACTUAL_QUEUE_SIZE]
  *Frame buffer queue.*
- volatile uint8_t queueUserReadIdx
  *Application gets full-filled frame buffer from this index.*
- volatile uint8_t queueUserWriteIdx
  *Application puts empty frame buffer to this index.*
- volatile uint8_t queueDrvReadIdx
  *Driver gets empty frame buffer from this index.*
- volatile uint8_t queueDrvWriteIdx
  *Driver puts the full-filled frame buffer to this index.*
- volatile uint8_t activeBufferNum
  *How many frame buffers are in progres currently.*
- volatile uint8_t nextBufferIdx
  *The CSI frame buffer index to use for next frame.*
- volatile bool transferStarted

*User has called CSI_TransferStart to start frame receiving.*

- volatile bool transferOnGoing
  *CSI is working and receiving incoming frames.*
- csi_transfer_callback_t callback
  *Callback function.*
- void * userData
  *CSI callback function parameter.*

#### 12.5.2.0.0.5 Field Documentation

#### 12.5.2.0.0.5.1 uint32_t csi_handle_t::frameBufferQueue[CSI_DRIVER_ACTUAL_QUEUE_SIZE]

#### 12.5.2.0.0.5.2 volatile uint8_t csi_handle_t::queueUserReadIdx

#### 12.5.2.0.0.5.3 volatile uint8_t csi_handle_t::queueUserWriteIdx

#### 12.5.2.0.0.5.4 volatile uint8_t csi_handle_t::queueDrvReadIdx

#### 12.5.2.0.0.5.5 volatile uint8_t csi_handle_t::queueDrvWriteIdx

#### 12.5.2.0.0.5.6 volatile uint8_t csi_handle_t::activeBufferNum

#### 12.5.2.0.0.5.7 volatile uint8_t csi_handle_t::nextBufferIdx

#### 12.5.2.0.0.5.8 volatile bool csi_handle_t::transferStarted

#### 12.5.2.0.0.5.9 volatile bool csi_handle_t::transferOnGoing

#### 12.5.2.0.0.5.10 csi_transfer_callback_t csi_handle_t::callback

#### 12.5.2.0.0.5.11 void* csi_handle_t::userData

## 12.6 Macro Definition Documentation

### 12.6.1 #define CSI_DRIVER_QUEUE_SIZE 4U

### 12.6.2 #define CSI_DRIVER_FRAG_MODE 0U

## 12.7 Typedef Documentation

### 12.7.1 typedef void(* csi_transfer_callback_t)(CSI_Type *base, csi_handle_t *handle, status_t status, void *userData)

When a new frame is received and saved to the frame buffer queue, the callback is called and the pass the status kStatus_CSI_FrameDone to upper layer.

## 12.8 Enumeration Type Documentation

### 12.8.1 enum _csi_status

Enumerator

***kStatus_CSI_NoEmptyBuffer*** No empty frame buffer in queue to load to CSI.
***kStatus_CSI_NoFullBuffer*** No full frame buffer in queue to read out.
***kStatus_CSI_QueueFull*** Queue is full, no room to save new empty buffer.
***kStatus_CSI_FrameDone*** New frame received and saved to queue.

### 12.8.2 enum csi_work_mode_t

The CCIR656 interlace mode is not supported currently.

Enumerator

***kCSI_GatedClockMode*** HSYNC, VSYNC, and PIXCLK signals are used.
***kCSI_NonGatedClockMode*** VSYNC, and PIXCLK signals are used.
***kCSI_CCIR656ProgressiveMode*** CCIR656 progressive mode.

### 12.8.3 enum csi_data_bus_t

Currently only support 8-bit width.

Enumerator

***kCSI_DataBus8Bit*** 8-bit data bus.

### 12.8.4 enum _csi_polarity_flags

Enumerator

***kCSI_HsyncActiveLow*** HSYNC is active low.
***kCSI_HsyncActiveHigh*** HSYNC is active high.
***kCSI_DataLatchOnRisingEdge*** Pixel data latched at rising edge of pixel clock.
***kCSI_DataLatchOnFallingEdge*** Pixel data latched at falling edge of pixel clock.
***kCSI_VsyncActiveHigh*** VSYNC is active high.
***kCSI_VsyncActiveLow*** VSYNC is active low.

### 12.8.5 enum csi_fifo_t

Enumerator

***kCSI_RxFifo*** RXFIFO.
***kCSI_StatFifo*** STAT FIFO.
***kCSI_AllFifo*** Both RXFIFO and STAT FIFO.

### 12.8.6 enum _csi_interrupt_enable

Enumerator

***kCSI_EndOfFrameInterruptEnable*** End of frame interrupt enable.
***kCSI_ChangeOfFieldInterruptEnable*** Change of field interrupt enable.
***kCSI_StatFifoOverrunInterruptEnable*** STAT FIFO overrun interrupt enable.
***kCSI_RxFifoOverrunInterruptEnable*** RXFIFO overrun interrupt enable.
***kCSI_StatFifoDmaDoneInterruptEnable*** STAT FIFO DMA done interrupt enable.
***kCSI_StatFifoFullInterruptEnable*** STAT FIFO full interrupt enable.
***kCSI_RxBuffer1DmaDoneInterruptEnable*** RX frame buffer 1 DMA transfer done.
***kCSI_RxBuffer0DmaDoneInterruptEnable*** RX frame buffer 0 DMA transfer done.
***kCSI_RxFifoFullInterruptEnable*** RXFIFO full interrupt enable.
***kCSI_StartOfFrameInterruptEnable*** Start of frame (SOF) interrupt enable.
***kCSI_EccErrorInterruptEnable*** ECC error detection interrupt enable.
***kCSI_AhbResErrorInterruptEnable*** AHB response Error interrupt enable.
***kCSI_BaseAddrChangeErrorInterruptEnable*** The DMA output buffer base address changes before DMA completed.
***kCSI_Field0DoneInterruptEnable*** Field 0 done interrupt enable.
***kCSI_Field1DoneInterruptEnable*** Field 1 done interrupt enable.

### 12.8.7 enum _csi_flags

The following status register flags can be cleared:

- kCSI_EccErrorFlag
- kCSI_AhbResErrorFlag
- kCSI_ChangeOfFieldFlag
- kCSI_StartOfFrameFlag
- kCSI_EndOfFrameFlag
- kCSI_RxBuffer1DmaDoneFlag
- kCSI_RxBuffer0DmaDoneFlag
- kCSI_StatFifoDmaDoneFlag
- kCSI_StatFifoOverrunFlag
- kCSI_RxFifoOverrunFlag

- kCSI_Field0DoneFlag
- kCSI_Field1DoneFlag
- kCSI_BaseAddrChangeErrorFlag

Enumerator

***kCSI_RxFifoDataReadyFlag*** RXFIFO data ready.
***kCSI_EccErrorFlag*** ECC error detected.
***kCSI_AhbResErrorFlag*** Hresponse (AHB bus response) Error.
***kCSI_ChangeOfFieldFlag*** Change of field.
***kCSI_Field0PresentFlag*** Field 0 present in CCIR mode.
***kCSI_Field1PresentFlag*** Field 1 present in CCIR mode.
***kCSI_StartOfFrameFlag*** Start of frame (SOF) detected.
***kCSI_EndOfFrameFlag*** End of frame (EOF) detected.
***kCSI_RxFifoFullFlag*** RXFIFO full (Number of data reaches trigger level).
***kCSI_RxBuffer1DmaDoneFlag*** RX frame buffer 1 DMA transfer done.
***kCSI_RxBuffer0DmaDoneFlag*** RX frame buffer 0 DMA transfer done.
***kCSI_StatFifoFullFlag*** STAT FIFO full (Reach trigger level).
***kCSI_StatFifoDmaDoneFlag*** STAT FIFO DMA transfer done.
***kCSI_StatFifoOverrunFlag*** STAT FIFO overrun.
***kCSI_RxFifoOverrunFlag*** RXFIFO overrun.
***kCSI_Field0DoneFlag*** Field 0 transfer done.
***kCSI_Field1DoneFlag*** Field 1 transfer done.
***kCSI_BaseAddrChangeErrorFlag*** The DMA output buffer base address changes before DMA completed.

## 12.9 Function Documentation

### 12.9.1 status_t CSI_Init ( CSI_Type * *base,* const csi_config_t * *config* )

This function enables the CSI peripheral clock, and resets the CSI registers.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *config* | Pointer to the configuration structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Initialize successfully. |

| | |
|---|---|
| *kStatus_InvalidArgument* | Initialize failed because of invalid argument. |

### 12.9.2 void CSI_Deinit ( CSI_Type * *base* )

This function disables the CSI peripheral clock.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |

### 12.9.3 void CSI_Reset ( CSI_Type * *base* )

This function resets the CSI peripheral registers to default status.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |

### 12.9.4 void CSI_GetDefaultConfig ( csi_config_t * *config* )

The default configuration value is:

```
config->width = 320U;
config->height = 240U;
config->polarityFlags = kCSI_HsyncActiveHigh |
      kCSI_DataLatchOnRisingEdge;
config->bytesPerPixel = 2U;
config->linePitch_Bytes = 320U * 2U;
config->workMode = kCSI_GatedClockMode;
config->dataBus = kCSI_DataBus8Bit;
config->useExtVsync = true;
```

Parameters

| | |
|---|---|
| *config* | Pointer to the CSI configuration. |

### 12.9.5 void CSI_ClearFifo ( CSI_Type * *base*, csi_fifo_t *fifo* )

This function clears the CSI FIFO.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *fifo* | The FIFO to clear. |

### 12.9.6 void CSI_ReflashFifoDma ( CSI_Type * *base*, csi_fifo_t *fifo* )

This function reflashes the CSI FIFO DMA.

For RXFIFO, there are two frame buffers. When the CSI module started, it saves the frames to frame buffer 0 then frame buffer 1, the two buffers will be written by turns. After reflash DMA using this function, the CSI is reset to save frame to buffer 0.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *fifo* | The FIFO DMA to reflash. |

### 12.9.7 void CSI_EnableFifoDmaRequest ( CSI_Type * *base*, csi_fifo_t *fifo*, bool *enable* )

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *fifo* | The FIFO DMA reques to enable or disable. |
| *enable* | True to enable, false to disable. |

### 12.9.8 static void CSI_Start ( CSI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |

### 12.9.9 static void CSI_Stop ( CSI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |

### 12.9.10 void CSI_SetRxBufferAddr ( CSI_Type * *base*, uint8_t *index*, uint32_t *addr* )

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *index* | Buffer index. |
| *addr* | Frame buffer address to set. |

### 12.9.11 void CSI_EnableInterrupts ( CSI_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *mask* | The interrupts to enable, pass in as OR'ed value of _csi_interrupt_enable. |

### 12.9.12 void CSI_DisableInterrupts ( CSI_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *mask* | The interrupts to disable, pass in as OR'ed value of _csi_interrupt_enable. |

### 12.9.13 static uint32_t CSI_GetStatusFlags ( CSI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |

Returns

status flag, it is OR'ed value of _csi_flags.

### 12.9.14 static void CSI_ClearStatusFlags ( CSI_Type * *base*, uint32_t *statusMask* ) [inline], [static]

The flags to clear are passed in as OR'ed value of _csi_flags. The following flags are cleared automatically by hardware:

- kCSI_RxFifoFullFlag,
- kCSI_StatFifoFullFlag,
- kCSI_Field0PresentFlag,
- kCSI_Field1PresentFlag,
- kCSI_RxFifoDataReadyFlag,

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *statusMask* | The status flags mask, OR'ed value of _csi_flags. |

### 12.9.15 status_t CSI_TransferCreateHandle ( CSI_Type * *base*, csi_handle_t * *handle*, csi_transfer_callback_t *callback*, void * *userData* )

This function initializes CSI handle, it should be called before any other CSI transactional functions.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *handle* | Pointer to the handle structure. |
| *callback* | Callback function for CSI transfer. |
| *userData* | Callback function parameter. |

Return values

| | |
|---|---|
| *kStatus_Success* | Handle created successfully. |

### 12.9.16 status_t CSI_TransferStart ( CSI_Type * *base*, csi_handle_t * *handle* )

When the empty frame buffers have been submit to CSI driver using function CSI_TransferSubmitEmpty-Buffer, user could call this function to start the transfer. The incoming frame will be saved to the empty frame buffer, and user could be optionally notified through callback function.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *handle* | Pointer to the handle structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Started successfully. |
| *kStatus_CSI_NoEmpty-Buffer* | Could not start because no empty frame buffer in queue. |

### 12.9.17 status_t CSI_TransferStop ( CSI_Type * *base*, csi_handle_t * *handle* )

The driver does not clean the full frame buffers in queue. In other words, after calling this function, user still could get the full frame buffers in queue using function CSI_TransferGetFullBuffer.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *handle* | Pointer to the handle structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Stoped successfully. |

### 12.9.18 status_t CSI_TransferSubmitEmptyBuffer ( CSI_Type * *base*, csi_handle_t * *handle*, uint32_t *frameBuffer* )

This function could be called before CSI_TransferStart or after CSI_TransferStart. If there is no room in queue to store the empty frame buffer, this function returns error.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *handle* | Pointer to the handle structure. |
| *frameBuffer* | Empty frame buffer to submit. |

Return values

| | |
|---|---|
| *kStatus_Success* | Started successfully. |
| *kStatus_CSI_QueueFull* | Could not submit because there is no room in queue. |

### 12.9.19 status_t CSI_TransferGetFullBuffer ( CSI_Type * *base*, csi_handle_t * *handle*, uint32_t * *frameBuffer* )

After the transfer started using function CSI_TransferStart, the incoming frames will be saved to the empty frame buffers in queue. This function gets the full-filled frame buffer from the queue. If there is no full frame buffer in queue, this function returns error.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *handle* | Pointer to the handle structure. |
| *frameBuffer* | Full frame buffer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Started successfully. |
| *kStatus_CSI_NoFull-Buffer* | There is no full frame buffer in queue. |

### 12.9.20 void CSI_TransferHandleIRQ ( CSI_Type * *base*, csi_handle_t * *handle* )

This function handles the CSI IRQ request to work with CSI driver transactional APIs.

Parameters

| | |
|---|---|
| *base* | CSI peripheral base address. |
| *handle* | CSI handle pointer. |

# Chapter 13
# DCDC: DCDC Converter

## 13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the DCDC Converter (DCDC) module of MCUXpresso SDK devices.

This part describes the programming interface of the DCDC Converter Peripheral driver. The DCDC converter module is a synchronous buck mode DCDC converter. It can produce single outputs for SoC peripherals and external devices with high conversion efficiency. The converter can be operated in continuous or pulsed mode.

As a module to provide the power for hardware system, the DCDC starts working when the system is powered up before the software takes over the SoC. Some important configuration is done by the board settings. Before the software can access the DCDC's registers, the DCDC is already working normally with the default settings.

However, if the application needs to improve the DCDC's performance or change the default settings, the DCDC driver would help. The DCDC's register cannot be accessed by software before its initialization (open the clock gate). The user can configure the hardware according to the application guide from reference manual.

## 13.2 Function groups

### 13.2.1 Initialization and deinitialization

This function group is to enable/disable the operations to DCDC module through the driver.

### 13.2.2 Status

Provides functions to get the DCDC status.

### 13.2.3 Misc control

Provides functions to set the DCDC's miscellaneous control.

## 13.3 Application guideline

### 13.3.1 Continous conduction mode

### 13.3.2 Discontinous conduction mode

## Data Structures

- struct dcdc_detection_config_t
  *Configuration for DCDC detection. More...*
- struct dcdc_loop_control_config_t
  *Configuration for the loop control. More...*
- struct dcdc_low_power_config_t
  *Configuration for DCDC low power. More...*
- struct dcdc_internal_regulator_config_t
  *Configuration for DCDC internal regulator. More...*
- struct dcdc_min_power_config_t
  *Configuration for min power setting. More...*

## Macros

- #define FSL_DCDC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *DCDC driver version.*

## Enumerations

- enum _dcdc_status_flags_t { kDCDC_LockedOKStatus = (1U << 0U) }
  *DCDC status flags.*
- enum dcdc_comparator_current_bias_t {
  kDCDC_ComparatorCurrentBias50nA = 0U,
  kDCDC_ComparatorCurrentBias100nA = 1U,
  kDCDC_ComparatorCurrentBias200nA = 2U,
  kDCDC_ComparatorCurrentBias400nA = 3U }
  *The current bias of low power comparator.*
- enum dcdc_over_current_threshold_t {
  kDCDC_OverCurrentThresholdAlt0 = 0U,
  kDCDC_OverCurrentThresholdAlt1 = 1U,
  kDCDC_OverCurrentThresholdAlt2 = 2U,
  kDCDC_OverCurrentThresholdAlt3 = 3U }
  *The threshold of over current detection.*
- enum dcdc_peak_current_threshold_t {
  kDCDC_PeakCurrentThresholdAlt0 = 0U,
  kDCDC_PeakCurrentThresholdAlt1 = 1U,
  kDCDC_PeakCurrentThresholdAlt2 = 2U,
  kDCDC_PeakCurrentThresholdAlt3 = 3U,
  kDCDC_PeakCurrentThresholdAlt4 = 4U,
  kDCDC_PeakCurrentThresholdAlt5 = 5U }
  *The threshold if peak current detection.*

- enum dcdc_count_charging_time_period_t {
  kDCDC_CountChargingTimePeriod8Cycle = 0U,
  kDCDC_CountChargingTimePeriod16Cycle = 1U }
  *The period of counting the charging times in power save mode.*
- enum dcdc_count_charging_time_threshold_t {
  kDCDC_CountChargingTimeThreshold32 = 0U,
  kDCDC_CountChargingTimeThreshold64 = 1U,
  kDCDC_CountChargingTimeThreshold16 = 2U,
  kDCDC_CountChargingTimeThreshold8 = 3U }
  *The threshold of the counting number of charging times.*
- enum dcdc_clock_source_t {
  kDCDC_ClockAutoSwitch = 0U,
  kDCDC_ClockInternalOsc = 1U,
  kDCDC_ClockExternalOsc = 2U }
  *Oscillator clock option.*

## Variables

- bool dcdc_detection_config_t::enableXtalokDetection
  *Enable xtalok detection circuit.*
- bool dcdc_detection_config_t::powerDownOverVoltageDetection
  *Power down over-voltage detection comparator.*
- bool dcdc_detection_config_t::powerDownLowVlotageDetection
  *Power down low-voltage detection comparator.*
- bool dcdc_detection_config_t::powerDownOverCurrentDetection
  *Power down over-current detection.*
- bool dcdc_detection_config_t::powerDownPeakCurrentDetection
  *Power down peak-current detection.*
- bool dcdc_detection_config_t::powerDownZeroCrossDetection
  *Power down the zero cross detection function for discontinuous conductor mode.*
- dcdc_over_current_threshold_t dcdc_detection_config_t::OverCurrentThreshold
  *The threshold of over current detection.*
- dcdc_peak_current_threshold_t dcdc_detection_config_t::PeakCurrentThreshold
  *The threshold of peak current detection.*
- bool dcdc_loop_control_config_t::enableCommonHysteresis
  *Enable hysteresis in switching converter common mode analog comparators.*
- bool dcdc_loop_control_config_t::enableCommonThresholdDetection
  *Increase the threshold detection for common mode analog comparator.*
- bool dcdc_loop_control_config_t::enableInvertHysteresisSign
  *Invert the sign of the hysteresis in DC-DC analog comparators.*
- bool dcdc_loop_control_config_t::enableRCThresholdDetection
  *Increase the threshold detection for RC scale circuit.*
- uint32_t dcdc_loop_control_config_t::enableRCScaleCircuit
  *Available range is 0~7.*
- uint32_t dcdc_loop_control_config_t::complementFeedForwardStep
  *Available range is 0~7.*
- uint32_t dcdc_loop_control_config_t::controlParameterMagnitude
  *Available range is 0~15.*
- uint32_t dcdc_loop_control_config_t::integralProportionalRatio
  *Available range is 0~3.Ratio of integral control parameter to proportional control parameter in the switch-*

*ing DC-DC converter, and can be used to optimize efficiency and loop response.*

- bool dcdc_low_power_config_t::enableOverloadDetection
  *Enable the overload detection in power save mode, if current is larger than the overloading threshold (typical value is 50 mA), DCDC will switch to the run mode automatically.*
- bool dcdc_low_power_config_t::enableAdjustHystereticValue
  *Adjust hysteretic value in low power from 12.5mV to 25mV.*
- dcdc_count_charging_time_period_t dcdc_low_power_config_t::countChargingTimePeriod
  *The period of counting the charging times in power save mode.*
- dcdc_count_charging_time_threshold_t dcdc_low_power_config_t::countChargingTimeThreshold
  *the threshold of the counting number of charging times during the period that lp_overload_freq_sel sets in power save mode.*
- bool dcdc_internal_regulator_config_t::enableLoadResistor
  *control the load resistor of the internal regulator of DCDC, the load resistor is connected as default "true", and need set to "false" to disconnect the load resistor.*
- uint32_t dcdc_internal_regulator_config_t::feedbackPoint
  *Available range is 0∼3.*
- bool dcdc_min_power_config_t::enableUseHalfFreqForContinuous
  *Set DCDC clock to half frequency for the continuous mode.*

## Initialization and deinitialization

- void DCDC_Init (DCDC_Type ∗base)
  *Enable the access to DCDC registers.*
- void DCDC_Deinit (DCDC_Type ∗base)
  *Disable the access to DCDC registers.*

## Status

- uint32_t DCDC_GetstatusFlags (DCDC_Type ∗base)
  *Get DCDC status flags.*

## Misc control.

- static void DCDC_EnableOutputRangeComparator (DCDC_Type ∗base, bool enable)
  *Enable the output range comparator.*
- void DCDC_SetClockSource (DCDC_Type ∗base, dcdc_clock_source_t clockSource)
  *Configure the DCDC clock source.*
- void DCDC_GetDefaultDetectionConfig (dcdc_detection_config_t ∗config)
  *Get the default setting for detection configuration.*
- void DCDC_SetDetectionConfig (DCDC_Type ∗base, const dcdc_detection_config_t ∗config)
  *Configure the DCDC detection.*
- void DCDC_GetDefaultLowPowerConfig (dcdc_low_power_config_t ∗config)
  *Get the default setting for low power configuration.*
- void DCDC_SetLowPowerConfig (DCDC_Type ∗base, const dcdc_low_power_config_t ∗config)
  *Configure the DCDC low power.*
- void DCDC_ResetCurrentAlertSignal (DCDC_Type ∗base, bool enable)
  *Reset current alert signal.*
- static void DCDC_SetBandgapVoltageTrimValue (DCDC_Type ∗base, uint32_t trimValue)
  *Set the bangap trim value to trim bandgap voltage.*
- void DCDC_GetDefaultLoopControlConfig (dcdc_loop_control_config_t ∗config)
  *Get the default setting for loop control configuration.*

- void DCDC_SetLoopControlConfig (DCDC_Type *base, const dcdc_loop_control_config_t *config)
  *Configure the DCDC loop control.*
- void DCDC_SetMinPowerConfig (DCDC_Type *base, const dcdc_min_power_config_t *config)
  *Configure for the min power.*
- static void DCDC_SetLPComparatorBiasValue (DCDC_Type *base, dcdc_comparator_current_bias_t biasVaule)
  *Set the current bias of low power comparator.*
- static void **DCDC_LockTargetVoltage** (DCDC_Type *base)
- void DCDC_AdjustTargetVoltage (DCDC_Type *base, uint32_t VDDRun, uint32_t VDDStandby)
  *Adjust the target voltage of VDD_SOC in run mode and low power mode.*
- void DCDC_SetInternalRegulatorConfig (DCDC_Type *base, const dcdc_internal_regulator_config_t *config)
  *Configure the DCDC internal regulator.*
- static void DCDC_EnableAdjustDelay (DCDC_Type *base, bool enable)
  *Ajust delay to reduce ground noise.*
- static void DCDC_EnableImproveTransition (DCDC_Type *base, bool enable)
  *Enable/Disable to improve the transition from heavy load to light load.*

### Application guideline.

- void DCDC_BootIntoDCM (DCDC_Type *base)
  *Boot DCDC into DCM(discontinous conduction mode).*
- void DCDC_BootIntoCCM (DCDC_Type *base)
  *Boot DCDC into CCM(continous conduction mode).*

## 13.4 Data Structure Documentation

### 13.4.1 struct dcdc_detection_config_t

### Data Fields

- bool enableXtalokDetection
  *Enable xtalok detection circuit.*
- bool powerDownOverVoltageDetection
  *Power down over-voltage detection comparator.*
- bool powerDownLowVlotageDetection
  *Power down low-voltage detection comparator.*
- bool powerDownOverCurrentDetection
  *Power down over-current detection.*
- bool powerDownPeakCurrentDetection
  *Power down peak-current detection.*
- bool powerDownZeroCrossDetection
  *Power down the zero cross detection function for discontinuous conductor mode.*
- dcdc_over_current_threshold_t OverCurrentThreshold
  *The threshold of over current detection.*
- dcdc_peak_current_threshold_t PeakCurrentThreshold
  *The threshold of peak current detection.*

### 13.4.2 struct dcdc_loop_control_config_t

#### Data Fields

- bool enableCommonHysteresis
  *Enable hysteresis in switching converter common mode analog comparators.*
- bool enableCommonThresholdDetection
  *Increase the threshold detection for common mode analog comparator.*
- bool enableInvertHysteresisSign
  *Invert the sign of the hysteresis in DC-DC analog comparators.*
- bool enableRCThresholdDetection
  *Increase the threshold detection for RC scale circuit.*
- uint32_t enableRCScaleCircuit
  *Available range is 0∼7.*
- uint32_t complementFeedForwardStep
  *Available range is 0∼7.*
- uint32_t controlParameterMagnitude
  *Available range is 0∼15.*
- uint32_t integralProportionalRatio
  *Available range is 0∼3.Ratio of integral control parameter to proportional control parameter in the switching DC-DC converter, and can be used to optimize efficiency and loop response.*

### 13.4.3 struct dcdc_low_power_config_t

#### Data Fields

- bool enableOverloadDetection
  *Enable the overload detection in power save mode, if current is larger than the overloading threshold (typical value is 50 mA), DCDC will switch to the run mode automatically.*
- bool enableAdjustHystereticValue
  *Adjust hysteretic value in low power from 12.5mV to 25mV.*
- dcdc_count_charging_time_period_t countChargingTimePeriod
  *The period of counting the charging times in power save mode.*
- dcdc_count_charging_time_threshold_t countChargingTimeThreshold
  *the threshold of the counting number of charging times during the period that lp_overload_freq_sel sets in power save mode.*

### 13.4.4 struct dcdc_internal_regulator_config_t

#### Data Fields

- bool enableLoadResistor
  *control the load resistor of the internal regulator of DCDC, the load resistor is connected as default "true", and need set to "false" to disconnect the load resistor.*
- uint32_t feedbackPoint
  *Available range is 0∼3.*

### 13.4.5 struct dcdc_min_power_config_t

#### Data Fields

- bool enableUseHalfFreqForContinuous
  *Set DCDC clock to half frequency for the continuous mode.*

## 13.5 Macro Definition Documentation

### 13.5.1 #define FSL_DCDC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

Version 2.0.0.

## 13.6 Enumeration Type Documentation

### 13.6.1 enum _dcdc_status_flags_t

Enumerator

***kDCDC_LockedOKStatus*** Indicate DCDC status. 1'b1: DCDC already settled 1'b0: DCDC is settling.

### 13.6.2 enum dcdc_comparator_current_bias_t

Enumerator

***kDCDC_ComparatorCurrentBias50nA*** The current bias of low power comparator is 50nA.
***kDCDC_ComparatorCurrentBias100nA*** The current bias of low power comparator is 100nA.
***kDCDC_ComparatorCurrentBias200nA*** The current bias of low power comparator is 200nA.
***kDCDC_ComparatorCurrentBias400nA*** The current bias of low power comparator is 400nA.

### 13.6.3 enum dcdc_over_current_threshold_t

Enumerator

***kDCDC_OverCurrentThresholdAlt0*** 1A in the run mode, 0.25A in the power save mode.
***kDCDC_OverCurrentThresholdAlt1*** 2A in the run mode, 0.25A in the power save mode.
***kDCDC_OverCurrentThresholdAlt2*** 1A in the run mode, 0.2A in the power save mode.
***kDCDC_OverCurrentThresholdAlt3*** 2A in the run mode, 0.2A in the power save mode.

### 13.6.4 enum dcdc_peak_current_threshold_t

Enumerator

***kDCDC_PeakCurrentThresholdAlt0*** 150mA peak current threshold.
***kDCDC_PeakCurrentThresholdAlt1*** 250mA peak current threshold.
***kDCDC_PeakCurrentThresholdAlt2*** 350mA peak current threshold.
***kDCDC_PeakCurrentThresholdAlt3*** 450mA peak current threshold.
***kDCDC_PeakCurrentThresholdAlt4*** 550mA peak current threshold.
***kDCDC_PeakCurrentThresholdAlt5*** 650mA peak current threshold.

### 13.6.5 enum dcdc_count_charging_time_period_t

Enumerator

***kDCDC_CountChargingTimePeriod8Cycle*** Eight 32k cycle.
***kDCDC_CountChargingTimePeriod16Cycle*** Sixteen 32k cycle.

### 13.6.6 enum dcdc_count_charging_time_threshold_t

Enumerator

***kDCDC_CountChargingTimeThreshold32*** 0x0: 32.
***kDCDC_CountChargingTimeThreshold64*** 0x1: 64.
***kDCDC_CountChargingTimeThreshold16*** 0x2: 16.
***kDCDC_CountChargingTimeThreshold8*** 0x3: 8.

### 13.6.7 enum dcdc_clock_source_t

Enumerator

***kDCDC_ClockAutoSwitch*** Automatic clock switch from internal oscillator to external clock.
***kDCDC_ClockInternalOsc*** Use internal oscillator.
***kDCDC_ClockExternalOsc*** Use external 24M crystal oscillator.

## 13.7 Function Documentation

### 13.7.1 void DCDC_Init ( DCDC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |

### 13.7.2 void DCDC_Deinit ( DCDC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |

### 13.7.3 uint32_t DCDC_GetstatusFlags ( DCDC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | peripheral base address. |

Returns

Mask of asserted status flags. See to "_dcdc_status_flags_t".

### 13.7.4 static void DCDC_EnableOutputRangeComparator ( DCDC_Type * *base,* bool *enable* ) [inline], [static]

The output range comparator is disabled by default.

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *enable* | Enable the feature or not. |

### 13.7.5 void DCDC_SetClockSource ( DCDC_Type * *base,* dcdc_clock_source_t *clockSource* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *clockSource* | Clock source for DCDC. See to "dcdc_clock_source_t". |

### 13.7.6 void DCDC_GetDefaultDetectionConfig ( dcdc_detection_config_t * *config* )

The default configuration are set according to responding registers' setting when powered on. They are:

```
*   config->enableXtalokDetection = false;
*   config->powerDownOverVoltageDetection = true;
*   config->powerDownLowVlotageDetection = false;
*   config->powerDownOverCurrentDetection = true;
*   config->powerDownPeakCurrentDetection = true;
*   config->powerDownZeroCrossDetection = true;
*   config->OverCurrentThreshold = kDCDC_OverCurrentThresholdAlt0;
*   config->PeakCurrentThreshold = kDCDC_PeakCurrentThresholdAlt0;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to configuration structure. See to "dcdc_detection_config_t" |

### 13.7.7 void DCDC_SetDetectionConfig ( DCDC_Type * *base,* const dcdc_detection_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *config* | Pointer to configuration structure. See to "dcdc_detection_config_t" |

### 13.7.8 void DCDC_GetDefaultLowPowerConfig ( dcdc_low_power_config_t * *config* )

The default configuration are set according to responding registers' setting when powered on. They are:

```
*   config->enableOverloadDetection = true;
*   config->enableAdjustHystereticValue = false;
*   config->countChargingTimePeriod = kDCDC_CountChargingTimePeriod8Cycle
      ;
*   config->countChargingTimeThreshold = kDCDC_CountChargingTimeThreshold32
      ;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to configuration structure. See to "dcdc_low_power_config_t" |

### 13.7.9 void DCDC_SetLowPowerConfig ( DCDC_Type * *base*, const dcdc_low_power_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *config* | Pointer to configuration structure. See to "dcdc_low_power_config_t". |

### 13.7.10 void DCDC_ResetCurrentAlertSignal ( DCDC_Type * *base*, bool *enable* )

Alert signal is generate by peak current detection.

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *enable* | Switcher to reset signal. True means reset signal. False means don't reset signal. |

### 13.7.11 static void DCDC_SetBandgapVoltageTrimValue ( DCDC_Type * *base*, uint32_t *trimValue* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *TrimValue* | The bangap trim value. Available range is 0U-31U. |

### 13.7.12 void DCDC_GetDefaultLoopControlConfig ( dcdc_loop_control_config_t * *config* )

The default configuration are set according to responding registers' setting when powered on. They are:

```
*   config->enableCommonHysteresis = false;
*   config->enableCommonThresholdDetection = false;
*   config->enableInvertHysteresisSign = false;
*   config->enableRCThresholdDetection = false;
```

```
*   config->enableRCScaleCircuit = 0U;
*   config->complementFeedForwardStep = 0U;
*   config->controlParameterMagnitude = 2U;
*   config->integralProportionalRatio = 2U;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to configuration structure. See to "dcdc_loop_control_config_t" |

### 13.7.13 void DCDC_SetLoopControlConfig ( DCDC_Type * *base,* const dcdc_loop_control_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *config* | Pointer to configuration structure. See to "dcdc_loop_control_config_t". |

### 13.7.14 void DCDC_SetMinPowerConfig ( DCDC_Type * *base,* const dcdc_min_power_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *config* | Pointer to configuration structure. See to "dcdc_min_power_config_t". |

### 13.7.15 static void DCDC_SetLPComparatorBiasValue ( DCDC_Type * *base,* dcdc_comparator_current_bias_t *biasVaule* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *biasVaule* | The current bias of low power comparator. Refer to "dcdc_comparator_current_bias_t". |

### 13.7.16 void DCDC_AdjustTargetVoltage ( DCDC_Type * *base*, uint32_t *VDDRun*, uint32_t *VDDStandby* )

This function is to adjust the target voltage of DCDC output. Change them and finally wait until the output is stabled. Set the target value of run mode the same as low power mode before entering power save mode, because DCDC will switch back to run mode if it detects the current loading is larger than about 50 mA(typical value).

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *VDDRun* | Target value in run mode. 25 mV each step from 0x00 to 0x1F. 00 is for 0.8V, 0x1F is for 1.575V. |
| *VDDStandby* | Target value in low power mode. 25 mV each step from 0x00 to 0x4. 00 is for 0.9V, 0x4 is for 1.0V. |

### 13.7.17 void DCDC_SetInternalRegulatorConfig ( DCDC_Type * *base*, const dcdc_internal_regulator_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *config* | Pointer to configuration structure. See to "dcdc_internal_regulator_config_t". |

### 13.7.18 static void DCDC_EnableAdjustDelay ( DCDC_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *enable* | Enable the feature or not. |

### 13.7.19 static void DCDC_EnableImproveTransition ( DCDC_Type * *base*, bool *enable* ) [inline], [static]

It is valid while zero cross detection is enabled. If ouput exceeds the threshold, DCDC would return CCM from DCM.

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |
| *enable* | Enable the feature or not. |

### 13.7.20 void DCDC_BootIntoDCM ( DCDC_Type * *base* )

pwd_zcd=0x0; pwd_cmp_offset=0x0; dcdc_loopctrl_en_rcscale=0x3 or 0x5; DCM_set_ctrl=1'b1;

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |

### 13.7.21 void DCDC_BootIntoCCM ( DCDC_Type * *base* )

pwd_zcd=0x1; pwd_cmp_offset=0x0; dcdc_loopctrl_en_rcscale=0x3;

Parameters

| | |
|---|---|
| *base* | DCDC peripheral base address. |

## 13.8 Variable Documentation

### 13.8.1 bool dcdc_detection_config_t::enableXtalokDetection

### 13.8.2 bool dcdc_detection_config_t::powerDownOverVoltageDetection

### 13.8.3 bool dcdc_detection_config_t::powerDownLowVlotageDetection

### 13.8.4 bool dcdc_detection_config_t::powerDownOverCurrentDetection

### 13.8.5 bool dcdc_detection_config_t::powerDownPeakCurrentDetection

### 13.8.6 bool dcdc_detection_config_t::powerDownZeroCrossDetection

### 13.8.7 dcdc_over_current_threshold_t dcdc_detection_config_t::OverCurrentThreshold

### 13.8.8 dcdc_peak_current_threshold_t dcdc_detection_config_t::PeakCurrentThreshold

### 13.8.9 bool dcdc_loop_control_config_t::enableCommonHysteresis

This feature will improve transient supply ripple and efficiency.

### 13.8.10 bool dcdc_loop_control_config_t::enableCommonThresholdDetection

### 13.8.11 bool dcdc_loop_control_config_t::enableInvertHysteresisSign

### 13.8.12 bool dcdc_loop_control_config_t::enableRCThresholdDetection

### 13.8.13 uint32_t dcdc_loop_control_config_t::enableRCScaleCircuit

Enable analog circuit of DC-DC converter to respond faster under transient load conditions.

### 13.8.14 uint32_t dcdc_loop_control_config_t::complementFeedForwardStep

Two's complement feed forward step in duty cycle in the switching DC-DC converter. Each time this field makes a transition from 0x0, the loop filter of the DC-DC converter is stepped once by a value proportional

to the change. This can be used to force a certain control loop behavior, such as improving response under known heavy load transients.

### 13.8.15 uint32_t dcdc_loop_control_config_t::controlParameterMagnitude

Magnitude of proportional control parameter in the switching DC-DC converter control loop.

### 13.8.16 uint32_t dcdc_loop_control_config_t::integralProportionalRatio

### 13.8.17 bool dcdc_low_power_config_t::enableOverloadDetection

### 13.8.18 bool dcdc_low_power_config_t::enableAdjustHystereticValue

### 13.8.19 dcdc_count_charging_time_period_t dcdc_low_power_config_t::countChargingTimePeriod

### 13.8.20 dcdc_count_charging_time_threshold_t dcdc_low_power_config_t::countChargingTimeThreshold

### 13.8.21 bool dcdc_internal_regulator_config_t::enableLoadResistor

### 13.8.22 uint32_t dcdc_internal_regulator_config_t::feedbackPoint

Select the feedback point of the internal regulator.

### 13.8.23 bool dcdc_min_power_config_t::enableUseHalfFreqForContinuous

# Chapter 14
# DCP: Data Co-Processor

## 14.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Data Co-Processor (DCP) module. For security purposes, the Data Co-Processor (DCP) provides hardware acceleration for the cryptographic algorithms. The features of DCP are: Encryption Algorithms: AES-128 (ECB and CBC modes), Hashing Algorithms: SHA-1 and SHA-256, modified CRC-32, Key selection from the SNVS, DCP internal key storage, or general memory, Internal Memory for storing up to four AES-128 keys-when a key is written to a key slot it can be read only by the DCP AES-128 engine, IP slave interface, and DMA.

The driver comprises two sets of API functions.

In the first set, blocking APIs are provided, for selected subset of operations supported by DCP hardware. The DCP operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until a DCP operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status.

The DCP work packets (descriptors) are placed on the system stack during the blocking API calls. The driver uses critical part (implemented as global interrupt enable/disable) for a short time whenever it needs to pass DCP work packets to DCP channel for processing. Therefore, the driver functions are designed to be re-entrant and as a consequence, one CPU thread can call one blocking API, such as AES Encrypt, while other CPU thread can call another blocking API, such as SHA-256 Update. The blocking functions provide typical interface to upper layer or application software. In the second set, non-blocking variants of the first set APIs are provided. Internally, the blocking APIs are implemented as a non-blocking operation start, followed by a blocking wait (CPU polling DCP work packet's status word) for an operation completion. The non-blocking functions allow upper layer to inject an application specific operation after the DCP operation start and DCP channel complete events. RTOS event wait and RTOS event set can be an example of such an operation.

## 14.2 DCP Driver Initialization and Configuration

Initialize DCP after Power On Reset or reset cycle Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/dcp

The DCP Driver is initialized by calling the DCP_Init() function. It enables the DCP module clock and configures DCP for operation.

### Key Management

The DCP implements four different key storage mechanisms: OTP-key, OTP-Unique key, Payload key, and SRAM-based keys that can be used by the software to securely store keys on a semi-permanent basis (kDCP_KeySlot0 ~ kDCP_KeySlot3). Once the function DCP_AES_SetKey() is called, it sets the AES key for encryption/decryption with the dcp_handle_t structure. In case the SRAM-based key is selected,

the function copies and holds the key in memory. In case the OTP key is used, please make sure to set DCP related IOMUXC_GPRs before DCP initialization, since the software reset of DCP must be issued to take these setting in effect. Refer to the DCP_OTPKeySelect() function in BEE driver example.

## 14.3 Comments about API usage in RTOS

DCP transactional (encryption or hash) APIs can be called from multiple threads.

## 14.4 Comments about API usage in interrupt handler

Assuming the host processor receiving interrupt has the ownership of the DCP module, it can request Encrypt/Decrypt/Hash/public_key operations in an interrupt routine. Additionally, as the DCP accesses system memory for it's operation with data (such as message, plaintext, ciphertext, or keys) all data should remain valid until the DCP operation completes.

## 14.5 DCP Driver Examples

### 14.5.1 Simple examples

Encrypt plaintext by AES engine Refer to the driver examples codes located at <SDK_ROOT>/boards/<-BOARD>/driver_examples/DCP

Compute hash (CRC-32) The CRC-32 algorithm implements a 32-bit CRC algorithm similar to the one used by Ethernet and many other protocols. The CRC differs from the Unix cksum() function in these four ways: The CRC initial value is 0xFFFFFFFF instead of 0x00000000, final XOR value is 0x00000000 instead of 0xFFFFFFFF, the logic pads the zeros to a 32-bit boundary for the trailing bytes, and it does not post-pend the file length. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOA-RD>/driver_examples/DCP

Compute hash (SHA-256) Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOA-RD>/driver_examples/DCP

## Modules

- DCP AES blocking driver
- DCP AES non-blocking driver
- DCP HASH driver

## Data Structures

- struct dcp_work_packet_t
  *DCP's work packet. More...*
- struct dcp_handle_t
  *Specify DCP's key resource and DCP channel. More...*
- struct dcp_context_t
  *DCP's context buffer, used by DCP for context switching between channels. More...*
- struct dcp_config_t
  *DCP's configuration structure. More...*

## Enumerations

- enum _dcp_ch_enable_t {
  kDCP_chDisable = 0U,
  kDCP_ch0Enable = 1U,
  kDCP_ch1Enable = 2U,
  kDCP_ch2Enable = 4U,
  kDCP_ch3Enable = 8U,
  kDCP_chEnableAll = 15U }
  *DCP channel enable.*
- enum _dcp_ch_int_enable_t {
  kDCP_chIntDisable = 0U,
  kDCP_ch0IntEnable = 1U,
  kDCP_ch1IntEnable = 2U,
  kDCP_ch2IntEnable = 4U,
  kDCP_ch3IntEnable = 8U }
  *DCP interrupt enable.*
- enum dcp_channel_t {
  kDCP_Channel0 = (1u << 16),
  kDCP_Channel1 = (1u << 17),
  kDCP_Channel2 = (1u << 18),
  kDCP_Channel3 = (1u << 19) }
  *DCP channel selection.*
- enum dcp_key_slot_t {
  kDCP_KeySlot0 = 0U,
  kDCP_KeySlot1 = 1U,
  kDCP_KeySlot2 = 2U,
  kDCP_KeySlot3 = 3U,
  kDCP_OtpKey = 4U,
  kDCP_OtpUniqueKey = 5U,
  kDCP_PayloadKey = 6U }
  *DCP key slot selection.*
- enum dcp_swap_t
  *DCP key, input & output swap options.*

## Functions

- void DCP_Init (DCP_Type *base, const dcp_config_t *config)
  *Enables clock to and enables DCP.*
- void DCP_Deinit (DCP_Type *base)
  *Disable DCP clock.*
- void DCP_GetDefaultConfig (dcp_config_t *config)
  *Gets the default configuration structure.*
- status_t DCP_WaitForChannelComplete (DCP_Type *base, dcp_handle_t *handle)
  *Poll and wait on DCP channel.*

## Driver version

- #define FSL_DCP_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

*DCP driver version.*

## 14.6 Data Structure Documentation

### 14.6.1 struct dcp_work_packet_t

### 14.6.2 struct dcp_handle_t

#### Data Fields

- dcp_channel_t channel
  *Specify DCP channel.*
- dcp_key_slot_t keySlot
  *For operations with key (such as AES encryption/decryption), specify DCP key slot.*
- uint32_t swapConfig
  *For configuration of key, input, output byte/word swap options.*

##### 14.6.2.0.0.6 Field Documentation

###### 14.6.2.0.0.6.1 dcp_channel_t dcp_handle_t::channel

###### 14.6.2.0.0.6.2 dcp_key_slot_t dcp_handle_t::keySlot

### 14.6.3 struct dcp_context_t

### 14.6.4 struct dcp_config_t

#### Data Fields

- bool gatherResidualWrites
  *Enable the ragged writes to the unaligned buffers.*
- bool enableContextCaching
  *Enable the caching of contexts between the operations.*
- bool enableContextSwitching
  *Enable automatic context switching for the channels.*
- uint8_t enableChannel
  *DCP channel enable.*
- uint8_t enableChannelInterrupt
  *Per-channel interrupt enable.*

#### 14.6.4.0.0.7 Field Documentation

##### 14.6.4.0.0.7.1 bool dcp_config_t::gatherResidualWrites

##### 14.6.4.0.0.7.2 bool dcp_config_t::enableContextCaching

##### 14.6.4.0.0.7.3 bool dcp_config_t::enableContextSwitching

##### 14.6.4.0.0.7.4 uint8_t dcp_config_t::enableChannel

##### 14.6.4.0.0.7.5 uint8_t dcp_config_t::enableChannelInterrupt

## 14.7 Macro Definition Documentation

### 14.7.1 #define FSL_DCP_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.1.1
  - Add DCP status clearing when channel operation is complete
- 2.1.0
  - Add byte/word swap feature for key, input and output data
- Version 2.0.0
  - Initial version

## 14.8 Enumeration Type Documentation

### 14.8.1 enum _dcp_ch_enable_t

Enumerator

***kDCP_chDisable*** DCP channel disable.
***kDCP_ch0Enable*** DCP channel 0 enable.
***kDCP_ch1Enable*** DCP channel 1 enable.
***kDCP_ch2Enable*** DCP channel 2 enable.
***kDCP_ch3Enable*** DCP channel 3 enable.
***kDCP_chEnableAll*** DCP channel enable all.

### 14.8.2 enum _dcp_ch_int_enable_t

Enumerator

***kDCP_chIntDisable*** DCP interrupts disable.
***kDCP_ch0IntEnable*** DCP channel 0 interrupt enable.

***kDCP_ch1IntEnable*** DCP channel 1 interrupt enable.
***kDCP_ch2IntEnable*** DCP channel 2 interrupt enable.
***kDCP_ch3IntEnable*** DCP channel 3 interrupt enable.

### 14.8.3 enum dcp_channel_t

Enumerator

***kDCP_Channel0*** DCP channel 0.
***kDCP_Channel1*** DCP channel 1.
***kDCP_Channel2*** DCP channel 2.
***kDCP_Channel3*** DCP channel 3.

### 14.8.4 enum dcp_key_slot_t

Enumerator

***kDCP_KeySlot0*** DCP key slot 0.
***kDCP_KeySlot1*** DCP key slot 1.
***kDCP_KeySlot2*** DCP key slot 2.
***kDCP_KeySlot3*** DCP key slot 3.
***kDCP_OtpKey*** DCP OTP key.
***kDCP_OtpUniqueKey*** DCP unique OTP key.
***kDCP_PayloadKey*** DCP payload key.

### 14.8.5 enum dcp_swap_t

## 14.9 Function Documentation

### 14.9.1 void DCP_Init ( DCP_Type * *base*, const dcp_config_t * *config* )

Enable DCP clock and configure DCP.

Parameters

| | |
|---|---|
| *base* | DCP base address |

| | |
|---|---|
| *config* | Pointer to configuration structure. |

### 14.9.2 void DCP_Deinit ( DCP_Type * *base* )

Reset DCP and Disable DCP clock.

Parameters

| | |
|---|---|
| *base* | DCP base address |

### 14.9.3 void DCP_GetDefaultConfig ( dcp_config_t * *config* )

This function initializes the DCP configuration structure to a default value. The default values are as follows. dcpConfig->gatherResidualWrites = true; dcpConfig->enableContextCaching = true; dcpConfig->enableContextSwitching = true; dcpConfig->enableChannnel = kDCP_chEnableAll; dcpConfig->enableChannelInterrupt = kDCP_chIntDisable;

Parameters

| | | |
|---|---|---|
| out | *config* | Pointer to configuration structure. |

### 14.9.4 status_t DCP_WaitForChannelComplete ( DCP_Type * *base,* dcp_handle_t * *handle* )

Polls the specified DCP channel until current it completes activity.

Parameters

| | |
|---|---|
| *base* | DCP peripheral base address. |
| *handle* | Specifies DCP channel. |

Returns

kStatus_Success When data processing completes without error.
kStatus_Fail When error occurs.

## 14.10 DCP AES blocking driver

### 14.10.1 Overview

This section describes the programming interface of the DCP AES blocking driver.

### Macros

- #define DCP_AES_BLOCK_SIZE 16
  *AES block size in bytes.*

### Functions

- status_t DCP_AES_SetKey (DCP_Type *base, dcp_handle_t *handle, const uint8_t *key, size_t keySize)
  *Set AES key to dcp_handle_t struct and optionally to DCP.*
- status_t DCP_AES_EncryptEcb (DCP_Type *base, dcp_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)
  *Encrypts AES on one or multiple 128-bit block(s).*
- status_t DCP_AES_DecryptEcb (DCP_Type *base, dcp_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)
  *Decrypts AES on one or multiple 128-bit block(s).*
- status_t DCP_AES_EncryptCbc (DCP_Type *base, dcp_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])
  *Encrypts AES using CBC block mode.*
- status_t DCP_AES_DecryptCbc (DCP_Type *base, dcp_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])
  *Decrypts AES using CBC block mode.*

### 14.10.2 Function Documentation

#### 14.10.2.1 status_t DCP_AES_SetKey ( DCP_Type * *base*, dcp_handle_t * *handle*, const uint8_t * *key*, size_t *keySize* )

Sets the AES key for encryption/decryption with the dcp_handle_t structure. The dcp_handle_t input argument specifies keySlot. If the keySlot is kDCP_OtpKey, the function will check the OTP_KEY_READY bit and will return it's ready to use status. For other keySlot selections, the function will copy and hold the key in dcp_handle_t struct. If the keySlot is one of the four DCP SRAM-based keys (one of kDCP_KeySlot0, kDCP_KeySlot1, kDCP_KeySlot2, kDCP_KeySlot3), this function will also load the supplied key to the specified keySlot in DCP.

Parameters

| | |
|---|---|
| *base* | DCP peripheral base address. |
| *handle* | Handle used for the request. |
| *key* | 0-mod-4 aligned pointer to AES key. |
| *keySize* | AES key size in bytes. Shall equal 16. |

Returns

status from set key operation

#### 14.10.2.2 status_t DCP_AES_EncryptEcb ( DCP_Type * *base*, dcp_handle_t * *handle*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size* )

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. |
| | *plaintext* | Input plain text to encrypt |
| out | *ciphertext* | Output cipher text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

Status from encrypt operation

#### 14.10.2.3 status_t DCP_AES_DecryptEcb ( DCP_Type * *base*, dcp_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size* )

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. |
| | *ciphertext* | Input plain text to encrypt |
| out | *plaintext* | Output cipher text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

Status from decrypt operation

#### 14.10.2.4 status_t DCP_AES_EncryptCbc ( DCP_Type * *base*, dcp_handle_t * *handle*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size*, const uint8_t *iv[16]* )

Encrypts AES using CBC block mode. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. |
| | *plaintext* | Input plain text to encrypt |
| out | *ciphertext* | Output cipher text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |
| | *iv* | Input initial vector to combine with the first input block. |

Returns

Status from encrypt operation

#### 14.10.2.5 status_t DCP_AES_DecryptCbc ( DCP_Type * *base*, dcp_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*, const uint8_t *iv[16]* )

Decrypts AES using CBC block mode. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. |
| | *ciphertext* | Input cipher text to decrypt |
| out | *plaintext* | Output plain text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |
| | *iv* | Input initial vector to combine with the first input block. |

Returns

Status from decrypt operation

## 14.11 DCP AES non-blocking driver

### 14.11.1 Overview

This section describes the programming interface of the DCP AES non-blocking driver.

### Functions

- status_t DCP_AES_EncryptEcbNonBlocking (DCP_Type *base, dcp_handle_t *handle, dcp_work_packet_t *dcpPacket, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)
  *Encrypts AES using the ECB block mode.*
- status_t DCP_AES_DecryptEcbNonBlocking (DCP_Type *base, dcp_handle_t *handle, dcp_work_packet_t *dcpPacket, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)
  *Decrypts AES using ECB block mode.*
- status_t DCP_AES_EncryptCbcNonBlocking (DCP_Type *base, dcp_handle_t *handle, dcp_work_packet_t *dcpPacket, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv)
  *Encrypts AES using CBC block mode.*
- status_t DCP_AES_DecryptCbcNonBlocking (DCP_Type *base, dcp_handle_t *handle, dcp_work_packet_t *dcpPacket, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv)
  *Decrypts AES using CBC block mode.*

### 14.11.2 Function Documentation

#### 14.11.2.1 status_t DCP_AES_EncryptEcbNonBlocking ( DCP_Type * *base*, dcp_handle_t * *handle*, dcp_work_packet_t * *dcpPacket*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size* )

Puts AES ECB encrypt work packet to DCP channel.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. |
| out | *dcpPacket* | Memory for the DCP work packet. |
| | *plaintext* | Input plain text to encrypt. |
| out | *ciphertext* | Output cipher text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

kStatus_Success The work packet has been scheduled at DCP channel.
kStatus_DCP_Again The DCP channel is busy processing previous request.

#### 14.11.2.2 status_t DCP_AES_DecryptEcbNonBlocking ( DCP_Type * *base,* dcp_handle_t * *handle,* dcp_work_packet_t * *dcpPacket,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* size_t *size* )

Puts AES ECB decrypt dcpPacket to DCP input job ring.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. |
| out | *dcpPacket* | Memory for the DCP work packet. |
| | *ciphertext* | Input cipher text to decrypt |
| out | *plaintext* | Output plain text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

kStatus_Success The work packet has been scheduled at DCP channel.
kStatus_DCP_Again The DCP channel is busy processing previous request.

#### 14.11.2.3 status_t DCP_AES_EncryptCbcNonBlocking ( DCP_Type * *base*, dcp_handle_t * *handle*, dcp_work_packet_t * *dcpPacket*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size*, const uint8_t * *iv* )

Puts AES CBC encrypt dcpPacket to DCP input job ring.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. Specifies jobRing. |
| out | *dcpPacket* | Memory for the DCP work packet. |
| | *plaintext* | Input plain text to encrypt |
| out | *ciphertext* | Output cipher text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |
| | *iv* | Input initial vector to combine with the first input block. |

Returns

kStatus_Success The work packet has been scheduled at DCP channel.
kStatus_DCP_Again The DCP channel is busy processing previous request.

#### 14.11.2.4 status_t DCP_AES_DecryptCbcNonBlocking ( DCP_Type * *base*, dcp_handle_t * *handle*, dcp_work_packet_t * *dcpPacket*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*, const uint8_t * *iv* )

Puts AES CBC decrypt dcpPacket to DCP input job ring.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for this request. Specifies jobRing. |
| out | *dcpPacket* | Memory for the DCP work packet. |
| | *ciphertext* | Input cipher text to decrypt |
| out | *plaintext* | Output plain text |
| | *size* | Size of input and output data in bytes. Must be multiple of 16 bytes. |
| | *iv* | Input initial vector to combine with the first input block. |

Returns

kStatus_Success The work packet has been scheduled at DCP channel.
kStatus_DCP_Again The DCP channel is busy processing previous request.

## 14.12 DCP HASH driver

### 14.12.1 Overview

This section describes the programming interface of the DCP HASH driver.

### Data Structures

- struct dcp_hash_ctx_t
  *Storage type used to save hash context. More...*

### Macros

- #define DCP_SHA_BLOCK_SIZE 128
  *DCP HASH Context size.*
- #define DCP_HASH_BLOCK_SIZE DCP_SHA_BLOCK_SIZE
  *DCP hash block size.*
- #define DCP_HASH_CTX_SIZE 58
  *DCP HASH Context size.*

### Enumerations

- enum dcp_hash_algo_t {
  kDCP_Sha1,
  kDCP_Sha256,
  kDCP_Crc32 }
  *Supported cryptographic block cipher functions for HASH creation.*

### Functions

- status_t DCP_HASH_Init (DCP_Type *base, dcp_handle_t *handle, dcp_hash_ctx_t *ctx, dcp_hash_algo_t algo)
  *Initialize HASH context.*
- status_t DCP_HASH_Update (DCP_Type *base, dcp_hash_ctx_t *ctx, const uint8_t *input, size_t inputSize)
  *Add data to current HASH.*
- status_t DCP_HASH_Finish (DCP_Type *base, dcp_hash_ctx_t *ctx, uint8_t *output, size_t *outputSize)
  *Finalize hashing.*
- status_t DCP_HASH (DCP_Type *base, dcp_handle_t *handle, dcp_hash_algo_t algo, const uint8_t *input, size_t inputSize, uint8_t *output, size_t *outputSize)
  *Create HASH on given data.*

### 14.12.2 Data Structure Documentation

#### 14.12.2.1 struct dcp_hash_ctx_t

### 14.12.3 Macro Definition Documentation

#### 14.12.3.1 #define DCP_SHA_BLOCK_SIZE 128

internal buffer block size

#### 14.12.3.2 #define DCP_HASH_CTX_SIZE 58

### 14.12.4 Enumeration Type Documentation

#### 14.12.4.1 enum dcp_hash_algo_t

Enumerator

***kDCP_Sha1*** SHA_1.
***kDCP_Sha256*** SHA_256.
***kDCP_Crc32*** CRC_32.

### 14.12.5 Function Documentation

#### 14.12.5.1 status_t DCP_HASH_Init ( DCP_Type * *base*, dcp_handle_t * *handle*, dcp_hash_ctx_t * *ctx*, dcp_hash_algo_t *algo* )

This function initializes the HASH.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Specifies the DCP channel used for hashing. |
| out | *ctx* | Output hash context |
| | *algo* | Underlaying algorithm to use for hash computation. |

Returns

Status of initialization

#### 14.12.5.2 status_t DCP_HASH_Update ( DCP_Type * *base*, dcp_hash_ctx_t * *ctx*, const uint8_t * *input*, size_t *inputSize* )

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus_Success, the running hash has been updated (DCP has processed the input data), so the memory at input pointer can be released back to system. The DCP context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| in,out | *ctx* | HASH context |
| | *input* | Input data |
| | *inputSize* | Size of input data in bytes |

Returns

Status of the hash update operation

#### 14.12.5.3 status_t DCP_HASH_Finish ( DCP_Type * *base*, dcp_hash_ctx_t * *ctx*, uint8_t * *output*, size_t * *outputSize* )

Outputs the final hash (computed by DCP_HASH_Update()) and erases the context.

Parameters

| | | |
|---|---|---|
| in,out | *ctx* | Input hash context |
| out | *output* | Output hash data |
| in,out | *outputSize* | Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes. |

Returns

Status of the hash finish operation

#### 14.12.5.4 status_t DCP_HASH ( DCP_Type * *base*, dcp_handle_t * *handle*, dcp_hash_algo_t *algo*, const uint8_t * *input*, size_t *inputSize*, uint8_t * *output*, size_t * *outputSize* )

Perform the full SHA or CRC32 in one function call. The function is blocking.

Parameters

| | | |
|---|---|---|
| | *base* | DCP peripheral base address |
| | *handle* | Handle used for the request. |
| | *algo* | Underlaying algorithm to use for hash computation. |
| | *input* | Input data |
| | *inputSize* | Size of input data in bytes |
| out | *output* | Output hash data |
| out | *outputSize* | Output parameter storing the size of the output hash in bytes |

Returns

Status of the one call hash operation.

# Chapter 15
# DMAMUX: Direct Memory Access Multiplexer Driver

## 15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access Multiplexer (DMAMUX) of MCUXpresso SDK devices.

## 15.2 Typical use case

### 15.2.1 DMAMUX Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/dmamux

## Driver version

- #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  *DMAMUX driver version 2.0.3.*

## DMAMUX Initialization and de-initialization

- void DMAMUX_Init (DMAMUX_Type *base)
  *Initializes the DMAMUX peripheral.*
- void DMAMUX_Deinit (DMAMUX_Type *base)
  *Deinitializes the DMAMUX peripheral.*

## DMAMUX Channel Operation

- static void DMAMUX_EnableChannel (DMAMUX_Type *base, uint32_t channel)
  *Enables the DMAMUX channel.*
- static void DMAMUX_DisableChannel (DMAMUX_Type *base, uint32_t channel)
  *Disables the DMAMUX channel.*
- static void DMAMUX_SetSource (DMAMUX_Type *base, uint32_t channel, uint32_t source)
  *Configures the DMAMUX channel source.*
- static void DMAMUX_EnablePeriodTrigger (DMAMUX_Type *base, uint32_t channel)
  *Enables the DMAMUX period trigger.*
- static void DMAMUX_DisablePeriodTrigger (DMAMUX_Type *base, uint32_t channel)
  *Disables the DMAMUX period trigger.*
- static void DMAMUX_EnableAlwaysOn (DMAMUX_Type *base, uint32_t channel, bool enable)
  *Enables the DMA channel to be always ON.*

## 15.3 Macro Definition Documentation

### 15.3.1 #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

## 15.4 Function Documentation

### 15.4.1 void DMAMUX_Init ( DMAMUX_Type * *base* )

This function ungates the DMAMUX clock.

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |

### 15.4.2 void DMAMUX_Deinit ( DMAMUX_Type * *base* )

This function gates the DMAMUX clock.

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |

### 15.4.3 static void DMAMUX_EnableChannel ( DMAMUX_Type * *base,* uint32_t *channel* ) [inline], [static]

This function enables the DMAMUX channel.

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |
| *channel* | DMAMUX channel number. |

### 15.4.4 static void DMAMUX_DisableChannel ( DMAMUX_Type * *base,* uint32_t *channel* ) [inline], [static]

This function disables the DMAMUX channel.

Note

The user must disable the DMAMUX channel before configuring it.

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |

| | |
|---|---|
| *channel* | DMAMUX channel number. |

### 15.4.5 static void DMAMUX_SetSource ( DMAMUX_Type * *base,* uint32_t *channel,* uint32_t *source* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |
| *channel* | DMAMUX channel number. |
| *source* | Channel source, which is used to trigger the DMA transfer. |

### 15.4.6 static void DMAMUX_EnablePeriodTrigger ( DMAMUX_Type * *base,* uint32_t *channel* ) [inline], [static]

This function enables the DMAMUX period trigger feature.

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |
| *channel* | DMAMUX channel number. |

### 15.4.7 static void DMAMUX_DisablePeriodTrigger ( DMAMUX_Type * *base,* uint32_t *channel* ) [inline], [static]

This function disables the DMAMUX period trigger.

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |
| *channel* | DMAMUX channel number. |

### 15.4.8 static void DMAMUX_EnableAlwaysOn ( DMAMUX_Type * *base,* uint32_t *channel,* bool *enable* ) [inline], [static]

This function enables the DMAMUX channel always ON feature.

Parameters

| | |
|---|---|
| *base* | DMAMUX peripheral base address. |
| *channel* | DMAMUX channel number. |
| *enable* | Switcher of the always ON feature. "true" means enabled, "false" means disabled. |

# Chapter 16
# eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

## 16.1 Overview

The MCUXpresso SDK provides a peripheral driver for the enhanced Direct Memory Access (eDMA) of MCUXpresso SDK devices.

## 16.2 Typical use case

### 16.2.1 eDMA Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/edma

## Data Structures

- struct edma_config_t
  *eDMA global configuration structure. More...*
- struct edma_transfer_config_t
  *eDMA transfer configuration More...*
- struct edma_channel_Preemption_config_t
  *eDMA channel priority configuration More...*
- struct edma_minor_offset_config_t
  *eDMA minor offset configuration More...*
- struct edma_tcd_t
  *eDMA TCD. More...*
- struct edma_handle_t
  *eDMA transfer handle structure More...*

## Macros

- #define DMA_DCHPRI_INDEX(channel) (((channel) & ~0x03U) | (3U - ((channel)&0x03U)))
  *Compute the offset unit from DCHPRI3.*

## Typedefs

- typedef void(* edma_callback )(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)
  *Define callback function for eDMA.*

**Typical use case**

## Enumerations

- enum edma_transfer_size_t {
  kEDMA_TransferSize1Bytes = 0x0U,
  kEDMA_TransferSize2Bytes = 0x1U,
  kEDMA_TransferSize4Bytes = 0x2U,
  kEDMA_TransferSize8Bytes = 0x3U,
  kEDMA_TransferSize16Bytes = 0x4U,
  kEDMA_TransferSize32Bytes = 0x5U }
  *eDMA transfer configuration*
- enum edma_modulo_t {
  kEDMA_ModuloDisable = 0x0U,
  kEDMA_Modulo2bytes,
  kEDMA_Modulo4bytes,
  kEDMA_Modulo8bytes,
  kEDMA_Modulo16bytes,
  kEDMA_Modulo32bytes,
  kEDMA_Modulo64bytes,
  kEDMA_Modulo128bytes,
  kEDMA_Modulo256bytes,
  kEDMA_Modulo512bytes,
  kEDMA_Modulo1Kbytes,
  kEDMA_Modulo2Kbytes,
  kEDMA_Modulo4Kbytes,
  kEDMA_Modulo8Kbytes,
  kEDMA_Modulo16Kbytes,
  kEDMA_Modulo32Kbytes,
  kEDMA_Modulo64Kbytes,
  kEDMA_Modulo128Kbytes,
  kEDMA_Modulo256Kbytes,
  kEDMA_Modulo512Kbytes,
  kEDMA_Modulo1Mbytes,
  kEDMA_Modulo2Mbytes,
  kEDMA_Modulo4Mbytes,
  kEDMA_Modulo8Mbytes,
  kEDMA_Modulo16Mbytes,
  kEDMA_Modulo32Mbytes,
  kEDMA_Modulo64Mbytes,
  kEDMA_Modulo128Mbytes,
  kEDMA_Modulo256Mbytes,
  kEDMA_Modulo512Mbytes,
  kEDMA_Modulo1Gbytes,
  kEDMA_Modulo2Gbytes }
  *eDMA modulo configuration*
- enum edma_bandwidth_t {

kEDMA_BandwidthStallNone = 0x0U,
kEDMA_BandwidthStall4Cycle = 0x2U,
kEDMA_BandwidthStall8Cycle = 0x3U }
*Bandwidth control.*

- enum edma_channel_link_type_t {
kEDMA_LinkNone = 0x0U,
kEDMA_MinorLink,
kEDMA_MajorLink }
*Channel link type.*
- enum _edma_channel_status_flags {
kEDMA_DoneFlag = 0x1U,
kEDMA_ErrorFlag = 0x2U,
kEDMA_InterruptFlag = 0x4U }
*eDMA channel status flags.*
- enum _edma_error_status_flags {
kEDMA_DestinationBusErrorFlag = DMA_ES_DBE_MASK,
kEDMA_SourceBusErrorFlag = DMA_ES_SBE_MASK,
kEDMA_ScatterGatherErrorFlag = DMA_ES_SGE_MASK,
kEDMA_NbytesErrorFlag = DMA_ES_NCE_MASK,
kEDMA_DestinationOffsetErrorFlag = DMA_ES_DOE_MASK,
kEDMA_DestinationAddressErrorFlag = DMA_ES_DAE_MASK,
kEDMA_SourceOffsetErrorFlag = DMA_ES_SOE_MASK,
kEDMA_SourceAddressErrorFlag = DMA_ES_SAE_MASK,
kEDMA_ErrorChannelFlag = DMA_ES_ERRCHN_MASK,
kEDMA_ChannelPriorityErrorFlag = DMA_ES_CPE_MASK,
kEDMA_TransferCanceledFlag = DMA_ES_ECX_MASK,
kEDMA_ValidFlag = (int)DMA_ES_VLD_MASK }
*eDMA channel error status flags.*
- enum edma_interrupt_enable_t {
kEDMA_ErrorInterruptEnable = 0x1U,
kEDMA_MajorInterruptEnable = DMA_CSR_INTMAJOR_MASK,
kEDMA_HalfInterruptEnable = DMA_CSR_INTHALF_MASK }
*eDMA interrupt source*
- enum edma_transfer_type_t {
kEDMA_MemoryToMemory = 0x0U,
kEDMA_PeripheralToMemory,
kEDMA_MemoryToPeripheral }
*eDMA transfer type*
- enum _edma_transfer_status {
kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),
kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }
*eDMA transfer status*

## Driver version

- #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))
*eDMA driver version*

**Typical use case**

## eDMA initialization and de-initialization

- void EDMA_Init (DMA_Type *base, const edma_config_t *config)
  *Initializes the eDMA peripheral.*
- void EDMA_Deinit (DMA_Type *base)
  *Deinitializes the eDMA peripheral.*
- void EDMA_InstallTCD (DMA_Type *base, uint32_t channel, edma_tcd_t *tcd)
  *Push content of TCD structure into hardware TCD register.*
- void EDMA_GetDefaultConfig (edma_config_t *config)
  *Gets the eDMA default configuration structure.*

## eDMA Channel Operation

- void EDMA_ResetChannel (DMA_Type *base, uint32_t channel)
  *Sets all TCD registers to default values.*
- void EDMA_SetTransferConfig (DMA_Type *base, uint32_t channel, const edma_transfer_config_t *config, edma_tcd_t *nextTcd)
  *Configures the eDMA transfer attribute.*
- void EDMA_SetMinorOffsetConfig (DMA_Type *base, uint32_t channel, const edma_minor_offset_config_t *config)
  *Configures the eDMA minor offset feature.*
- void EDMA_SetChannelPreemptionConfig (DMA_Type *base, uint32_t channel, const edma_channel_Preemption_config_t *config)
  *Configures the eDMA channel preemption feature.*
- void EDMA_SetChannelLink (DMA_Type *base, uint32_t channel, edma_channel_link_type_t type, uint32_t linkedChannel)
  *Sets the channel link for the eDMA transfer.*
- void EDMA_SetBandWidth (DMA_Type *base, uint32_t channel, edma_bandwidth_t bandWidth)
  *Sets the bandwidth for the eDMA transfer.*
- void EDMA_SetModulo (DMA_Type *base, uint32_t channel, edma_modulo_t srcModulo, edma_modulo_t destModulo)
  *Sets the source modulo and the destination modulo for the eDMA transfer.*
- static void EDMA_EnableAsyncRequest (DMA_Type *base, uint32_t channel, bool enable)
  *Enables an async request for the eDMA transfer.*
- static void EDMA_EnableAutoStopRequest (DMA_Type *base, uint32_t channel, bool enable)
  *Enables an auto stop request for the eDMA transfer.*
- void EDMA_EnableChannelInterrupts (DMA_Type *base, uint32_t channel, uint32_t mask)
  *Enables the interrupt source for the eDMA transfer.*
- void EDMA_DisableChannelInterrupts (DMA_Type *base, uint32_t channel, uint32_t mask)
  *Disables the interrupt source for the eDMA transfer.*

## eDMA TCD Operation

- void EDMA_TcdReset (edma_tcd_t *tcd)
  *Sets all fields to default values for the TCD structure.*
- void EDMA_TcdSetTransferConfig (edma_tcd_t *tcd, const edma_transfer_config_t *config, edma_tcd_t *nextTcd)
  *Configures the eDMA TCD transfer attribute.*
- void EDMA_TcdSetMinorOffsetConfig (edma_tcd_t *tcd, const edma_minor_offset_config_t *config)

*Configures the eDMA TCD minor offset feature.*

- void EDMA_TcdSetChannelLink (edma_tcd_t *tcd, edma_channel_link_type_t type, uint32_t linkedChannel)
  *Sets the channel link for the eDMA TCD.*
- static void EDMA_TcdSetBandWidth (edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
  *Sets the bandwidth for the eDMA TCD.*
- void EDMA_TcdSetModulo (edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo)
  *Sets the source modulo and the destination modulo for the eDMA TCD.*
- static void EDMA_TcdEnableAutoStopRequest (edma_tcd_t *tcd, bool enable)
  *Sets the auto stop request for the eDMA TCD.*
- void EDMA_TcdEnableInterrupts (edma_tcd_t *tcd, uint32_t mask)
  *Enables the interrupt source for the eDMA TCD.*
- void EDMA_TcdDisableInterrupts (edma_tcd_t *tcd, uint32_t mask)
  *Disables the interrupt source for the eDMA TCD.*

## eDMA Channel Transfer Operation

- static void EDMA_EnableChannelRequest (DMA_Type *base, uint32_t channel)
  *Enables the eDMA hardware channel request.*
- static void EDMA_DisableChannelRequest (DMA_Type *base, uint32_t channel)
  *Disables the eDMA hardware channel request.*
- static void EDMA_TriggerChannelStart (DMA_Type *base, uint32_t channel)
  *Starts the eDMA transfer by using the software trigger.*

## eDMA Channel Status Operation

- uint32_t EDMA_GetRemainingMajorLoopCount (DMA_Type *base, uint32_t channel)
  *Gets the remaining major loop count from the eDMA current channel TCD.*
- static uint32_t EDMA_GetErrorStatusFlags (DMA_Type *base)
  *Gets the eDMA channel error status flags.*
- uint32_t EDMA_GetChannelStatusFlags (DMA_Type *base, uint32_t channel)
  *Gets the eDMA channel status flags.*
- void EDMA_ClearChannelStatusFlags (DMA_Type *base, uint32_t channel, uint32_t mask)
  *Clears the eDMA channel status flags.*

## eDMA Transactional Operation

- void EDMA_CreateHandle (edma_handle_t *handle, DMA_Type *base, uint32_t channel)
  *Creates the eDMA handle.*
- void EDMA_InstallTCDMemory (edma_handle_t *handle, edma_tcd_t *tcdPool, uint32_t tcdSize)
  *Installs the TCDs memory pool into the eDMA handle.*
- void EDMA_SetCallback (edma_handle_t *handle, edma_callback callback, void *userData)
  *Installs a callback function for the eDMA transfer.*
- void EDMA_PrepareTransfer (edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth, void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, edma_transfer_type_t type)
  *Prepares the eDMA transfer structure.*
- status_t EDMA_SubmitTransfer (edma_handle_t *handle, const edma_transfer_config_t *config)
  *Submits the eDMA transfer request.*

- void EDMA_StartTransfer (edma_handle_t *handle)
  *eDMA starts transfer.*
- void EDMA_StopTransfer (edma_handle_t *handle)
  *eDMA stops transfer.*
- void EDMA_AbortTransfer (edma_handle_t *handle)
  *eDMA aborts transfer.*
- static uint32_t EDMA_GetUnusedTCDNumber (edma_handle_t *handle)
  *Get unused TCD slot number.*
- static uint32_t EDMA_GetNextTCDAddress (edma_handle_t *handle)
  *Get the next tcd address.*
- void EDMA_HandleIRQ (edma_handle_t *handle)
  *eDMA IRQ handler for the current major loop transfer completion.*

## 16.3 Data Structure Documentation

### 16.3.1 struct edma_config_t

### Data Fields

- bool enableContinuousLinkMode
  *Enable (true) continuous link mode.*
- bool enableHaltOnError
  *Enable (true) transfer halt on error.*
- bool enableRoundRobinArbitration
  *Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection.*
- bool enableDebugMode
  *Enable(true) eDMA debug mode.*

#### 16.3.1.0.0.8 Field Documentation

##### 16.3.1.0.0.8.1 bool edma_config_t::enableContinuousLinkMode

Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

##### 16.3.1.0.0.8.2 bool edma_config_t::enableHaltOnError

Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

##### 16.3.1.0.0.8.3 bool edma_config_t::enableDebugMode

When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

### 16.3.2 struct edma_transfer_config_t

This structure configures the source/destination transfer attribute.

## Data Fields

- uint32_t srcAddr
  *Source data address.*
- uint32_t destAddr
  *Destination data address.*
- edma_transfer_size_t srcTransferSize
  *Source data transfer size.*
- edma_transfer_size_t destTransferSize
  *Destination data transfer size.*
- int16_t srcOffset
  *Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.*
- int16_t destOffset
  *Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.*
- uint32_t minorLoopBytes
  *Bytes to transfer in a minor loop.*
- uint32_t majorLoopCounts
  *Major loop iteration count.*

#### 16.3.2.0.0.9 Field Documentation

#### 16.3.2.0.0.9.1 uint32_t edma_transfer_config_t::srcAddr

#### 16.3.2.0.0.9.2 uint32_t edma_transfer_config_t::destAddr

#### 16.3.2.0.0.9.3 edma_transfer_size_t edma_transfer_config_t::srcTransferSize

#### 16.3.2.0.0.9.4 edma_transfer_size_t edma_transfer_config_t::destTransferSize

#### 16.3.2.0.0.9.5 int16_t edma_transfer_config_t::srcOffset

#### 16.3.2.0.0.9.6 int16_t edma_transfer_config_t::destOffset

#### 16.3.2.0.0.9.7 uint32_t edma_transfer_config_t::majorLoopCounts

### 16.3.3 struct edma_channel_Preemption_config_t

## Data Fields

- bool enableChannelPreemption
  *If true: a channel can be suspended by other channel with higher priority.*
- bool enablePreemptAbility

*If true: a channel can suspend other channel with low priority.*
- uint8_t channelPriority
  *Channel priority.*

### 16.3.4 struct edma_minor_offset_config_t

## Data Fields

- bool enableSrcMinorOffset
  *Enable(true) or Disable(false) source minor loop offset.*
- bool enableDestMinorOffset
  *Enable(true) or Disable(false) destination minor loop offset.*
- uint32_t minorOffset
  *Offset for a minor loop mapping.*

#### 16.3.4.0.0.10 Field Documentation

#### 16.3.4.0.0.10.1 bool edma_minor_offset_config_t::enableSrcMinorOffset

#### 16.3.4.0.0.10.2 bool edma_minor_offset_config_t::enableDestMinorOffset

#### 16.3.4.0.0.10.3 uint32_t edma_minor_offset_config_t::minorOffset

### 16.3.5 struct edma_tcd_t

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

## Data Fields

- __IO uint32_t SADDR
  *SADDR register, used to save source address.*
- __IO uint16_t SOFF
  *SOFF register, save offset bytes every transfer.*
- __IO uint16_t ATTR
  *ATTR register, source/destination transfer size and modulo.*
- __IO uint32_t NBYTES
  *Nbytes register, minor loop length in bytes.*
- __IO uint32_t SLAST
  *SLAST register.*
- __IO uint32_t DADDR
  *DADDR register, used for destination address.*
- __IO uint16_t DOFF
  *DOFF register, used for destination offset.*
- __IO uint16_t CITER
  *CITER register, current minor loop numbers, for unfinished minor loop.*
- __IO uint32_t DLAST_SGA

*DLASTSGA register, next tcd address used in scatter-gather mode.*
- __IO uint16_t CSR
  *CSR register, for TCD control status.*
- __IO uint16_t BITER
  *BITER register, begin minor loop count.*

#### 16.3.5.0.0.11 Field Documentation

##### 16.3.5.0.0.11.1 __IO uint16_t edma_tcd_t::CITER

##### 16.3.5.0.0.11.2 __IO uint16_t edma_tcd_t::BITER

### 16.3.6 struct edma_handle_t

## Data Fields

- edma_callback callback
  *Callback function for major count exhausted.*
- void * userData
  *Callback function parameter.*
- DMA_Type * base
  *eDMA peripheral base address.*
- edma_tcd_t * tcdPool
  *Pointer to memory stored TCDs.*
- uint8_t channel
  *eDMA channel number.*
- volatile int8_t header
  *The first TCD index.*
- volatile int8_t tail
  *The last TCD index.*
- volatile int8_t tcdUsed
  *The number of used TCD slots.*
- volatile int8_t tcdSize
  *The total number of TCD slots in the queue.*
- uint8_t flags
  *The status of the current channel.*

#### 16.3.6.0.0.12 Field Documentation

##### 16.3.6.0.0.12.1 edma_callback edma_handle_t::callback

##### 16.3.6.0.0.12.2 void* edma_handle_t::userData

##### 16.3.6.0.0.12.3 DMA_Type* edma_handle_t::base

##### 16.3.6.0.0.12.4 edma_tcd_t* edma_handle_t::tcdPool

##### 16.3.6.0.0.12.5 uint8_t edma_handle_t::channel

##### 16.3.6.0.0.12.6 volatile int8_t edma_handle_t::header

Should point to the next TCD to be loaded into the eDMA engine.

##### 16.3.6.0.0.12.7 volatile int8_t edma_handle_t::tail

Should point to the next TCD to be stored into the memory pool.

##### 16.3.6.0.0.12.8 volatile int8_t edma_handle_t::tcdUsed

Should reflect the number of TCDs can be used/loaded in the memory.

##### 16.3.6.0.0.12.9 volatile int8_t edma_handle_t::tcdSize

##### 16.3.6.0.0.12.10 uint8_t edma_handle_t::flags

## 16.4 Macro Definition Documentation

### 16.4.1 #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))

Version 2.1.7.

## 16.5 Typedef Documentation

### 16.5.1 typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Parameters

| | |
|---|---|
| *handle* | EDMA handle pointer, users shall not touch the values inside. |
| *userData* | The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function. |
| *transferDone* | If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers. |
| *tcds* | How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this. |

## 16.6 Enumeration Type Documentation

### 16.6.1 enum edma_transfer_size_t

Enumerator

***kEDMA_TransferSize1Bytes*** Source/Destination data transfer size is 1 byte every time.
***kEDMA_TransferSize2Bytes*** Source/Destination data transfer size is 2 bytes every time.
***kEDMA_TransferSize4Bytes*** Source/Destination data transfer size is 4 bytes every time.
***kEDMA_TransferSize8Bytes*** Source/Destination data transfer size is 8 bytes every time.
***kEDMA_TransferSize16Bytes*** Source/Destination data transfer size is 16 bytes every time.
***kEDMA_TransferSize32Bytes*** Source/Destination data transfer size is 32 bytes every time.

### 16.6.2 enum edma_modulo_t

Enumerator

***kEDMA_ModuloDisable*** Disable modulo.
***kEDMA_Modulo2bytes*** Circular buffer size is 2 bytes.
***kEDMA_Modulo4bytes*** Circular buffer size is 4 bytes.
***kEDMA_Modulo8bytes*** Circular buffer size is 8 bytes.
***kEDMA_Modulo16bytes*** Circular buffer size is 16 bytes.
***kEDMA_Modulo32bytes*** Circular buffer size is 32 bytes.
***kEDMA_Modulo64bytes*** Circular buffer size is 64 bytes.
***kEDMA_Modulo128bytes*** Circular buffer size is 128 bytes.
***kEDMA_Modulo256bytes*** Circular buffer size is 256 bytes.
***kEDMA_Modulo512bytes*** Circular buffer size is 512 bytes.
***kEDMA_Modulo1Kbytes*** Circular buffer size is 1 K bytes.

***kEDMA_Modulo2Kbytes*** Circular buffer size is 2 K bytes.
***kEDMA_Modulo4Kbytes*** Circular buffer size is 4 K bytes.
***kEDMA_Modulo8Kbytes*** Circular buffer size is 8 K bytes.
***kEDMA_Modulo16Kbytes*** Circular buffer size is 16 K bytes.
***kEDMA_Modulo32Kbytes*** Circular buffer size is 32 K bytes.
***kEDMA_Modulo64Kbytes*** Circular buffer size is 64 K bytes.
***kEDMA_Modulo128Kbytes*** Circular buffer size is 128 K bytes.
***kEDMA_Modulo256Kbytes*** Circular buffer size is 256 K bytes.
***kEDMA_Modulo512Kbytes*** Circular buffer size is 512 K bytes.
***kEDMA_Modulo1Mbytes*** Circular buffer size is 1 M bytes.
***kEDMA_Modulo2Mbytes*** Circular buffer size is 2 M bytes.
***kEDMA_Modulo4Mbytes*** Circular buffer size is 4 M bytes.
***kEDMA_Modulo8Mbytes*** Circular buffer size is 8 M bytes.
***kEDMA_Modulo16Mbytes*** Circular buffer size is 16 M bytes.
***kEDMA_Modulo32Mbytes*** Circular buffer size is 32 M bytes.
***kEDMA_Modulo64Mbytes*** Circular buffer size is 64 M bytes.
***kEDMA_Modulo128Mbytes*** Circular buffer size is 128 M bytes.
***kEDMA_Modulo256Mbytes*** Circular buffer size is 256 M bytes.
***kEDMA_Modulo512Mbytes*** Circular buffer size is 512 M bytes.
***kEDMA_Modulo1Gbytes*** Circular buffer size is 1 G bytes.
***kEDMA_Modulo2Gbytes*** Circular buffer size is 2 G bytes.

### 16.6.3 enum edma_bandwidth_t

Enumerator

***kEDMA_BandwidthStallNone*** No eDMA engine stalls.
***kEDMA_BandwidthStall4Cycle*** eDMA engine stalls for 4 cycles after each read/write.
***kEDMA_BandwidthStall8Cycle*** eDMA engine stalls for 8 cycles after each read/write.

### 16.6.4 enum edma_channel_link_type_t

Enumerator

***kEDMA_LinkNone*** No channel link.
***kEDMA_MinorLink*** Channel link after each minor loop.
***kEDMA_MajorLink*** Channel link while major loop count exhausted.

### 16.6.5 enum _edma_channel_status_flags

Enumerator

***kEDMA_DoneFlag*** DONE flag, set while transfer finished, CITER value exhausted.

***kEDMA_ErrorFlag*** eDMA error flag, an error occurred in a transfer
***kEDMA_InterruptFlag*** eDMA interrupt flag, set while an interrupt occurred of this channel

### 16.6.6 enum _edma_error_status_flags

Enumerator

***kEDMA_DestinationBusErrorFlag*** Bus error on destination address.
***kEDMA_SourceBusErrorFlag*** Bus error on the source address.
***kEDMA_ScatterGatherErrorFlag*** Error on the Scatter/Gather address, not 32byte aligned.
***kEDMA_NbytesErrorFlag*** NBYTES/CITER configuration error.
***kEDMA_DestinationOffsetErrorFlag*** Destination offset not aligned with destination size.
***kEDMA_DestinationAddressErrorFlag*** Destination address not aligned with destination size.
***kEDMA_SourceOffsetErrorFlag*** Source offset not aligned with source size.
***kEDMA_SourceAddressErrorFlag*** Source address not aligned with source size.
***kEDMA_ErrorChannelFlag*** Error channel number of the cancelled channel number.
***kEDMA_ChannelPriorityErrorFlag*** Channel priority is not unique.
***kEDMA_TransferCanceledFlag*** Transfer cancelled.
***kEDMA_ValidFlag*** No error occurred, this bit is 0. Otherwise, it is 1.

### 16.6.7 enum edma_interrupt_enable_t

Enumerator

***kEDMA_ErrorInterruptEnable*** Enable interrupt while channel error occurs.
***kEDMA_MajorInterruptEnable*** Enable interrupt while major count exhausted.
***kEDMA_HalfInterruptEnable*** Enable interrupt while major count to half value.

### 16.6.8 enum edma_transfer_type_t

Enumerator

***kEDMA_MemoryToMemory*** Transfer from memory to memory.
***kEDMA_PeripheralToMemory*** Transfer from peripheral to memory.
***kEDMA_MemoryToPeripheral*** Transfer from memory to peripheral.

### 16.6.9 enum _edma_transfer_status

Enumerator

***kStatus_EDMA_QueueFull*** TCD queue is full.
***kStatus_EDMA_Busy*** Channel is busy and can't handle the transfer request.

## 16.7 Function Documentation

### 16.7.1 void EDMA_Init ( DMA_Type * *base,* const edma_config_t * *config* )

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure.

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *config* | A pointer to the configuration structure, see "edma_config_t". |

Note

This function enables the minor loop map feature.

### 16.7.2 void EDMA_Deinit ( DMA_Type * *base* )

This function gates the eDMA clock.

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |

### 16.7.3 void EDMA_InstallTCD ( DMA_Type * *base*, uint32_t *channel*, edma_tcd_t * *tcd* )

Parameters

| | |
|---|---|
| *base* | EDMA peripheral base address. |
| *channel* | EDMA channel number. |
| *tcd* | Point to TCD structure. |

### 16.7.4 void EDMA_GetDefaultConfig ( edma_config_t * *config* )

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

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

Parameters

| | |
|---|---|
| *config* | A pointer to the eDMA configuration structure. |

### 16.7.5 void EDMA_ResetChannel ( DMA_Type * *base*, uint32_t *channel* )

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

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |

Note

This function must not be called while the channel transfer is ongoing or it causes unpredictable results.
This function enables the auto stop request feature.

### 16.7.6 void EDMA_SetTransferConfig ( DMA_Type * *base*, uint32_t *channel*, const edma_transfer_config_t * *config*, edma_tcd_t * *nextTcd* )

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

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

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |

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

Note

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

### 16.7.7 void EDMA_SetMinorOffsetConfig ( DMA_Type * *base*, uint32_t *channel*, const edma_minor_offset_config_t * *config* )

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

Parameters

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

### 16.7.8 void EDMA_SetChannelPreemptionConfig ( DMA_Type * *base*, uint32_t *channel*, const edma_channel_Preemption_config_t * *config* )

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

Parameters

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

### 16.7.9 void EDMA_SetChannelLink ( DMA_Type * *base*, uint32_t *channel*, edma_channel_link_type_t *type*, uint32_t *linkedChannel* )

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

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |
| *type* | A channel link type, which can be one of the following:<br>• kEDMA_LinkNone<br>• kEDMA_MinorLink<br>• kEDMA_MajorLink |
| *linkedChannel* | The linked channel number. |

Note

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

### 16.7.10 void EDMA_SetBandWidth ( DMA_Type * *base*, uint32_t *channel*, edma_bandwidth_t *bandWidth* )

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

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |
| *bandWidth* | A bandwidth setting, which can be one of the following:<br>• kEDMABandwidthStallNone<br>• kEDMABandwidthStall4Cycle<br>• kEDMABandwidthStall8Cycle |

### 16.7.11 void EDMA_SetModulo ( DMA_Type * *base*, uint32_t *channel*, edma_modulo_t *srcModulo*, edma_modulo_t *destModulo* )

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

Parameters

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

### 16.7.12 static void EDMA_EnableAsyncRequest ( DMA_Type * *base,* uint32_t *channel,* bool *enable* ) [inline], [static]

Parameters

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

### 16.7.13 static void EDMA_EnableAutoStopRequest ( DMA_Type * *base,* uint32_t *channel,* bool *enable* ) [inline], [static]

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

Parameters

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

### 16.7.14 void EDMA_EnableChannelInterrupts ( DMA_Type * *base,* uint32_t *channel,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |
| *mask* | The mask of interrupt source to be set. Users need to use the defined edma_interrupt-_enable_t type. |

### 16.7.15 void EDMA_DisableChannelInterrupts ( DMA_Type * *base*, uint32_t *channel*, uint32_t *mask* )

Parameters

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

### 16.7.16 void EDMA_TcdReset ( edma_tcd_t * *tcd* )

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

Parameters

| | |
|---|---|
| *tcd* | Pointer to the TCD structure. |

Note

This function enables the auto stop request feature.

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

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

```
*   edma_transfer_t config = {
*   ...
*   }
*   edma_tcd_t tcd __aligned(32);
*   edma_tcd_t nextTcd __aligned(32);
```

```
*    EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
*
```

Parameters

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

Note

TCD address should be 32 bytes aligned or it causes an eDMA error.
If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

### 16.7.18 void EDMA_TcdSetMinorOffsetConfig ( edma_tcd_t * *tcd,* const edma_minor_offset_config_t * *config* )

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

Parameters

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

### 16.7.19 void EDMA_TcdSetChannelLink ( edma_tcd_t * *tcd,* edma_channel_link_type_t *type,* uint32_t *linkedChannel* )

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

Note

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

Parameters

| | |
|---|---|
| *tcd* | Point to the TCD structure. |
| *type* | Channel link type, it can be one of:<br>• kEDMA_LinkNone<br>• kEDMA_MinorLink<br>• kEDMA_MajorLink |
| *linkedChannel* | The linked channel number. |

### 16.7.20 static void EDMA_TcdSetBandWidth ( edma_tcd_t * *tcd,* edma_bandwidth_t *bandWidth* ) [inline], [static]

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

Parameters

| | |
|---|---|
| *tcd* | A pointer to the TCD structure. |
| *bandWidth* | A bandwidth setting, which can be one of the following:<br>• kEDMABandwidthStallNone<br>• kEDMABandwidthStall4Cycle<br>• kEDMABandwidthStall8Cycle |

### 16.7.21 void EDMA_TcdSetModulo ( edma_tcd_t * *tcd,* edma_modulo_t *srcModulo,* edma_modulo_t *destModulo* )

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

Parameters

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

### 16.7.22 static void EDMA_TcdEnableAutoStopRequest ( edma_tcd_t * *tcd,* bool *enable* ) [inline], [static]

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

Parameters

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

### 16.7.23 void EDMA_TcdEnableInterrupts ( edma_tcd_t * *tcd,* uint32_t *mask* )

Parameters

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

### 16.7.24 void EDMA_TcdDisableInterrupts ( edma_tcd_t * *tcd,* uint32_t *mask* )

Parameters

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

### 16.7.25 static void EDMA_EnableChannelRequest ( DMA_Type * *base,* uint32_t *channel* ) [inline], [static]

This function enables the hardware channel request.

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |

### 16.7.26 static void EDMA_DisableChannelRequest ( DMA_Type * *base,* uint32_t *channel* ) [inline], [static]

This function disables the hardware channel request.

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |

### 16.7.27 static void EDMA_TriggerChannelStart ( DMA_Type * *base*, uint32_t *channel* ) [inline], [static]

This function starts a minor loop transfer.

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |

### 16.7.28 uint32_t EDMA_GetRemainingMajorLoopCount ( DMA_Type * *base*, uint32_t *channel* )

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

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |

Returns

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

Note

1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.
   1. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of

NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTES(initially configured)

### 16.7.29 static uint32_t EDMA_GetErrorStatusFlags ( DMA_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |

Returns

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

### 16.7.30 uint32_t EDMA_GetChannelStatusFlags ( DMA_Type * *base,* uint32_t *channel* )

Parameters

| | |
|---|---|
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |

Returns

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

### 16.7.31 void EDMA_ClearChannelStatusFlags ( DMA_Type * *base,* uint32_t *channel,* uint32_t *mask* )

Parameters

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

### 16.7.32 void EDMA_CreateHandle ( edma_handle_t * *handle*, DMA_Type * *base*, uint32_t *channel* )

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

| | |
|---|---|
| *handle* | eDMA handle pointer. The eDMA handle stores callback function and parameters. |
| *base* | eDMA peripheral base address. |
| *channel* | eDMA channel number. |

### 16.7.33 void EDMA_InstallTCDMemory ( edma_handle_t * *handle*, edma_tcd_t * *tcdPool*, uint32_t *tcdSize* )

This function is called after the EDMA_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA_SubmitTransfer.

Parameters

| | |
|---|---|
| *handle* | eDMA handle pointer. |
| *tcdPool* | A memory pool to store TCDs. It must be 32 bytes aligned. |
| *tcdSize* | The number of TCD slots. |

### 16.7.34 void EDMA_SetCallback ( edma_handle_t * *handle*, edma_callback *callback*, void * *userData* )

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

| | |
|---|---|
| *handle* | eDMA handle pointer. |
| *callback* | eDMA callback function pointer. |
| *userData* | A parameter for the callback function. |

### 16.7.35 void EDMA_PrepareTransfer ( edma_transfer_config_t * *config*, void * *srcAddr*, uint32_t *srcWidth*, void * *destAddr*, uint32_t *destWidth*, uint32_t *bytesEachRequest*, uint32_t *transferBytes*, edma_transfer_type_t *type* )

This function prepares the transfer configuration structure according to the user input.

Parameters

| | |
|---|---|
| *config* | The user configuration structure of type edma_transfer_t. |
| *srcAddr* | eDMA transfer source address. |
| *srcWidth* | eDMA transfer source address width(bytes). |
| *destAddr* | eDMA transfer destination address. |
| *destWidth* | eDMA transfer destination address width(bytes). |
| *bytesEach-Request* | eDMA transfer bytes per channel request. |
| *transferBytes* | eDMA transfer bytes to be transferred. |
| *type* | eDMA transfer type. |

Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

### 16.7.36 status_t EDMA_SubmitTransfer ( edma_handle_t * *handle,* const edma_transfer_config_t * *config* )

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

Parameters

| | |
|---|---|
| *handle* | eDMA handle pointer. |
| *config* | Pointer to eDMA transfer configuration structure. |

Return values

| | |
|---|---|
| *kStatus_EDMA_Success* | It means submit transfer request succeed. |
| *kStatus_EDMA_Queue-Full* | It means TCD queue is full. Submit transfer request is not allowed. |

| | |
|---|---|
| *kStatus_EDMA_Busy* | It means the given channel is busy, need to submit request later. |

### 16.7.37 void EDMA_StartTransfer ( edma_handle_t * *handle* )

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

| | |
|---|---|
| *handle* | eDMA handle pointer. |

### 16.7.38 void EDMA_StopTransfer ( edma_handle_t * *handle* )

This function disables the channel request to pause the transfer. Users can call EDMA_StartTransfer() again to resume the transfer.

Parameters

| | |
|---|---|
| *handle* | eDMA handle pointer. |

### 16.7.39 void EDMA_AbortTransfer ( edma_handle_t * *handle* )

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

| | |
|---|---|
| *handle* | DMA handle pointer. |

### 16.7.40 static uint32_t EDMA_GetUnusedTCDNumber ( edma_handle_t * *handle* ) [inline], [static]

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

| | |
|---|---|
| *handle* | DMA handle pointer. |

Returns

The unused tcd slot number.

### 16.7.41 static uint32_t EDMA_GetNextTCDAddress ( edma_handle_t * *handle* ) [inline], [static]

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

| | |
|---|---|
| *handle* | DMA handle pointer. |

Returns

The next TCD address.

### 16.7.42 void EDMA_HandleIRQ ( edma_handle_t * *handle* )

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description part of the Reference Manual for further details.

Parameters

| | |
|---|---|
| *handle* | eDMA handle pointer. |

# Chapter 17
# eLCDIF: Enhanced LCD Interface

## 17.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Enhanced LCD Interface(eLCDIF)

The Enhanced LCD Interface supports MPU mode, VSYNC mode, RGB mode (or DOTCLK mode), and DVI mode. The current eLCDIF driver only supports RGB mode.

## 17.2 Typical use case

### 17.2.1 Frame buffer update

The function ELCDIF_SetNextBufferAddr sets the next frame to show to eLCDIF, the eLCDIF loads the new frame and sets the interrupt kELCDIF_CurFrameDone. If no new frame is set, the old one is displayed.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/elcdif

### 17.2.2 Alpha surface

The alpha surface can be enabled to add an extra overlay on the normal display buffer. In this example, the alpha surface is enabled, and the alpha value is updated after every frame loaded to eLCDIF.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/elcdif

## Data Structures

- struct elcdif_pixel_format_reg_t
  *The register value when using different pixel format. More...*
- struct elcdif_rgb_mode_config_t
  *eLCDIF configure structure for RGB mode (DOTCLK mode). More...*
- struct elcdif_as_buffer_config_t
  *eLCDIF alpha surface buffer configuration. More...*
- struct elcdif_as_blend_config_t
  *eLCDIF alpha surface blending configuration. More...*

**Typical use case**

## Enumerations

- enum _elcdif_polarity_flags {
  kELCDIF_VsyncActiveLow = 0U,
  kELCDIF_VsyncActiveHigh = LCDIF_VDCTRL0_VSYNC_POL_MASK,
  kELCDIF_HsyncActiveLow = 0U,
  kELCDIF_HsyncActiveHigh = LCDIF_VDCTRL0_HSYNC_POL_MASK,
  kELCDIF_DataEnableActiveLow = 0U,
  kELCDIF_DataEnableActiveHigh = LCDIF_VDCTRL0_ENABLE_POL_MASK,
  kELCDIF_DriveDataOnFallingClkEdge = 0U,
  kELCDIF_DriveDataOnRisingClkEdge = LCDIF_VDCTRL0_DOTCLK_POL_MASK }
  *eLCDIF signal polarity flags*
- enum _elcdif_interrupt_enable {
  kELCDIF_BusMasterErrorInterruptEnable = LCDIF_CTRL1_BM_ERROR_IRQ_EN_MASK,
  kELCDIF_TxFifoOverflowInterruptEnable = LCDIF_CTRL1_OVERFLOW_IRQ_EN_MASK,
  kELCDIF_TxFifoUnderflowInterruptEnable = LCDIF_CTRL1_UNDERFLOW_IRQ_EN_MASK,
  kELCDIF_CurFrameDoneInterruptEnable,
  kELCDIF_VsyncEdgeInterruptEnable }
  *The eLCDIF interrupts to enable.*
- enum _elcdif_interrupt_flags {
  kELCDIF_BusMasterError = LCDIF_CTRL1_BM_ERROR_IRQ_MASK,
  kELCDIF_TxFifoOverflow = LCDIF_CTRL1_OVERFLOW_IRQ_MASK,
  kELCDIF_TxFifoUnderflow = LCDIF_CTRL1_UNDERFLOW_IRQ_MASK,
  kELCDIF_CurFrameDone,
  kELCDIF_VsyncEdge = LCDIF_CTRL1_VSYNC_EDGE_IRQ_MASK }
  *The eLCDIF interrupt status flags.*
- enum _elcdif_status_flags {
  kELCDIF_LFifoFull = LCDIF_STAT_LFIFO_FULL_MASK,
  kELCDIF_LFifoEmpty = LCDIF_STAT_LFIFO_EMPTY_MASK,
  kELCDIF_TxFifoFull = LCDIF_STAT_TXFIFO_FULL_MASK,
  kELCDIF_TxFifoEmpty = LCDIF_STAT_TXFIFO_EMPTY_MASK }
  *eLCDIF status flags*
- enum elcdif_pixel_format_t {
  kELCDIF_PixelFormatRAW8 = 0,
  kELCDIF_PixelFormatRGB565 = 1,
  kELCDIF_PixelFormatRGB666 = 2,
  kELCDIF_PixelFormatXRGB8888 = 3,
  kELCDIF_PixelFormatRGB888 = 4 }
  *The pixel format.*
- enum elcdif_lcd_data_bus_t {
  kELCDIF_DataBus8Bit = LCDIF_CTRL_LCD_DATABUS_WIDTH(1),
  kELCDIF_DataBus16Bit = LCDIF_CTRL_LCD_DATABUS_WIDTH(0),
  kELCDIF_DataBus18Bit = LCDIF_CTRL_LCD_DATABUS_WIDTH(2),
  kELCDIF_DataBus24Bit = LCDIF_CTRL_LCD_DATABUS_WIDTH(3) }
  *The LCD data bus type.*
- enum elcdif_as_pixel_format_t {

kELCDIF_AsPixelFormatARGB8888 = 0x0,
kELCDIF_AsPixelFormatRGB888 = 0x4,
kELCDIF_AsPixelFormatARGB1555 = 0x8,
kELCDIF_AsPixelFormatARGB4444 = 0x9,
kELCDIF_AsPixelFormatRGB555 = 0xC,
kELCDIF_AsPixelFormatRGB444 = 0xD,
kELCDIF_AsPixelFormatRGB565 = 0xE }
*eLCDIF alpha surface pixel format.*

- enum elcdif_alpha_mode_t {
  kELCDIF_AlphaEmbedded,
  kELCDIF_AlphaOverride,
  kELCDIF_AlphaMultiply,
  kELCDIF_AlphaRop }
  *eLCDIF alpha mode during blending.*
- enum elcdif_rop_mode_t {
  kELCDIF_RopMaskAs = 0x0,
  kELCDIF_RopMaskNotAs = 0x1,
  kELCDIF_RopMaskAsNot = 0x2,
  kELCDIF_RopMergeAs = 0x3,
  kELCDIF_RopMergeNotAs = 0x4,
  kELCDIF_RopMergeAsNot = 0x5,
  kELCDIF_RopNotCopyAs = 0x6,
  kELCDIF_RopNot = 0x7,
  kELCDIF_RopNotMaskAs = 0x8,
  kELCDIF_RopNotMergeAs = 0x9,
  kELCDIF_RopXorAs = 0xA,
  kELCDIF_RopNotXorAs = 0xB }
  *eLCDIF ROP mode during blending.*
- enum elcdif_lut_t {
  kELCDIF_Lut0 = 0,
  kELCDIF_Lut1 }
  *eLCDIF LUT*

## Driver version

- #define FSL_ELCDIF_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *eLCDIF driver version*

## eLCDIF initialization and de-initialization

- void ELCDIF_RgbModeInit (LCDIF_Type *base, const elcdif_rgb_mode_config_t *config)
  *Initializes the eLCDIF to work in RGB mode (DOTCLK mode).*
- static uint32_t ELCDIF_GetStatus (LCDIF_Type *base)
  *Gets the eLCDIF default configuration structure for RGB (DOTCLK) mode.*
- static uint32_t ELCDIF_GetLFifoCount (LCDIF_Type *base)
  *Get current count in Latency buffer (LFIFO).*

### Interrupts

- static void ELCDIF_EnableInterrupts (LCDIF_Type *base, uint32_t mask)
  *Enables eLCDIF interrupt requests.*
- static void ELCDIF_DisableInterrupts (LCDIF_Type *base, uint32_t mask)
  *Disables eLCDIF interrupt requests.*
- static uint32_t ELCDIF_GetInterruptStatus (LCDIF_Type *base)
  *Get eLCDIF interrupt peding status.*
- static void ELCDIF_ClearInterruptStatus (LCDIF_Type *base, uint32_t mask)
  *Clear eLCDIF interrupt peding status.*

## 17.3 Data Structure Documentation

### 17.3.1 struct elcdif_pixel_format_reg_t

These register bits control the pixel format:

- CTRL[DATA_FORMAT_24_BIT]
- CTRL[DATA_FORMAT_18_BIT]
- CTRL[DATA_FORMAT_16_BIT]
- CTRL[WORD_LENGTH]
- CTRL1[BYTE_PACKING_FORMAT]

#### Data Fields

- uint32_t regCtrl
  *Value of register CTRL.*
- uint32_t regCtrl1
  *Value of register CTRL1.*

##### 17.3.1.0.0.13 Field Documentation

###### 17.3.1.0.0.13.1 uint32_t elcdif_pixel_format_reg_t::regCtrl

###### 17.3.1.0.0.13.2 uint32_t elcdif_pixel_format_reg_t::regCtrl1

### 17.3.2 struct elcdif_rgb_mode_config_t

#### Data Fields

- uint16_t panelWidth
  *Display panel width, pixels per line.*
- uint16_t panelHeight
  *Display panel height, how many lines per panel.*
- uint8_t hsw
  *HSYNC pulse width.*
- uint8_t hfp
  *Horizontal front porch.*

- uint8_t hbp
  *Horizontal back porch.*
- uint8_t vsw
  *VSYNC pulse width.*
- uint8_t vfp
  *Vrtical front porch.*
- uint8_t vbp
  *Vertical back porch.*
- uint32_t polarityFlags
  *OR'ed value of _elcdif_polarity_flags, used to contol the signal polarity.*
- uint32_t bufferAddr
  *Frame buffer address.*
- elcdif_pixel_format_t pixelFormat
  *Pixel format.*
- elcdif_lcd_data_bus_t dataBus
  *LCD data bus.*

#### 17.3.2.0.0.14 Field Documentation

#### 17.3.2.0.0.14.1 uint16_t elcdif_rgb_mode_config_t::panelWidth

#### 17.3.2.0.0.14.2 uint16_t elcdif_rgb_mode_config_t::panelHeight

#### 17.3.2.0.0.14.3 uint8_t elcdif_rgb_mode_config_t::hsw

#### 17.3.2.0.0.14.4 uint8_t elcdif_rgb_mode_config_t::hfp

#### 17.3.2.0.0.14.5 uint8_t elcdif_rgb_mode_config_t::hbp

#### 17.3.2.0.0.14.6 uint8_t elcdif_rgb_mode_config_t::vsw

#### 17.3.2.0.0.14.7 uint8_t elcdif_rgb_mode_config_t::vfp

#### 17.3.2.0.0.14.8 uint8_t elcdif_rgb_mode_config_t::vbp

#### 17.3.2.0.0.14.9 uint32_t elcdif_rgb_mode_config_t::polarityFlags

#### 17.3.2.0.0.14.10 uint32_t elcdif_rgb_mode_config_t::bufferAddr

#### 17.3.2.0.0.14.11 elcdif_pixel_format_t elcdif_rgb_mode_config_t::pixelFormat

#### 17.3.2.0.0.14.12 elcdif_lcd_data_bus_t elcdif_rgb_mode_config_t::dataBus

### 17.3.3 struct elcdif_as_buffer_config_t

## Data Fields

- uint32_t bufferAddr
  *Buffer address.*
- elcdif_as_pixel_format_t pixelFormat

*Pixel format.*

#### 17.3.3.0.0.15 Field Documentation

#### 17.3.3.0.0.15.1 uint32_t elcdif_as_buffer_config_t::bufferAddr

#### 17.3.3.0.0.15.2 elcdif_as_pixel_format_t elcdif_as_buffer_config_t::pixelFormat

### 17.3.4 struct elcdif_as_blend_config_t

### Data Fields

- uint8_t alpha
  *User defined alpha value, only used when alphaMode is kELCDIF_AlphaOverride or kELCDIF_AlphaRop.*
- bool invertAlpha
  *Set true to invert the alpha.*
- elcdif_alpha_mode_t alphaMode
  *Alpha mode.*
- elcdif_rop_mode_t ropMode
  *ROP mode, only valid when alphaMode is kELCDIF_AlphaRop.*

#### 17.3.4.0.0.16 Field Documentation

#### 17.3.4.0.0.16.1 uint8_t elcdif_as_blend_config_t::alpha

#### 17.3.4.0.0.16.2 bool elcdif_as_blend_config_t::invertAlpha

#### 17.3.4.0.0.16.3 elcdif_alpha_mode_t elcdif_as_blend_config_t::alphaMode

#### 17.3.4.0.0.16.4 elcdif_rop_mode_t elcdif_as_blend_config_t::ropMode

## 17.4 Macro Definition Documentation

### 17.4.1 #define FSL_ELCDIF_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Version 2.0.1.

## 17.5 Enumeration Type Documentation

### 17.5.1 enum _elcdif_polarity_flags

Enumerator

***kELCDIF_VsyncActiveLow*** VSYNC active low.
***kELCDIF_VsyncActiveHigh*** VSYNC active high.
***kELCDIF_HsyncActiveLow*** HSYNC active low.
***kELCDIF_HsyncActiveHigh*** HSYNC active high.
***kELCDIF_DataEnableActiveLow*** Data enable line active low.

***kELCDIF_DataEnableActiveHigh*** Data enable line active high.
***kELCDIF_DriveDataOnFallingClkEdge*** Drive data on falling clock edge, capture data on rising clock edge.
***kELCDIF_DriveDataOnRisingClkEdge*** Drive data on falling clock edge, capture data on rising clock edge.

### 17.5.2 enum _elcdif_interrupt_enable

Enumerator

***kELCDIF_BusMasterErrorInterruptEnable*** Bus master error interrupt.
***kELCDIF_TxFifoOverflowInterruptEnable*** TXFIFO overflow interrupt.
***kELCDIF_TxFifoUnderflowInterruptEnable*** TXFIFO underflow interrupt.
***kELCDIF_CurFrameDoneInterruptEnable*** Interrupt when hardware enters vertical blanking state.

***kELCDIF_VsyncEdgeInterruptEnable*** Interrupt when hardware encounters VSYNC edge.

### 17.5.3 enum _elcdif_interrupt_flags

Enumerator

***kELCDIF_BusMasterError*** Bus master error interrupt.
***kELCDIF_TxFifoOverflow*** TXFIFO overflow interrupt.
***kELCDIF_TxFifoUnderflow*** TXFIFO underflow interrupt.
***kELCDIF_CurFrameDone*** Interrupt when hardware enters vertical blanking state.
***kELCDIF_VsyncEdge*** Interrupt when hardware encounters VSYNC edge.

### 17.5.4 enum _elcdif_status_flags

Enumerator

***kELCDIF_LFifoFull*** LFIFO full.
***kELCDIF_LFifoEmpty*** LFIFO empty.
***kELCDIF_TxFifoFull*** TXFIFO full.
***kELCDIF_TxFifoEmpty*** TXFIFO empty.

### 17.5.5 enum elcdif_pixel_format_t

This enumerator should be defined together with the array s_pixelFormatReg. To support new pixel format, enhance this enumerator and s_pixelFormatReg.

Enumerator

***kELCDIF_PixelFormatRAW8*** RAW 8 bit, four data use 32 bits.
***kELCDIF_PixelFormatRGB565*** RGB565, two pixel use 32 bits.
***kELCDIF_PixelFormatRGB666*** RGB666 unpacked, one pixel uses 32 bits, high byte unused, upper 2 bits of other bytes unused.
***kELCDIF_PixelFormatXRGB8888*** XRGB8888 unpacked, one pixel uses 32 bits, high byte unused.

***kELCDIF_PixelFormatRGB888*** RGB888 packed, one pixel uses 24 bits.

### 17.5.6 enum elcdif_lcd_data_bus_t

Enumerator

***kELCDIF_DataBus8Bit*** 8-bit data bus.
***kELCDIF_DataBus16Bit*** 16-bit data bus, support RGB565.
***kELCDIF_DataBus18Bit*** 18-bit data bus, support RGB666.
***kELCDIF_DataBus24Bit*** 24-bit data bus, support RGB888.

### 17.5.7 enum elcdif_as_pixel_format_t

Enumerator

***kELCDIF_AsPixelFormatARGB8888*** 32-bit pixels with alpha.
***kELCDIF_AsPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)
***kELCDIF_AsPixelFormatARGB1555*** 16-bit pixels with alpha.
***kELCDIF_AsPixelFormatARGB4444*** 16-bit pixels with alpha.
***kELCDIF_AsPixelFormatRGB555*** 16-bit pixels without alpha.
***kELCDIF_AsPixelFormatRGB444*** 16-bit pixels without alpha.
***kELCDIF_AsPixelFormatRGB565*** 16-bit pixels without alpha.

### 17.5.8 enum elcdif_alpha_mode_t

Enumerator

***kELCDIF_AlphaEmbedded*** The alpha surface pixel alpha value will be used for blend.
***kELCDIF_AlphaOverride*** The user defined alpha value will be used for blend directly.
***kELCDIF_AlphaMultiply*** The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set set to 200, user defined alpha set to 100, then the reault alpha is 200 $*$ 100 / 255.
***kELCDIF_AlphaRop*** Raster operation.

### 17.5.9 enum elcdif_rop_mode_t

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Enumerator

***kELCDIF_RopMaskAs*** AS AND PS.
***kELCDIF_RopMaskNotAs*** nAS AND PS.
***kELCDIF_RopMaskAsNot*** AS AND nPS.
***kELCDIF_RopMergeAs*** AS OR PS.
***kELCDIF_RopMergeNotAs*** nAS OR PS.
***kELCDIF_RopMergeAsNot*** AS OR nPS.
***kELCDIF_RopNotCopyAs*** nAS.
***kELCDIF_RopNot*** nPS.
***kELCDIF_RopNotMaskAs*** AS NAND PS.
***kELCDIF_RopNotMergeAs*** AS NOR PS.
***kELCDIF_RopXorAs*** AS XOR PS.
***kELCDIF_RopNotXorAs*** AS XNOR PS.

### 17.5.10 enum elcdif_lut_t

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external displayer.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determins which memory to use.

Enumerator

***kELCDIF_Lut0*** LUT 0.
***kELCDIF_Lut1*** LUT 1.

## 17.6 Function Documentation

### 17.6.1 void ELCDIF_RgbModeInit ( LCDIF_Type * *base*, const elcdif_rgb_mode_config_t * *config* )

This function ungates the eLCDIF clock and configures the eLCDIF peripheral according to the configuration structure.

Parameters

| | |
|---|---|
| *base* | eLCDIF peripheral base address. |
| *config* | Pointer to the configuration structure. |

### 17.6.2 static uint32_t ELCDIF_GetStatus ( LCDIF_Type * *base* ) [inline], [static]

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

```
    config->panelWidth = 480U;
    config->panelHeight = 272U;
    config->hsw = 41;
    config->hfp = 4;
    config->hbp = 8;
    config->vsw = 10;
    config->vfp = 4;
    config->vbp = 2;
    config->polarityFlags = kELCDIF_VsyncActiveLow |
                            kELCDIF_HsyncActiveLow |
                            kELCDIF_DataEnableActiveLow |
                            kELCDIF_DriveDataOnFallingClkEdge;
    config->bufferAddr = 0U;
    config->pixelFormat = kELCDIF_PixelFormatRGB888;
    config->dataBus = kELCDIF_DataBus24Bit;
    @code
 *
 * @param config Pointer to the eLCDIF configuration structure.
 */
void ELCDIF_RgbModeGetDefaultConfig(elcdif_rgb_mode_config_t *config);

void ELCDIF_Deinit(LCDIF_Type *base);

/* @} */

void ELCDIF_RgbModeSetPixelFormat(LCDIF_Type *base, elcdif_pixel_format_t pixelFormat)
      ;

static inline void ELCDIF_RgbModeStart(LCDIF_Type *base)
{
    base->CTRL_SET = LCDIF_CTRL_RUN_MASK | LCDIF_CTRL_DOTCLK_MODE_MASK;
}

void ELCDIF_RgbModeStop(LCDIF_Type *base);

static inline void ELCDIF_SetNextBufferAddr(LCDIF_Type *base, uint32_t bufferAddr)
{
    base->NEXT_BUF = bufferAddr;
}

void ELCDIF_Reset(LCDIF_Type *base);

#if !(defined(FSL_FEATURE_LCDIF_HAS_NO_RESET_PIN) && FSL_FEATURE_LCDIF_HAS_NO_RESET_PIN)

static inline void ELCDIF_PullUpResetPin(LCDIF_Type *base, bool pullUp)
{
    if (pullUp)
    {
```

```
        base->CTRL1_SET = LCDIF_CTRL1_RESET_MASK;
    }
    else
    {
        base->CTRL1_CLR = LCDIF_CTRL1_RESET_MASK;
    }
}
#endif

static inline void ELCDIF_EnablePxpHandShake(LCDIF_Type *base, bool enable)
{
    if (enable)
    {
        base->CTRL_SET = LCDIF_CTRL_ENABLE_PXP_HANDSHAKE_MASK;
    }
    else
    {
        base->CTRL_CLR = LCDIF_CTRL_ENABLE_PXP_HANDSHAKE_MASK;
    }
}

/* @} */

static inline uint32_t ELCDIF_GetCrcValue(LCDIF_Type *base)
{
    return base->CRC_STAT;
}

static inline uint32_t ELCDIF_GetBusMasterErrorAddr(LCDIF_Type *base)
{
    return base->BM_ERROR_STAT;
}
```

Parameters

| | |
|---|---|
| *base* | eLCDIF peripheral base address. |

Returns

The mask value of status flags, it is OR'ed value of _elcdif_status_flags.

### 17.6.3 static uint32_t ELCDIF_GetLFifoCount ( LCDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | eLCDIF peripheral base address. |

Returns

The LFIFO current count

### 17.6.4 static void ELCDIF_EnableInterrupts ( LCDIF_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | eLCDIF peripheral base address. |
| *mask* | interrupt source, OR'ed value of _elcdif_interrupt_enable. |

### 17.6.5 static void ELCDIF_DisableInterrupts ( LCDIF_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | eLCDIF peripheral base address. |
| *mask* | interrupt source, OR'ed value of _elcdif_interrupt_enable. |

### 17.6.6 static uint32_t ELCDIF_GetInterruptStatus ( LCDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | eLCDIF peripheral base address. |

Returns

Interrupt pending status, OR'ed value of _elcdif_interrupt_flags.

### 17.6.7 static void ELCDIF_ClearInterruptStatus ( LCDIF_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | eLCDIF peripheral base address. |
| *mask* | of the flags to clear, OR'ed value of _elcdif_interrupt_flags. |

# Chapter 18
# ENC: Quadrature Encoder/Decoder

## 18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Quadrature Encoder/Decoder (ENC) module of MCUXpresso SDK devices.

This section describes the programming interface of the ENC Peripheral driver. The ENC driver configures the ENC module and provides a functional interface for the user to build the ENC application.

## 18.2 Function groups

### 18.2.1 Initialization and De-initialization

This function group initializes default configuration structure for the ENC counter and initializes ENC counter with the normal configuration and de-initialize ENC module. Some APIs are also created to control the features.

### 18.2.2 Status

This function group get/clear the ENC status.

### 18.2.3 Interrupts

This function group enable/disable the ENC interrupts.

### 18.2.4 Value Operation

This function group get the counter/hold value of positions.

## 18.3 Typical use case

### 18.3.1 Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/enc

## Data Structures

- struct enc_config_t

*Define user configuration structure for ENC module.* More...

- struct enc_self_test_config_t
  *Define configuration structure for self test module.* More...

## Macros

- #define FSL_ENC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *Version 2.0.0.*

## Enumerations

- enum _enc_interrupt_enable {
  kENC_HOMETransitionInterruptEnable = (1U << 0U),
  kENC_INDEXPulseInterruptEnable = (1U << 1U),
  kENC_WatchdogTimeoutInterruptEnable = (1U << 2U),
  kENC_PositionCompareInerruptEnable = (1U << 3U),
  kENC_SimultBothPhaseChangeInterruptEnable,
  kENC_PositionRollOverInterruptEnable = (1U << 5U),
  kENC_PositionRollUnderInterruptEnable = (1U << 6U) }
  *Interrupt enable/disable mask.*
- enum _enc_status_flags {
  kENC_HOMETransitionFlag = (1U << 0U),
  kENC_INDEXPulseFlag = (1U << 1U),
  kENC_WatchdogTimeoutFlag = (1U << 2U),
  kENC_PositionCompareFlag = (1U << 3U),
  kENC_SimultBothPhaseChangeFlag = (1U << 4U),
  kENC_PositionRollOverFlag = (1U << 5U),
  kENC_PositionRollUnderFlag = (1U << 6U),
  kENC_LastCountDirectionFlag = (1U << 7U) }
  *Status flag mask.*
- enum _enc_signal_status_flags {
  kENC_RawHOMEStatusFlag = ENC_IMR_HOME_MASK,
  kENC_RawINDEXStatusFlag = ENC_IMR_INDEX_MASK,
  kENC_RawPHBStatusFlag = ENC_IMR_PHB_MASK,
  kENC_RawPHAEXStatusFlag = ENC_IMR_PHA_MASK,
  kENC_FilteredHOMEStatusFlag = ENC_IMR_FHOM_MASK,
  kENC_FilteredINDEXStatusFlag = ENC_IMR_FIND_MASK,
  kENC_FilteredPHBStatusFlag = ENC_IMR_FPHB_MASK,
  kENC_FilteredPHAStatusFlag = ENC_IMR_FPHA_MASK }
  *Signal status flag mask.*
- enum enc_home_trigger_mode_t {
  kENC_HOMETriggerDisabled = 0U,
  kENC_HOMETriggerOnRisingEdge,
  kENC_HOMETriggerOnFallingEdge }
  *Define HOME signal's trigger mode.*
- enum enc_index_trigger_mode_t {

kENC_INDEXTriggerDisabled = 0U,
kENC_INDEXTriggerOnRisingEdge,
kENC_INDEXTriggerOnFallingEdge }
*Define INDEX signal's trigger mode.*

- enum enc_decoder_work_mode_t {
  kENC_DecoderWorkAsNormalMode = 0U,
  kENC_DecoderWorkAsSignalPhaseCountMode }
  *Define type for decoder work mode.*
- enum enc_position_match_mode_t {
  kENC_POSMATCHOnPositionCounterEqualToComapreValue = 0U,
  kENC_POSMATCHOnReadingAnyPositionCounter }
  *Define type for the condition of POSMATCH pulses.*
- enum enc_revolution_count_condition_t {
  kENC_RevolutionCountOnINDEXPulse = 0U,
  kENC_RevolutionCountOnRollOverModulus }
  *Define type for determining how the revolution counter (REV) is incremented/decremented.*
- enum enc_self_test_direction_t {
  kENC_SelfTestDirectionPositive = 0U,
  kENC_SelfTestDirectionNegative }
  *Define type for direction of self test generated signal.*

## Variables

- bool enc_config_t::enableReverseDirection
  *Enable reverse direction counting.*
- enc_decoder_work_mode_t enc_config_t::decoderWorkMode
  *Enable signal phase count mode.*
- enc_home_trigger_mode_t enc_config_t::HOMETriggerMode
  *Enable HOME to initialize position counters.*
- enc_index_trigger_mode_t enc_config_t::INDEXTriggerMode
  *Enable INDEX to initialize position counters.*
- bool enc_config_t::enableTRIGGERClearPositionCounter
  *Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.*
- bool enc_config_t::enableTRIGGERClearHoldPositionCounter
  *Enable update of hold registers on rising edge of TRIGGER, or not.*
- bool enc_config_t::enableWatchdog
  *Enable the watchdog to detect if the target is moving or not.*
- uint16_t enc_config_t::watchdogTimeoutValue
  *Watchdog timeout count value.*
- uint16_t enc_config_t::filterCount
  *Input Filter Sample Count.*
- uint16_t enc_config_t::filterSamplePeriod
  *Input Filter Sample Period.*
- enc_position_match_mode_t enc_config_t::positionMatchMode
  *The condition of POSMATCH pulses.*
- uint32_t enc_config_t::positionCompareValue
  *Position compare value.*
- enc_revolution_count_condition_t enc_config_t::revolutionCountCondition
  *Revolution Counter Modulus Enable.*
- bool enc_config_t::enableModuloCountMode

*Enable Modulo Counting.*
- uint32_t enc_config_t::positionModulusValue
  *Position modulus value.*
- uint32_t enc_config_t::positionInitialValue
  *Position initial value.*
- enc_self_test_direction_t enc_self_test_config_t::signalDirection
  *Direction of self test generated signal.*
- uint16_t enc_self_test_config_t::signalCount
  *Hold the number of quadrature advances to generate.*
- uint16_t enc_self_test_config_t::signalPeriod
  *Hold the period of quadrature phase in IPBus clock cycles.*

## Initialization and De-initialization

- void ENC_Init (ENC_Type *base, const enc_config_t *config)
  *Initialization for the ENC module.*
- void ENC_Deinit (ENC_Type *base)
  *De-initialization for the ENC module.*
- void ENC_GetDefaultConfig (enc_config_t *config)
  *Get an available pre-defined settings for ENC's configuration.*
- void ENC_DoSoftwareLoadInitialPositionValue (ENC_Type *base)
  *Load the initial position value to position counter.*
- void ENC_SetSelfTestConfig (ENC_Type *base, const enc_self_test_config_t *config)
  *Enable and configure the self test function.*
- void ENC_EnableWatchdog (ENC_Type *base, bool enable)
  *Enable watchdog for ENC module.*
- void ENC_SetInitialPositionValue (ENC_Type *base, uint32_t value)
  *Set initial position value for ENC module.*

## Status

- uint32_t ENC_GetStatusFlags (ENC_Type *base)
  *Get the status flags.*
- void ENC_ClearStatusFlags (ENC_Type *base, uint32_t mask)
  *Clear the status flags.*
- static uint16_t ENC_GetSignalStatusFlags (ENC_Type *base)
  *Get the signals' real-time status.*

## Interrupts

- void ENC_EnableInterrupts (ENC_Type *base, uint32_t mask)
  *Enable the interrupts.*
- void ENC_DisableInterrupts (ENC_Type *base, uint32_t mask)
  *Disable the interrupts.*
- uint32_t ENC_GetEnabledInterrupts (ENC_Type *base)
  *Get the enabled interrupts' flags.*

## Value Operation

- uint32_t ENC_GetPositionValue (ENC_Type *base)
  *Get the current position counter's value.*

- uint32_t ENC_GetHoldPositionValue (ENC_Type *base)
  *Get the hold position counter's value.*
- static uint16_t ENC_GetPositionDifferenceValue (ENC_Type *base)
  *Get the position difference counter's value.*
- static uint16_t ENC_GetHoldPositionDifferenceValue (ENC_Type *base)
  *Get the hold position difference counter's value.*
- static uint16_t ENC_GetRevolutionValue (ENC_Type *base)
  *Get the position revolution counter's value.*
- static uint16_t ENC_GetHoldRevolutionValue (ENC_Type *base)
  *Get the hold position revolution counter's value.*

## 18.4 Data Structure Documentation

### 18.4.1 struct enc_config_t

#### Data Fields

- bool enableReverseDirection
  *Enable reverse direction counting.*
- enc_decoder_work_mode_t decoderWorkMode
  *Enable signal phase count mode.*
- enc_home_trigger_mode_t HOMETriggerMode
  *Enable HOME to initialize position counters.*
- enc_index_trigger_mode_t INDEXTriggerMode
  *Enable INDEX to initialize position counters.*
- bool enableTRIGGERClearPositionCounter
  *Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.*
- bool enableTRIGGERClearHoldPositionCounter
  *Enable update of hold registers on rising edge of TRIGGER, or not.*
- bool enableWatchdog
  *Enable the watchdog to detect if the target is moving or not.*
- uint16_t watchdogTimeoutValue
  *Watchdog timeout count value.*
- uint16_t filterCount
  *Input Filter Sample Count.*
- uint16_t filterSamplePeriod
  *Input Filter Sample Period.*
- enc_position_match_mode_t positionMatchMode
  *The condition of POSMATCH pulses.*
- uint32_t positionCompareValue
  *Position compare value.*
- enc_revolution_count_condition_t revolutionCountCondition
  *Revolution Counter Modulus Enable.*
- bool enableModuloCountMode
  *Enable Modulo Counting.*
- uint32_t positionModulusValue
  *Position modulus value.*
- uint32_t positionInitialValue
  *Position initial value.*

### 18.4.2 struct enc_self_test_config_t

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

#### Data Fields

- enc_self_test_direction_t signalDirection
  *Direction of self test generated signal.*
- uint16_t signalCount
  *Hold the number of quadrature advances to generate.*
- uint16_t signalPeriod
  *Hold the period of quadrature phase in IPBus clock cycles.*

## 18.5 Macro Definition Documentation

### 18.5.1 #define FSL_ENC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

## 18.6 Enumeration Type Documentation

### 18.6.1 enum _enc_interrupt_enable

Enumerator

***kENC_HOMETransitionInterruptEnable*** HOME interrupt enable.
***kENC_INDEXPulseInterruptEnable*** INDEX pulse interrupt enable.
***kENC_WatchdogTimeoutInterruptEnable*** Watchdog timeout interrupt enable.
***kENC_PositionCompareInerruptEnable*** Position compare interrupt enable.
***kENC_SimultBothPhaseChangeInterruptEnable*** Simultaneous PHASEA and PHASEB change interrupt enable.
***kENC_PositionRollOverInterruptEnable*** Roll-over interrupt enable.
***kENC_PositionRollUnderInterruptEnable*** Roll-under interrupt enable.

### 18.6.2 enum _enc_status_flags

These flags indicate the counter's events.

Enumerator

***kENC_HOMETransitionFlag*** HOME signal transition interrupt request.
***kENC_INDEXPulseFlag*** INDEX Pulse Interrupt Request.
***kENC_WatchdogTimeoutFlag*** Watchdog timeout interrupt request.
***kENC_PositionCompareFlag*** Position compare interrupt request.
***kENC_SimultBothPhaseChangeFlag*** Simultaneous PHASEA and PHASEB change interrupt request.

***kENC_PositionRollOverFlag*** Roll-over interrupt request.
***kENC_PositionRollUnderFlag*** Roll-under interrupt request.
***kENC_LastCountDirectionFlag*** Last count was in the up direction, or the down direction.

### 18.6.3 enum _enc_signal_status_flags

These flags indicate the counter's signal.

Enumerator

***kENC_RawHOMEStatusFlag*** Raw HOME input.
***kENC_RawINDEXStatusFlag*** Raw INDEX input.
***kENC_RawPHBStatusFlag*** Raw PHASEB input.
***kENC_RawPHAEXStatusFlag*** Raw PHASEA input.
***kENC_FilteredHOMEStatusFlag*** The filtered version of HOME input.
***kENC_FilteredINDEXStatusFlag*** The filtered version of INDEX input.
***kENC_FilteredPHBStatusFlag*** The filtered version of PHASEB input.
***kENC_FilteredPHAStatusFlag*** The filtered version of PHASEA input.

### 18.6.4 enum enc_home_trigger_mode_t

The ENC would count the trigger from HOME signal line.

Enumerator

***kENC_HOMETriggerDisabled*** HOME signal's trigger is disabled.
***kENC_HOMETriggerOnRisingEdge*** Use positive going edge-to-trigger initialization of position counters.
***kENC_HOMETriggerOnFallingEdge*** Use negative going edge-to-trigger initialization of position counters.

### 18.6.5 enum enc_index_trigger_mode_t

The ENC would count the trigger from INDEX signal line.

Enumerator

***kENC_INDEXTriggerDisabled*** INDEX signal's trigger is disabled.
***kENC_INDEXTriggerOnRisingEdge*** Use positive going edge-to-trigger initialization of position counters.
***kENC_INDEXTriggerOnFallingEdge*** Use negative going edge-to-trigger initialization of position counters.

### 18.6.6 enum enc_decoder_work_mode_t

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

Enumerator

***kENC_DecoderWorkAsNormalMode*** Use standard quadrature decoder with PHASEA and PHASEB.

***kENC_DecoderWorkAsSignalPhaseCountMode*** PHASEA input generates a count signal while PHASEB input control the direction.

### 18.6.7 enum enc_position_match_mode_t

Enumerator

***kENC_POSMATCHOnPositionCounterEqualToComapreValue*** POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

***kENC_POSMATCHOnReadingAnyPositionCounter*** POSMATCH pulses when any position counter register is read.

### 18.6.8 enum enc_revolution_count_condition_t

Enumerator

***kENC_RevolutionCountOnINDEXPulse*** Use INDEX pulse to increment/decrement revolution counter.

***kENC_RevolutionCountOnRollOverModulus*** Use modulus counting roll-over/under to increment/decrement revolution counter.

### 18.6.9 enum enc_self_test_direction_t

Enumerator

***kENC_SelfTestDirectionPositive*** Self test generates the signal in positive direction.

***kENC_SelfTestDirectionNegative*** Self test generates the signal in negative direction.

## 18.7 Function Documentation

### 18.7.1 void ENC_Init ( ENC_Type * *base*, const enc_config_t * *config* )

This function is to make the initialization for the ENC module. It should be called firstly before any operation to the ENC with the operations like:

- Enable the clock for ENC module.
- Configure the ENC's working attributes.

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |
| *config* | Pointer to configuration structure. See to "enc_config_t". |

### 18.7.2 void ENC_Deinit ( ENC_Type * *base* )

This function is to make the de-initialization for the ENC module. It could be called when ENC is no longer used with the operations like:

- Disable the clock for ENC module.

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

### 18.7.3 void ENC_GetDefaultConfig ( enc_config_t * *config* )

This function initializes the ENC configuration structure with an available settings, the default value are:

```
*   config->enableReverseDirection               = false;
*   config->decoderWorkMode                      = kENC_DecoderWorkAsNormalMode
      ;
*   config->HOMETriggerMode                      = kENC_HOMETriggerDisabled;
*   config->INDEXTriggerMode                     = kENC_INDEXTriggerDisabled;
*   config->enableTRIGGERClearPositionCounter    = false;
*   config->enableTRIGGERClearHoldPositionCounter = false;
*   config->enableWatchdog                       = false;
*   config->watchdogTimeoutValue                 = 0U;
*   config->filterCount                          = 0U;
*   config->filterSamplePeriod                   = 0U;
*   config->positionMatchMode                    =
      kENC_POSMATCHOnPositionCounterEqualToComapreValue;
*   config->positionCompareValue                 = 0xFFFFFFFFU;
*   config->revolutionCountCondition             =
      kENC_RevolutionCountOnINDEXPulse;
*   config->enableModuloCountMode                = false;
*   config->positionModulusValue                 = 0U;
*   config->positionInitialValue                 = 0U;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to a variable of configuration structure. See to "enc_config_t". |

### 18.7.4 void ENC_DoSoftwareLoadInitialPositionValue ( ENC_Type * *base* )

This function is to transfer the initial position value (UINIT and LINIT) contents to position counter (UPOS and LPOS), so that to provide the consistent operation the position counter registers.

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

### 18.7.5 void ENC_SetSelfTestConfig ( ENC_Type * *base,* const enc_self_test_config_t * *config* )

This function is to enable and configuration the self test function. It controls and sets the frequency of a quadrature signal generator. It provides a quadrature test signal to the inputs of the quadrature decoder module. It is a factory test feature; however, it may be useful to customers' software development and testing.

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |
| *config* | Pointer to configuration structure. See to "enc_self_test_config_t". Pass "NULL" to disable. |

### 18.7.6 void ENC_EnableWatchdog ( ENC_Type * *base,* bool *enable* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address |
| *enable* | Enables or disables the watchdog |

### 18.7.7 void ENC_SetInitialPositionValue ( ENC_Type * *base,* uint32_t *value* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address |
| *value* | Positive initial value |

### 18.7.8 uint32_t ENC_GetStatusFlags ( ENC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

Mask value of status flags. For available mask, see to "_enc_status_flags".

### 18.7.9 void ENC_ClearStatusFlags ( ENC_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |
| *mask* | Mask value of status flags to be cleared. For available mask, see to "_enc_status_-flags". |

### 18.7.10 static uint16_t ENC_GetSignalStatusFlags ( ENC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

Mask value of signals' real-time status. For available mask, see to "_enc_signal_status_flags"

### 18.7.11 void ENC_EnableInterrupts ( ENC_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |
| *mask* | Mask value of interrupts to be enabled. For available mask, see to "_enc_interrupt_-enable". |

### 18.7.12 void ENC_DisableInterrupts ( ENC_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |
| *mask* | Mask value of interrupts to be disabled. For available mask, see to "_enc_interrupt_-enable". |

### 18.7.13 uint32_t ENC_GetEnabledInterrupts ( ENC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

Mask value of enabled interrupts.

### 18.7.14 uint32_t ENC_GetPositionValue ( ENC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

Current position counter's value.

### 18.7.15 uint32_t ENC_GetHoldPositionValue ( ENC_Type * *base* )

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

Hold position counter's value.

### 18.7.16 static uint16_t ENC_GetPositionDifferenceValue ( ENC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

The position difference counter's value.

### 18.7.17 static uint16_t ENC_GetHoldPositionDifferenceValue ( ENC_Type * *base* ) [inline], [static]

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

Hold position difference counter's value.

### 18.7.18 static uint16_t ENC_GetRevolutionValue ( ENC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

The position revolution counter's value.

### 18.7.19 static uint16_t ENC_GetHoldRevolutionValue ( ENC_Type * *base* ) [inline], [static]

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

| | |
|---|---|
| *base* | ENC peripheral base address. |

Returns

Hold position revolution counter's value.

## 18.8 Variable Documentation

### 18.8.1 bool enc_config_t::enableReverseDirection

### 18.8.2 enc_decoder_work_mode_t enc_config_t::decoderWorkMode

### 18.8.3 enc_home_trigger_mode_t enc_config_t::HOMETriggerMode

### 18.8.4 enc_index_trigger_mode_t enc_config_t::INDEXTriggerMode

### 18.8.5 bool enc_config_t::enableTRIGGERClearPositionCounter

### 18.8.6 bool enc_config_t::enableWatchdog

### 18.8.7 uint16_t enc_config_t::watchdogTimeoutValue

It stores the timeout count for the quadrature decoder module watchdog timer. This field is only available when "enableWatchdog" = true. The available value is a 16-bit unsigned number.

### 18.8.8 uint16_t enc_config_t::filterCount

This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. The Available range is 0 - 7.

### 18.8.9 uint16_t enc_config_t::filterSamplePeriod

This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available range is 0 - 255.

### 18.8.10 enc_position_match_mode_t enc_config_t::positionMatchMode

### 18.8.11 uint32_t enc_config_t::positionCompareValue

The available value is a 32-bit number.

### 18.8.12 enc_revolution_count_condition_t enc_config_t::revolutionCountCondition

### 18.8.13 bool enc_config_t::enableModuloCountMode

### 18.8.14 uint32_t enc_config_t::positionModulusValue

This value would be available only when "enableModuloCountMode" = true. The available value is a 32-bit number.

### 18.8.15 uint32_t enc_config_t::positionInitialValue

The available value is a 32-bit number.

### 18.8.16 enc_self_test_direction_t enc_self_test_config_t::signalDirection

### 18.8.17 uint16_t enc_self_test_config_t::signalCount

The available range is 0 - 255.

### 18.8.18 uint16_t enc_self_test_config_t::signalPeriod

The available range is 0 - 31.

# Chapter 19
# ENET: Ethernet MAC Driver

## 19.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 10/100 Mbps Ethernet MAC (ENET) module of MCUXpresso SDK devices.

The MII interface is the interface connected with MAC and PHY. the Serial management interface - MII management interface should be set before any access to the external PHY chip register. Call ENET_SetSMI() to initialize the MII management interface. Use ENET_StartSMIRead(), ENET_StartSMIWrite(), and ENET_ReadSMIData() to read/write to PHY registers. This function group sets up the MII and serial management SMI interface, gets data from the SMI interface, and starts the SMI read and write command. Use ENET_SetMII() to configure the MII before successfully getting data from the external PHY.

This group sets/gets the ENET mac address and the multicast group address filter. ENET_AddMulticastGroup() should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

This group has the receive active API ENET_ActiveRead() and ENET_ActiveReadMultiRing() for single and multiple rings. The ENET_AVBConfigure() is provided to configure the AVB features to support the AVB frames transmission. Note that due to the AVB frames transmission scheme being a credit-based TX scheme, it is only supported with the Enhanced buffer descriptors. Because of this, the AVB configuration should only be done with the Enhanced buffer descriptor. When the AVB feature is required, make sure the the "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" is defined before using this feature.

1. For single ring

For ENET receive, the ENET_GetRxFrameSize() function needs to be called to get the received data size. Then, call the ENET_ReadFrame() function to get the received data. If the received error occurs, call the ENET_GetRxErrBeforeReadFrame() function after ENET_GetRxFrameSize() and before ENET_ReadFrame() functions to get the detailed error information.

For ENET transmit, call the ENET_SendFrame() function to send the data out. The transmit data error information is only accessible for the IEEE 1588 enhanced buffer descriptor mode. When the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE is defined, the ENET_GetTxErrAfterSendFrame() can be used to get the detail transmit error information. The transmit error information can only be updated by uDMA after the data is transmitted. The ENET_GetTxErrAfterSendFrame() function is recommended to be called on the transmit interrupt handler.

1. For multiple-ring supported

The ENET driver now added a series transactional APIs with postfix "MultiRing" to support the extended multiple-ring for AVB feature. There are extended multiple-ring functions for receive side: ENET_GetRxErrBeforeReadFrameMultiRing(), ENET_GetRxFrameSizeMultiRing(), and ENET_ReadFrameMultiRing(). They are the similar to the single ring receive APIs and only add the "ringId" input param to identify the different ring index. For TX side add the ENET_SendFrameMultiRing, ENET_GetTxErr-

AfterSendFrameMultiRing(). They are the similar to the single ring transmit APIs and only add the "ringId" input param to identify the different ring index.

This function group configures the PTP IEEE 1588 feature, starts/stops/gets/sets/adjusts the PTP IEEE 1588 timer, gets the receive/transmit frame timestamp, and PTP IEEE 1588 timer channel feature setting.

The ENET_Ptp1588Configure() function needs to be called when the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE is defined and the IEEE 1588 feature is required. The ENET_GetRxFrameTime() and ENET_GetTxFrameTime() functions are called by the PTP stack to get the timestamp captured by the ENET driver.

## 19.2 Typical use case

### 19.2.1 ENET Initialization, receive, and transmit operations

For the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE undefined use case, use the legacy type buffer descriptor transmit/receive the frame as follows. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/enet For the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE defined use case, add the PTP IEEE 1588 configuration to enable the PTP IEEE 1588 feature. The initialization occurs as follows. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/enet

## Data Structures

- struct enet_rx_bd_struct_t
  *Defines the receive buffer descriptor structure for the little endian system. More...*
- struct enet_tx_bd_struct_t
  *Defines the enhanced transmit buffer descriptor structure for the little endian system. More...*
- struct enet_data_error_stats_t
  *Defines the ENET data error statistics structure. More...*
- struct enet_buffer_config_t
  *Defines the receive buffer descriptor configuration structure. More...*
- struct enet_ptp_time_t
  *Defines the ENET PTP time stamp structure. More...*
- struct enet_ptp_time_data_t
  *Defines the structure for the ENET PTP message data and timestamp data. More...*
- struct enet_ptp_time_data_ring_t
  *Defines the ENET PTP ring buffer structure for the PTP message timestamp store. More...*
- struct enet_ptp_config_t
  *Defines the ENET PTP configuration structure. More...*
- struct enet_intcoalesce_config_t
  *Defines the interrupt coalescing configure structure. More...*
- struct enet_config_t
  *Defines the basic configuration structure for the ENET device. More...*
- struct enet_handle_t
  *Defines the ENET handler structure. More...*

## Macros

- #define ENET_BUFFDESCRIPTOR_RX_ERR_MASK

*Defines the receive error status flag mask.*

## Typedefs

- typedef void(∗ enet_callback_t )(ENET_Type ∗base, enet_handle_t ∗handle, enet_event_t event, void ∗userData)
  *ENET callback function.*

## Enumerations

- enum _enet_status {
  kStatus_ENET_RxFrameError = MAKE_STATUS(kStatusGroup_ENET, 0U),
  kStatus_ENET_RxFrameFail = MAKE_STATUS(kStatusGroup_ENET, 1U),
  kStatus_ENET_RxFrameEmpty = MAKE_STATUS(kStatusGroup_ENET, 2U),
  kStatus_ENET_TxFrameOverLen = MAKE_STATUS(kStatusGroup_ENET, 3U),
  kStatus_ENET_TxFrameBusy = MAKE_STATUS(kStatusGroup_ENET, 4U),
  kStatus_ENET_TxFrameFail = MAKE_STATUS(kStatusGroup_ENET, 5U),
  kStatus_ENET_PtpTsRingFull = MAKE_STATUS(kStatusGroup_ENET, 6U),
  kStatus_ENET_PtpTsRingEmpty = MAKE_STATUS(kStatusGroup_ENET, 7U) }
  *Defines the status return codes for transaction.*
- enum enet_mii_mode_t {
  kENET_MiiMode = 0U,
  kENET_RmiiMode = 1U }
  *Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.*
- enum enet_mii_speed_t {
  kENET_MiiSpeed10M = 0U,
  kENET_MiiSpeed100M = 1U }
  *Defines the 10/100/1000 Mbps speed for the MII data interface.*
- enum enet_mii_duplex_t {
  kENET_MiiHalfDuplex = 0U,
  kENET_MiiFullDuplex }
  *Defines the half or full duplex for the MII data interface.*
- enum enet_mii_write_t {
  kENET_MiiWriteNoCompliant = 0U,
  kENET_MiiWriteValidFrame }
  *Define the MII opcode for normal MDIO_CLAUSES_22 Frame.*
- enum enet_mii_read_t {
  kENET_MiiReadValidFrame = 2U,
  kENET_MiiReadNoCompliant = 3U }
  *Defines the read operation for the MII management frame.*
- enum enet_mii_extend_opcode {
  kENET_MiiAddrWrite_C45 = 0U,
  kENET_MiiWriteFrame_C45 = 1U,
  kENET_MiiReadFrame_C45 = 3U }
  *Define the MII opcode for extended MDIO_CLAUSES_45 Frame.*
- enum enet_special_control_flag_t {

kENET_ControlFlowControlEnable = 0x0001U,
kENET_ControlRxPayloadCheckEnable = 0x0002U,
kENET_ControlRxPadRemoveEnable = 0x0004U,
kENET_ControlRxBroadCastRejectEnable = 0x0008U,
kENET_ControlMacAddrInsert = 0x0010U,
kENET_ControlStoreAndFwdDisable = 0x0020U,
kENET_ControlSMIPreambleDisable = 0x0040U,
kENET_ControlPromiscuousEnable = 0x0080U,
kENET_ControlMIILoopEnable = 0x0100U,
kENET_ControlVLANTagEnable = 0x0200U }

*Defines a special configuration for ENET MAC controller.*

- enum enet_interrupt_enable_t {
  kENET_BabrInterrupt = ENET_EIR_BABR_MASK,
  kENET_BabtInterrupt = ENET_EIR_BABT_MASK,
  kENET_GraceStopInterrupt = ENET_EIR_GRA_MASK,
  kENET_TxFrameInterrupt = ENET_EIR_TXF_MASK,
  kENET_TxBufferInterrupt = ENET_EIR_TXB_MASK,
  kENET_RxFrameInterrupt = ENET_EIR_RXF_MASK,
  kENET_RxBufferInterrupt = ENET_EIR_RXB_MASK,
  kENET_MiiInterrupt = ENET_EIR_MII_MASK,
  kENET_EBusERInterrupt = ENET_EIR_EBERR_MASK,
  kENET_LateCollisionInterrupt = ENET_EIR_LC_MASK,
  kENET_RetryLimitInterrupt = ENET_EIR_RL_MASK,
  kENET_UnderrunInterrupt = ENET_EIR_UN_MASK,
  kENET_PayloadRxInterrupt = ENET_EIR_PLR_MASK,
  kENET_WakeupInterrupt = ENET_EIR_WAKEUP_MASK,
  kENET_TsAvailInterrupt = ENET_EIR_TS_AVAIL_MASK,
  kENET_TsTimerInterrupt = ENET_EIR_TS_TIMER_MASK }

  *List of interrupts supported by the peripheral.*
- enum enet_event_t {
  kENET_RxEvent,
  kENET_TxEvent,
  kENET_ErrEvent,
  kENET_WakeUpEvent,
  kENET_TimeStampEvent,
  kENET_TimeStampAvailEvent }

  *Defines the common interrupt event for callback use.*
- enum enet_tx_accelerator_t {
  kENET_TxAccelIsShift16Enabled = ENET_TACC_SHIFT16_MASK,
  kENET_TxAccelIpCheckEnabled = ENET_TACC_IPCHK_MASK,
  kENET_TxAccelProtoCheckEnabled = ENET_TACC_PROCHK_MASK }

  *Defines the transmit accelerator configuration.*
- enum enet_rx_accelerator_t {

kENET_RxAccelPadRemoveEnabled = ENET_RACC_PADREM_MASK,
kENET_RxAccelIpCheckEnabled = ENET_RACC_IPDIS_MASK,
kENET_RxAccelProtoCheckEnabled = ENET_RACC_PRODIS_MASK,
kENET_RxAccelMacCheckEnabled = ENET_RACC_LINEDIS_MASK,
kENET_RxAccelisShift16Enabled = ENET_RACC_SHIFT16_MASK }
*Defines the receive accelerator configuration.*

- enum enet_ptp_event_type_t {
  kENET_PtpEventMsgType = 3U,
  kENET_PtpSrcPortIdLen = 10U,
  kENET_PtpEventPort = 319U,
  kENET_PtpGnrlPort = 320U }
  *Defines the ENET PTP message related constant.*
- enum enet_ptp_timer_channel_t {
  kENET_PtpTimerChannel1 = 0U,
  kENET_PtpTimerChannel2,
  kENET_PtpTimerChannel3,
  kENET_PtpTimerChannel4 }
  *Defines the IEEE 1588 PTP timer channel numbers.*
- enum enet_ptp_timer_channel_mode_t {
  kENET_PtpChannelDisable = 0U,
  kENET_PtpChannelRisingCapture = 1U,
  kENET_PtpChannelFallingCapture = 2U,
  kENET_PtpChannelBothCapture = 3U,
  kENET_PtpChannelSoftCompare = 4U,
  kENET_PtpChannelToggleCompare = 5U,
  kENET_PtpChannelClearCompare = 6U,
  kENET_PtpChannelSetCompare = 7U,
  kENET_PtpChannelClearCompareSetOverflow = 10U,
  kENET_PtpChannelSetCompareClearOverflow = 11U,
  kENET_PtpChannelPulseLowonCompare = 14U,
  kENET_PtpChannelPulseHighonCompare = 15U }
  *Defines the capture or compare mode for IEEE 1588 PTP timer channels.*

## Driver version

- #define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))
  *Defines the driver version.*

## Control and status region bit masks of the receive buffer descriptor.

Defines the queue number.

- #define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U
  *Empty bit mask.*
- #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U
  *Software owner one mask.*
- #define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U
  *Next buffer descriptor is the start address.*

**Typical use case**

- #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U
  *Software owner two mask.*
- #define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U
  *Last BD of the frame mask.*
- #define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U
  *Received because of the promiscuous mode.*
- #define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U
  *Broadcast packet mask.*
- #define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U
  *Multicast packet mask.*
- #define ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK 0x0020U
  *Length violation mask.*
- #define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U
  *Non-octet aligned frame mask.*
- #define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U
  *CRC error mask.*
- #define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U
  *FIFO overrun mask.*
- #define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U
  *Frame is truncated mask.*

## Control and status bit masks of the transmit buffer descriptor.

- #define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U
  *Ready bit mask.*
- #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK 0x4000U
  *Software owner one mask.*
- #define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U
  *Wrap buffer descriptor mask.*
- #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK 0x1000U
  *Software owner two mask.*
- #define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U
  *Last BD of the frame mask.*
- #define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U
  *Transmit CRC mask.*

## First extended control region bit masks of the receive buffer descriptor.

- #define ENET_BUFFDESCRIPTOR_RX_IPV4_MASK 0x0001U
  *Ipv4 frame mask.*
- #define ENET_BUFFDESCRIPTOR_RX_IPV6_MASK 0x0002U
  *Ipv6 frame mask.*
- #define ENET_BUFFDESCRIPTOR_RX_VLAN_MASK 0x0004U
  *VLAN frame mask.*
- #define ENET_BUFFDESCRIPTOR_RX_PROTOCOLCHECKSUM_MASK 0x0010U
  *Protocol checksum error mask.*
- #define ENET_BUFFDESCRIPTOR_RX_IPHEADCHECKSUM_MASK 0x0020U
  *IP header checksum error mask.*

## Second extended control region bit masks of the receive buffer descriptor.

- #define ENET_BUFFDESCRIPTOR_RX_INTERRUPT_MASK 0x0080U
  *BD interrupt mask.*
- #define ENET_BUFFDESCRIPTOR_RX_UNICAST_MASK 0x0100U
  *Unicast frame mask.*
- #define ENET_BUFFDESCRIPTOR_RX_COLLISION_MASK 0x0200U
  *BD collision mask.*
- #define ENET_BUFFDESCRIPTOR_RX_PHYERR_MASK 0x0400U
  *PHY error mask.*
- #define ENET_BUFFDESCRIPTOR_RX_MACERR_MASK 0x8000U
  *Mac error mask.*

## First extended control region bit masks of the transmit buffer descriptor.

- #define ENET_BUFFDESCRIPTOR_TX_ERR_MASK 0x8000U
  *Transmit error mask.*
- #define ENET_BUFFDESCRIPTOR_TX_UNDERFLOWERR_MASK 0x2000U
  *Underflow error mask.*
- #define ENET_BUFFDESCRIPTOR_TX_EXCCOLLISIONERR_MASK 0x1000U
  *Excess collision error mask.*
- #define ENET_BUFFDESCRIPTOR_TX_FRAMEERR_MASK 0x0800U
  *Frame error mask.*
- #define ENET_BUFFDESCRIPTOR_TX_LATECOLLISIONERR_MASK 0x0400U
  *Late collision error mask.*
- #define ENET_BUFFDESCRIPTOR_TX_OVERFLOWERR_MASK 0x0200U
  *Overflow error mask.*
- #define ENET_BUFFDESCRIPTOR_TX_TIMESTAMPERR_MASK 0x0100U
  *Timestamp error mask.*

## Second extended control region bit masks of the transmit buffer descriptor.

- #define ENET_BUFFDESCRIPTOR_TX_INTERRUPT_MASK 0x4000U
  *Interrupt mask.*
- #define ENET_BUFFDESCRIPTOR_TX_TIMESTAMP_MASK 0x2000U
  *Timestamp flag mask.*

## Defines some Ethernet parameters.

- #define ENET_FRAME_MAX_FRAMELEN 1518U
  *Default maximum Ethernet frame size.*
- #define ENET_FIFO_MIN_RX_FULL 5U
  *ENET minimum receive FIFO full.*
- #define ENET_RX_MIN_BUFFERSIZE 256U
  *ENET minimum buffer size.*
- #define **ENET_PHY_MAXADDRESS** (ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT)
- #define **ENET_TX_INTERRUPT** (kENET_TxFrameInterrupt | kENET_TxBufferInterrupt)
- #define **ENET_RX_INTERRUPT** (kENET_RxFrameInterrupt | kENET_RxBufferInterrupt)
- #define **ENET_TS_INTERRUPT** (kENET_TsTimerInterrupt | kENET_TsAvailInterrupt)
- #define **ENET_ERR_INTERRUPT**
- #define **ENET_ERR_INTERRUPT**

**Typical use case**

## Initialization and De-initialization

- void ENET_GetDefaultConfig (enet_config_t *config)
  *Gets the ENET default configuration structure.*
- void ENET_Init (ENET_Type *base, enet_handle_t *handle, const enet_config_t *config, const enet_buffer_config_t *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)
  *Initializes the ENET module.*
- void ENET_Deinit (ENET_Type *base)
  *Deinitializes the ENET module.*
- static void ENET_Reset (ENET_Type *base)
  *Resets the ENET module.*

## MII interface operation

- void ENET_SetMII (ENET_Type *base, enet_mii_speed_t speed, enet_mii_duplex_t duplex)
  *Sets the ENET MII speed and duplex.*
- void ENET_SetSMI (ENET_Type *base, uint32_t srcClock_Hz, bool isPreambleDisabled)
  *Sets the ENET SMI(serial management interface)- MII management interface.*
- static bool ENET_GetSMI (ENET_Type *base)
  *Gets the ENET SMI- MII management interface configuration.*
- static uint32_t ENET_ReadSMIData (ENET_Type *base)
  *Reads data from the PHY register through an SMI interface.*
- void ENET_StartSMIRead (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, enet_mii_read_t operation)
  *Starts an SMI (Serial Management Interface) read command.*
- void ENET_StartSMIWrite (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, enet_mii_write_t operation, uint32_t data)
  *Starts an SMI write command.*
- void ENET_StartExtC45SMIRead (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg)
  *Starts the extended IEEE802.3 Clause 45 MDIO format SMI read command.*
- void ENET_StartExtC45SMIWrite (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, uint32_t data)
  *Starts the extended IEEE802.3 Clause 45 MDIO format SMI write command.*

## MAC Address Filter

- void ENET_SetMacAddr (ENET_Type *base, uint8_t *macAddr)
  *Sets the ENET module Mac address.*
- void ENET_GetMacAddr (ENET_Type *base, uint8_t *macAddr)
  *Gets the ENET module Mac address.*
- void ENET_AddMulticastGroup (ENET_Type *base, uint8_t *address)
  *Adds the ENET device to a multicast group.*
- void ENET_LeaveMulticastGroup (ENET_Type *base, uint8_t *address)
  *Moves the ENET device from a multicast group.*

## Other basic operation

- static void ENET_ActiveRead (ENET_Type *base)
  *Activates ENET read or receive.*
- static void ENET_EnableSleepMode (ENET_Type *base, bool enable)
  *Enables/disables the MAC to enter sleep mode.*

- static void ENET_GetAccelFunction (ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)
  *Gets ENET transmit and receive accelerator functions from MAC controller.*

## Interrupts.

- static void ENET_EnableInterrupts (ENET_Type *base, uint32_t mask)
  *Enables the ENET interrupt.*
- static void ENET_DisableInterrupts (ENET_Type *base, uint32_t mask)
  *Disables the ENET interrupt.*
- static uint32_t ENET_GetInterruptStatus (ENET_Type *base)
  *Gets the ENET interrupt status flag.*
- static void ENET_ClearInterruptStatus (ENET_Type *base, uint32_t mask)
  *Clears the ENET interrupt events status flag.*

## Transactional operation

- void ENET_SetCallback (enet_handle_t *handle, enet_callback_t callback, void *userData)
  *Sets the callback function.*
- void ENET_GetRxErrBeforeReadFrame (enet_handle_t *handle, enet_data_error_stats_t *eErrorStatic)
  *Gets the error statistics of a received frame for ENET single ring.*
- status_t ENET_GetTxErrAfterSendFrame (enet_handle_t *handle, enet_data_error_stats_t *eErrorStatic)
  *Gets the ENET transmit frame statistics after the data send for single ring.*
- status_t ENET_GetRxFrameSize (enet_handle_t *handle, uint32_t *length)
  *Gets the size of the read frame for single ring.*
- status_t ENET_ReadFrame (ENET_Type *base, enet_handle_t *handle, uint8_t *data, uint32_t length)
  *Reads a frame from the ENET device for single ring.*
- status_t ENET_SendFrame (ENET_Type *base, enet_handle_t *handle, const uint8_t *data, uint32_t length)
  *Transmits an ENET frame for single ring.*
- void ENET_TransmitIRQHandler (ENET_Type *base, enet_handle_t *handle)
  *The transmit IRQ handler.*
- void ENET_ReceiveIRQHandler (ENET_Type *base, enet_handle_t *handle)
  *The receive IRQ handler.*
- void ENET_ErrorIRQHandler (ENET_Type *base, enet_handle_t *handle)
  *Some special IRQ handler including the error, mii, wakeup irq handler.*
- void ENET_CommonFrame0IRQHandler (ENET_Type *base)
  *the common IRQ handler for the tx/rx/error etc irq handler.*

## ENET PTP 1588 function operation

- void ENET_Ptp1588Configure (ENET_Type *base, enet_handle_t *handle, enet_ptp_config_t *ptpConfig)
  *Configures the ENET PTP IEEE 1588 feature with the basic configuration.*
- void ENET_Ptp1588StartTimer (ENET_Type *base, uint32_t ptpClkSrc)
  *Starts the ENET PTP 1588 Timer.*
- static void ENET_Ptp1588StopTimer (ENET_Type *base)

*Stops the ENET PTP 1588 Timer.*
- void ENET_Ptp1588AdjustTimer (ENET_Type ∗base, uint32_t corrIncrease, uint32_t corrPeriod)
  *Adjusts the ENET PTP 1588 timer.*
- static void ENET_Ptp1588SetChannelMode (ENET_Type ∗base, enet_ptp_timer_channel_t channel, enet_ptp_timer_channel_mode_t mode, bool intEnable)
  *Sets the ENET PTP 1588 timer channel mode.*
- static void ENET_Ptp1588SetChannelOutputPulseWidth (ENET_Type ∗base, enet_ptp_timer_channel_t channel, bool isOutputLow, uint8_t pulseWidth, bool intEnable)
  *Sets ENET PTP 1588 timer channel mode pulse width.*
- static void ENET_Ptp1588SetChannelCmpValue (ENET_Type ∗base, enet_ptp_timer_channel_t channel, uint32_t cmpValue)
  *Sets the ENET PTP 1588 timer channel comparison value.*
- static bool ENET_Ptp1588GetChannelStatus (ENET_Type ∗base, enet_ptp_timer_channel_t channel)
  *Gets the ENET PTP 1588 timer channel status.*
- static void ENET_Ptp1588ClearChannelStatus (ENET_Type ∗base, enet_ptp_timer_channel_t channel)
  *Clears the ENET PTP 1588 timer channel status.*
- void ENET_Ptp1588GetTimer (ENET_Type ∗base, enet_handle_t ∗handle, enet_ptp_time_t ∗ptpTime)
  *Gets the current ENET time from the PTP 1588 timer.*
- void ENET_Ptp1588SetTimer (ENET_Type ∗base, enet_handle_t ∗handle, enet_ptp_time_t ∗ptpTime)
  *Sets the ENET PTP 1588 timer to the assigned time.*
- void ENET_Ptp1588TimerIRQHandler (ENET_Type ∗base, enet_handle_t ∗handle)
  *The IEEE 1588 PTP time stamp interrupt handler.*
- status_t ENET_GetRxFrameTime (enet_handle_t ∗handle, enet_ptp_time_data_t ∗ptpTimeData)
  *Gets the time stamp of the received frame.*
- status_t ENET_GetTxFrameTime (enet_handle_t ∗handle, enet_ptp_time_data_t ∗ptpTimeData)
  *Gets the time stamp of the transmit frame.*

## 19.3 Data Structure Documentation

### 19.3.1 struct enet_rx_bd_struct_t

### Data Fields

- uint16_t length
  *Buffer descriptor data length.*
- uint16_t control
  *Buffer descriptor control and status.*
- uint8_t ∗ buffer
  *Data buffer pointer.*
- uint16_t controlExtend0
  *Extend buffer descriptor control0.*
- uint16_t controlExtend1
  *Extend buffer descriptor control1.*
- uint16_t payloadCheckSum
  *Internal payload checksum.*

- uint8_t headerLength
  *Header length.*
- uint8_t protocolTyte
  *Protocol type.*
- uint16_t controlExtend2
  *Extend buffer descriptor control2.*
- uint32_t timestamp
  *Timestamp.*

#### 19.3.1.0.0.17 Field Documentation

##### 19.3.1.0.0.17.1 uint16_t enet_rx_bd_struct_t::length

##### 19.3.1.0.0.17.2 uint16_t enet_rx_bd_struct_t::control

##### 19.3.1.0.0.17.3 uint8_t* enet_rx_bd_struct_t::buffer

##### 19.3.1.0.0.17.4 uint16_t enet_rx_bd_struct_t::controlExtend0

##### 19.3.1.0.0.17.5 uint16_t enet_rx_bd_struct_t::controlExtend1

##### 19.3.1.0.0.17.6 uint16_t enet_rx_bd_struct_t::payloadCheckSum

##### 19.3.1.0.0.17.7 uint8_t enet_rx_bd_struct_t::headerLength

##### 19.3.1.0.0.17.8 uint8_t enet_rx_bd_struct_t::protocolTyte

##### 19.3.1.0.0.17.9 uint16_t enet_rx_bd_struct_t::controlExtend2

##### 19.3.1.0.0.17.10 uint32_t enet_rx_bd_struct_t::timestamp

### 19.3.2 struct enet_tx_bd_struct_t

## Data Fields

- uint16_t length
  *Buffer descriptor data length.*
- uint16_t control
  *Buffer descriptor control and status.*
- uint8_t * buffer
  *Data buffer pointer.*
- uint16_t controlExtend0
  *Extend buffer descriptor control0.*
- uint16_t controlExtend1
  *Extend buffer descriptor control1.*
- uint16_t controlExtend2
  *Extend buffer descriptor control2.*
- uint32_t timestamp
  *Timestamp.*

#### 19.3.2.0.0.18 Field Documentation

#### 19.3.2.0.0.18.1 uint16_t enet_tx_bd_struct_t::length

#### 19.3.2.0.0.18.2 uint16_t enet_tx_bd_struct_t::control

#### 19.3.2.0.0.18.3 uint8_t* enet_tx_bd_struct_t::buffer

#### 19.3.2.0.0.18.4 uint16_t enet_tx_bd_struct_t::controlExtend0

#### 19.3.2.0.0.18.5 uint16_t enet_tx_bd_struct_t::controlExtend1

#### 19.3.2.0.0.18.6 uint16_t enet_tx_bd_struct_t::controlExtend2

#### 19.3.2.0.0.18.7 uint32_t enet_tx_bd_struct_t::timestamp

### 19.3.3 struct enet_data_error_stats_t

## Data Fields

- uint32_t statsRxLenGreaterErr
  *Receive length greater than RCR[MAX_FL].*
- uint32_t statsRxAlignErr
  *Receive non-octet alignment/.*
- uint32_t statsRxFcsErr
  *Receive CRC error.*
- uint32_t statsRxOverRunErr
  *Receive over run.*
- uint32_t statsRxTruncateErr
  *Receive truncate.*
- uint32_t statsRxProtocolChecksumErr
  *Receive protocol checksum error.*
- uint32_t statsRxIpHeadChecksumErr
  *Receive IP header checksum error.*
- uint32_t statsRxMacErr
  *Receive Mac error.*
- uint32_t statsRxPhyErr
  *Receive PHY error.*
- uint32_t statsRxCollisionErr
  *Receive collision.*
- uint32_t statsTxErr
  *The error happen when transmit the frame.*
- uint32_t statsTxFrameErr
  *The transmit frame is error.*
- uint32_t statsTxOverFlowErr
  *Transmit overflow.*
- uint32_t statsTxLateCollisionErr
  *Transmit late collision.*
- uint32_t statsTxExcessCollisionErr
  *Transmit excess collision.*

- uint32_t statsTxUnderFlowErr
  *Transmit under flow error.*
- uint32_t statsTxTsErr
  *Transmit time stamp error.*

#### 19.3.3.0.0.19 Field Documentation

##### 19.3.3.0.0.19.1 uint32_t enet_data_error_stats_t::statsRxLenGreaterErr

##### 19.3.3.0.0.19.2 uint32_t enet_data_error_stats_t::statsRxFcsErr

##### 19.3.3.0.0.19.3 uint32_t enet_data_error_stats_t::statsRxOverRunErr

##### 19.3.3.0.0.19.4 uint32_t enet_data_error_stats_t::statsRxTruncateErr

##### 19.3.3.0.0.19.5 uint32_t enet_data_error_stats_t::statsRxProtocolChecksumErr

##### 19.3.3.0.0.19.6 uint32_t enet_data_error_stats_t::statsRxIpHeadChecksumErr

##### 19.3.3.0.0.19.7 uint32_t enet_data_error_stats_t::statsRxMacErr

##### 19.3.3.0.0.19.8 uint32_t enet_data_error_stats_t::statsRxPhyErr

##### 19.3.3.0.0.19.9 uint32_t enet_data_error_stats_t::statsRxCollisionErr

##### 19.3.3.0.0.19.10 uint32_t enet_data_error_stats_t::statsTxErr

##### 19.3.3.0.0.19.11 uint32_t enet_data_error_stats_t::statsTxFrameErr

##### 19.3.3.0.0.19.12 uint32_t enet_data_error_stats_t::statsTxOverFlowErr

##### 19.3.3.0.0.19.13 uint32_t enet_data_error_stats_t::statsTxLateCollisionErr

##### 19.3.3.0.0.19.14 uint32_t enet_data_error_stats_t::statsTxExcessCollisionErr

##### 19.3.3.0.0.19.15 uint32_t enet_data_error_stats_t::statsTxUnderFlowErr

##### 19.3.3.0.0.19.16 uint32_t enet_data_error_stats_t::statsTxTsErr

### 19.3.4 struct enet_buffer_config_t

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

1. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.
2. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
3. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUF-

F_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber ∗ rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber ∗ txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

## Data Fields

- uint16_t rxBdNumber
  *Receive buffer descriptor number.*
- uint16_t txBdNumber
  *Transmit buffer descriptor number.*
- uint32_t rxBuffSizeAlign
  *Aligned receive data buffer size.*
- uint32_t txBuffSizeAlign
  *Aligned transmit data buffer size.*
- volatile enet_rx_bd_struct_t ∗ rxBdStartAddrAlign
  *Aligned receive buffer descriptor start address: should be non-cacheable.*
- volatile enet_tx_bd_struct_t ∗ txBdStartAddrAlign
  *Aligned transmit buffer descriptor start address: should be non-cacheable.*
- uint8_t ∗ rxBufferAlign
  *Receive data buffer start address.*
- uint8_t ∗ txBufferAlign
  *Transmit data buffer start address.*

#### 19.3.4.0.0.20 Field Documentation

#### 19.3.4.0.0.20.1 uint16_t enet_buffer_config_t::rxBdNumber

#### 19.3.4.0.0.20.2 uint16_t enet_buffer_config_t::txBdNumber

#### 19.3.4.0.0.20.3 uint32_t enet_buffer_config_t::rxBuffSizeAlign

#### 19.3.4.0.0.20.4 uint32_t enet_buffer_config_t::txBuffSizeAlign

#### 19.3.4.0.0.20.5 volatile enet_rx_bd_struct_t∗ enet_buffer_config_t::rxBdStartAddrAlign

#### 19.3.4.0.0.20.6 volatile enet_tx_bd_struct_t∗ enet_buffer_config_t::txBdStartAddrAlign

#### 19.3.4.0.0.20.7 uint8_t∗ enet_buffer_config_t::rxBufferAlign

#### 19.3.4.0.0.20.8 uint8_t∗ enet_buffer_config_t::txBufferAlign

### 19.3.5 struct enet_ptp_time_t

## Data Fields

- uint64_t second

*Second.*
- uint32_t nanosecond
  *Nanosecond.*

#### 19.3.5.0.0.21 Field Documentation

#### 19.3.5.0.0.21.1 uint64_t enet_ptp_time_t::second

#### 19.3.5.0.0.21.2 uint32_t enet_ptp_time_t::nanosecond

### 19.3.6 struct enet_ptp_time_data_t

## Data Fields

- uint8_t version
  *PTP version.*
- uint8_t sourcePortId [kENET_PtpSrcPortIdLen]
  *PTP source port ID.*
- uint16_t sequenceId
  *PTP sequence ID.*
- uint8_t messageType
  *PTP message type.*
- enet_ptp_time_t timeStamp
  *PTP timestamp.*

#### 19.3.6.0.0.22 Field Documentation

#### 19.3.6.0.0.22.1 uint8_t enet_ptp_time_data_t::version

#### 19.3.6.0.0.22.2 uint8_t enet_ptp_time_data_t::sourcePortId[kENET_PtpSrcPortIdLen]

#### 19.3.6.0.0.22.3 uint16_t enet_ptp_time_data_t::sequenceId

#### 19.3.6.0.0.22.4 uint8_t enet_ptp_time_data_t::messageType

#### 19.3.6.0.0.22.5 enet_ptp_time_t enet_ptp_time_data_t::timeStamp

### 19.3.7 struct enet_ptp_time_data_ring_t

## Data Fields

- uint32_t front
  *The first index of the ring.*
- uint32_t end
  *The end index of the ring.*
- uint32_t size
  *The size of the ring.*
- enet_ptp_time_data_t * ptpTsData
  *PTP message data structure.*

#### 19.3.7.0.0.23 Field Documentation

#### 19.3.7.0.0.23.1 uint32_t enet_ptp_time_data_ring_t::front

#### 19.3.7.0.0.23.2 uint32_t enet_ptp_time_data_ring_t::end

#### 19.3.7.0.0.23.3 uint32_t enet_ptp_time_data_ring_t::size

#### 19.3.7.0.0.23.4 enet_ptp_time_data_t* enet_ptp_time_data_ring_t::ptpTsData

### 19.3.8 struct enet_ptp_config_t

## Data Fields

- uint8_t ptpTsRxBuffNum
  *Receive 1588 timestamp buffer number.*
- uint8_t ptpTsTxBuffNum
  *Transmit 1588 timestamp buffer number.*
- enet_ptp_time_data_t * rxPtpTsData
  *The start address of 1588 receive timestamp buffers.*
- enet_ptp_time_data_t * txPtpTsData
  *The start address of 1588 transmit timestamp buffers.*
- enet_ptp_timer_channel_t channel
  *Used for ERRATA_2579: the PTP 1588 timer channel for time interrupt.*
- uint32_t ptp1588ClockSrc_Hz
  *The clock source of the PTP 1588 timer.*

#### 19.3.8.0.0.24 Field Documentation

#### 19.3.8.0.0.24.1 enet_ptp_timer_channel_t enet_ptp_config_t::channel

#### 19.3.8.0.0.24.2 uint32_t enet_ptp_config_t::ptp1588ClockSrc_Hz

### 19.3.9 struct enet_intcoalesce_config_t

## Data Fields

- uint8_t txCoalesceFrameCount [FSL_FEATURE_ENET_QUEUE]
  *Transmit interrupt coalescing frame count threshold.*
- uint16_t txCoalesceTimeCount [FSL_FEATURE_ENET_QUEUE]
  *Transmit interrupt coalescing timer count threshold.*
- uint8_t rxCoalesceFrameCount [FSL_FEATURE_ENET_QUEUE]
  *Receive interrupt coalescing frame count threshold.*
- uint16_t rxCoalesceTimeCount [FSL_FEATURE_ENET_QUEUE]
  *Receive interrupt coalescing timer count threshold.*

#### 19.3.9.0.0.25 Field Documentation

##### 19.3.9.0.0.25.1 uint8_t enet_intcoalesce_config_t::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]

##### 19.3.9.0.0.25.2 uint16_t enet_intcoalesce_config_t::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]

##### 19.3.9.0.0.25.3 uint8_t enet_intcoalesce_config_t::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]

##### 19.3.9.0.0.25.4 uint16_t enet_intcoalesce_config_t::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]

### 19.3.10 struct enet_config_t

Note:

1. macSpecialConfig is used for a special control configuration, A logical OR of "enet_special_control_flag_t". For a special configuration for MAC, set this parameter to 0.
2. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
3. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.
4. When "kENET_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
5. When "kENET_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
6. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.
7. The intCoalesceCfg can be used in the rx or tx enabled cases to decrese the CPU loading.

### Data Fields

- uint32_t macSpecialConfig
  *Mac special configuration.*
- uint32_t interrupt
  *Mac interrupt source.*

- uint16_t rxMaxFrameLen
  *Receive maximum frame length.*
- enet_mii_mode_t miiMode
  *MII mode.*
- enet_mii_speed_t miiSpeed
  *MII Speed.*
- enet_mii_duplex_t miiDuplex
  *MII duplex.*
- uint8_t rxAccelerConfig
  *Receive accelerator, A logical OR of "enet_rx_accelerator_t".*
- uint8_t txAccelerConfig
  *Transmit accelerator, A logical OR of "enet_rx_accelerator_t".*
- uint16_t pauseDuration
  *For flow control enabled case: Pause duration.*
- uint8_t rxFifoEmptyThreshold
  *For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.*
- uint8_t rxFifoFullThreshold
  `For store and forward disable case, the data required in RX FIFO to notify` *the MAC receive ready status.*
- uint8_t txFifoWatermark
  `For store and forward disable case, the data required in TX FIFO` *before a frame transmit start.*
- uint8_t ringNum
  *Number of used rings.*

#### 19.3.10.0.0.26 Field Documentation

##### 19.3.10.0.0.26.1 uint32_t enet_config_t::macSpecialConfig

A logical OR of "enet_special_control_flag_t".

##### 19.3.10.0.0.26.2 uint32_t enet_config_t::interrupt

A logical OR of "enet_interrupt_enable_t".

#### 19.3.10.0.0.26.3 uint16_t enet_config_t::rxMaxFrameLen

#### 19.3.10.0.0.26.4 enet_mii_mode_t enet_config_t::miiMode

#### 19.3.10.0.0.26.5 enet_mii_speed_t enet_config_t::miiSpeed

#### 19.3.10.0.0.26.6 enet_mii_duplex_t enet_config_t::miiDuplex

#### 19.3.10.0.0.26.7 uint8_t enet_config_t::rxAccelerConfig

#### 19.3.10.0.0.26.8 uint8_t enet_config_t::txAccelerConfig

#### 19.3.10.0.0.26.9 uint16_t enet_config_t::pauseDuration

#### 19.3.10.0.0.26.10 uint8_t enet_config_t::rxFifoEmptyThreshold

#### 19.3.10.0.0.26.11 uint8_t enet_config_t::rxFifoFullThreshold

#### 19.3.10.0.0.26.12 uint8_t enet_config_t::txFifoWatermark

#### 19.3.10.0.0.26.13 uint8_t enet_config_t::ringNum

default with 1 – single ring.

### 19.3.11 struct _enet_handle

## Data Fields

- volatile enet_rx_bd_struct_t * rxBdBase [FSL_FEATURE_ENET_QUEUE]
  *Receive buffer descriptor base address pointer.*
- volatile enet_rx_bd_struct_t * rxBdCurrent [FSL_FEATURE_ENET_QUEUE]
  *The current available receive buffer descriptor pointer.*
- volatile enet_tx_bd_struct_t * txBdBase [FSL_FEATURE_ENET_QUEUE]
  *Transmit buffer descriptor base address pointer.*
- volatile enet_tx_bd_struct_t * txBdCurrent [FSL_FEATURE_ENET_QUEUE]
  *The current available transmit buffer descriptor pointer.*
- uint32_t rxBuffSizeAlign [FSL_FEATURE_ENET_QUEUE]
  *Receive buffer size alignment.*
- uint32_t txBuffSizeAlign [FSL_FEATURE_ENET_QUEUE]
  *Transmit buffer size alignment.*
- uint8_t ringNum
  *Number of used rings.*
- enet_callback_t callback
  *Callback function.*
- void * userData
  *Callback function parameter.*
- volatile enet_tx_bd_struct_t * txBdDirtyStatic [FSL_FEATURE_ENET_QUEUE]
  *The dirty transmit buffer descriptor for error static update.*
- volatile enet_tx_bd_struct_t * txBdDirtyTime [FSL_FEATURE_ENET_QUEUE]

*The dirty transmit buffer descriptor for time stamp update.*

- uint64_t msTimerSecond
  *The second for Master PTP timer .*
- enet_ptp_time_data_ring_t rxPtpTsDataRing
  *Receive PTP 1588 time stamp data ring buffer.*
- enet_ptp_time_data_ring_t txPtpTsDataRing
  *Transmit PTP 1588 time stamp data ring buffer.*

#### 19.3.11.0.0.27 Field Documentation

##### 19.3.11.0.0.27.1 volatile enet_rx_bd_struct_t* enet_handle_t::rxBdBase[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.2 volatile enet_rx_bd_struct_t* enet_handle_t::rxBdCurrent[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.3 volatile enet_tx_bd_struct_t* enet_handle_t::txBdBase[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.4 volatile enet_tx_bd_struct_t* enet_handle_t::txBdCurrent[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.5 uint32_t enet_handle_t::rxBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.6 uint32_t enet_handle_t::txBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.7 uint8_t enet_handle_t::ringNum

##### 19.3.11.0.0.27.8 enet_callback_t enet_handle_t::callback

##### 19.3.11.0.0.27.9 void* enet_handle_t::userData

##### 19.3.11.0.0.27.10 volatile enet_tx_bd_struct_t* enet_handle_t::txBdDirtyStatic[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.11 volatile enet_tx_bd_struct_t* enet_handle_t::txBdDirtyTime[FSL_FEATURE_ENET_QUEUE]

##### 19.3.11.0.0.27.12 uint64_t enet_handle_t::msTimerSecond

##### 19.3.11.0.0.27.13 enet_ptp_time_data_ring_t enet_handle_t::rxPtpTsDataRing

##### 19.3.11.0.0.27.14 enet_ptp_time_data_ring_t enet_handle_t::txPtpTsDataRing

## 19.4 Macro Definition Documentation

### 19.4.1 #define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))

Version 2.2.4.

### 19.4.2 #define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U

### 19.4.3 #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U

### 19.4.4 #define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U

### 19.4.5 #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U

### 19.4.6 #define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U

### 19.4.7 #define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U

### 19.4.8 #define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U

### 19.4.9 #define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U

### 19.4.10 #define ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK 0x0020U

### 19.4.11 #define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U

### 19.4.12 #define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U

### 19.4.13 #define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U

### 19.4.14 #define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U

### 19.4.15 #define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U

### 19.4.16 #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK 0x4000U

### 19.4.17 #define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U

### 19.4.18 #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK 0x1000U

### 19.4.19 #define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U

### 19.4.20 #define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U

### 19.4.21 #define ENET_BUFFDESCRIPTOR_RX_IPV4_MASK 0x0001U

### 19.4.22 #define ENET_BUFFDESCRIPTOR_RX_IPV6_MASK 0x0002U

```
(ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK |
      ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK | \
     ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK |
      ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK |
      ENET_BUFFDESCRIPTOR_RX_CRC_MASK)
```

#### 19.4.41 #define ENET_FRAME_MAX_FRAMELEN 1518U

#### 19.4.42 #define ENET_FIFO_MIN_RX_FULL 5U

#### 19.4.43 #define ENET_RX_MIN_BUFFERSIZE 256U

### 19.5 Typedef Documentation

#### 19.5.1 typedef void(∗ enet_callback_t)(ENET_Type ∗base, enet_handle_t ∗handle, enet_event_t event, void ∗userData)

### 19.6 Enumeration Type Documentation

#### 19.6.1 enum _enet_status

Enumerator

***kStatus_ENET_RxFrameError*** A frame received but data error happen.
***kStatus_ENET_RxFrameFail*** Failed to receive a frame.
***kStatus_ENET_RxFrameEmpty*** No frame arrive.
***kStatus_ENET_TxFrameOverLen*** Tx frame over length.
***kStatus_ENET_TxFrameBusy*** Tx buffer descriptors are under process.
***kStatus_ENET_TxFrameFail*** Transmit frame fail.
***kStatus_ENET_PtpTsRingFull*** Timestamp ring full.
***kStatus_ENET_PtpTsRingEmpty*** Timestamp ring empty.

#### 19.6.2 enum enet_mii_mode_t

Enumerator

***kENET_MiiMode*** MII mode for data interface.
***kENET_RmiiMode*** RMII mode for data interface.

#### 19.6.3 enum enet_mii_speed_t

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

Enumerator

***kENET_MiiSpeed10M*** Speed 10 Mbps.
***kENET_MiiSpeed100M*** Speed 100 Mbps.

### 19.6.4 enum enet_mii_duplex_t

Enumerator

***kENET_MiiHalfDuplex*** Half duplex mode.
***kENET_MiiFullDuplex*** Full duplex mode.

### 19.6.5 enum enet_mii_write_t

Enumerator

***kENET_MiiWriteNoCompliant*** Write frame operation, but not MII-compliant.
***kENET_MiiWriteValidFrame*** Write frame operation for a valid MII management frame.

### 19.6.6 enum enet_mii_read_t

Enumerator

***kENET_MiiReadValidFrame*** Read frame operation for a valid MII management frame.
***kENET_MiiReadNoCompliant*** Read frame operation, but not MII-compliant.

### 19.6.7 enum enet_mii_extend_opcode

Enumerator

***kENET_MiiAddrWrite_C45*** Address Write operation.
***kENET_MiiWriteFrame_C45*** Write frame operation for a valid MII management frame.
***kENET_MiiReadFrame_C45*** Read frame operation for a valid MII management frame.

### 19.6.8 enum enet_special_control_flag_t

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the enet_config_t.

Enumerator

***kENET_ControlFlowControlEnable*** Enable ENET flow control: pause frame.
***kENET_ControlRxPayloadCheckEnable*** Enable ENET receive payload length check.
***kENET_ControlRxPadRemoveEnable*** Padding is removed from received frames.
***kENET_ControlRxBroadCastRejectEnable*** Enable broadcast frame reject.
***kENET_ControlMacAddrInsert*** Enable MAC address insert.
***kENET_ControlStoreAndFwdDisable*** Enable FIFO store and forward.
***kENET_ControlSMIPreambleDisable*** Enable SMI preamble.
***kENET_ControlPromiscuousEnable*** Enable promiscuous mode.
***kENET_ControlMIILoopEnable*** Enable ENET MII loop back.
***kENET_ControlVLANTagEnable*** Enable normal VLAN (single vlan tag).

### 19.6.9 enum enet_interrupt_enable_t

This enumeration uses one-bot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Enumerator

***kENET_BabrInterrupt*** Babbling receive error interrupt source.
***kENET_BabtInterrupt*** Babbling transmit error interrupt source.
***kENET_GraceStopInterrupt*** Graceful stop complete interrupt source.
***kENET_TxFrameInterrupt*** TX FRAME interrupt source.
***kENET_TxBufferInterrupt*** TX BUFFER interrupt source.
***kENET_RxFrameInterrupt*** RX FRAME interrupt source.
***kENET_RxBufferInterrupt*** RX BUFFER interrupt source.
***kENET_MiiInterrupt*** MII interrupt source.
***kENET_EBusERInterrupt*** Ethernet bus error interrupt source.
***kENET_LateCollisionInterrupt*** Late collision interrupt source.
***kENET_RetryLimitInterrupt*** Collision Retry Limit interrupt source.
***kENET_UnderrunInterrupt*** Transmit FIFO underrun interrupt source.
***kENET_PayloadRxInterrupt*** Payload Receive error interrupt source.
***kENET_WakeupInterrupt*** WAKEUP interrupt source.
***kENET_TsAvailInterrupt*** TS AVAIL interrupt source for PTP.
***kENET_TsTimerInterrupt*** TS WRAP interrupt source for PTP.

### 19.6.10 enum enet_event_t

Enumerator

***kENET_RxEvent*** Receive event.
***kENET_TxEvent*** Transmit event.
***kENET_ErrEvent*** Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

***kENET_WakeUpEvent*** Wake up from sleep mode event.
***kENET_TimeStampEvent*** Time stamp event.
***kENET_TimeStampAvailEvent*** Time stamp available event.

### 19.6.11 enum enet_tx_accelerator_t

Enumerator

***kENET_TxAccelIsShift16Enabled*** Transmit FIFO shift-16.
***kENET_TxAccelIpCheckEnabled*** Insert IP header checksum.
***kENET_TxAccelProtoCheckEnabled*** Insert protocol checksum.

### 19.6.12 enum enet_rx_accelerator_t

Enumerator

***kENET_RxAccelPadRemoveEnabled*** Padding removal for short IP frames.
***kENET_RxAccelIpCheckEnabled*** Discard with wrong IP header checksum.
***kENET_RxAccelProtoCheckEnabled*** Discard with wrong protocol checksum.
***kENET_RxAccelMacCheckEnabled*** Discard with Mac layer errors.
***kENET_RxAccelisShift16Enabled*** Receive FIFO shift-16.

### 19.6.13 enum enet_ptp_event_type_t

Enumerator

***kENET_PtpEventMsgType*** PTP event message type.
***kENET_PtpSrcPortIdLen*** PTP message sequence id length.
***kENET_PtpEventPort*** PTP event port number.
***kENET_PtpGnrlPort*** PTP general port number.

### 19.6.14 enum enet_ptp_timer_channel_t

Enumerator

***kENET_PtpTimerChannel1*** IEEE 1588 PTP timer Channel 1.
***kENET_PtpTimerChannel2*** IEEE 1588 PTP timer Channel 2.
***kENET_PtpTimerChannel3*** IEEE 1588 PTP timer Channel 3.
***kENET_PtpTimerChannel4*** IEEE 1588 PTP timer Channel 4.

### 19.6.15 enum enet_ptp_timer_channel_mode_t

Enumerator

***kENET_PtpChannelDisable*** Disable timer channel.
***kENET_PtpChannelRisingCapture*** Input capture on rising edge.
***kENET_PtpChannelFallingCapture*** Input capture on falling edge.
***kENET_PtpChannelBothCapture*** Input capture on both edges.
***kENET_PtpChannelSoftCompare*** Output compare software only.
***kENET_PtpChannelToggleCompare*** Toggle output on compare.
***kENET_PtpChannelClearCompare*** Clear output on compare.
***kENET_PtpChannelSetCompare*** Set output on compare.
***kENET_PtpChannelClearCompareSetOverflow*** Clear output on compare, set output on overflow.
***kENET_PtpChannelSetCompareClearOverflow*** Set output on compare, clear output on overflow.
***kENET_PtpChannelPulseLowonCompare*** Pulse output low on compare for one IEEE 1588 clock cycle.
***kENET_PtpChannelPulseHighonCompare*** Pulse output high on compare for one IEEE 1588 clock cycle.

## 19.7 Function Documentation

### 19.7.1 void ENET_GetDefaultConfig ( enet_config_t * *config* )

The purpose of this API is to get the default ENET MAC controller configure structure for ENET_Init(). User may use the initialized structure unchanged in ENET_Init(), or modify some fields of the structure before calling ENET_Init(). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

| | |
|---|---|
| *config* | The ENET mac controller configuration structure pointer. |

### 19.7.2 void ENET_Init ( ENET_Type * *base*, enet_handle_t * *handle*, const enet_config_t * *config*, const enet_buffer_config_t * *bufferConfig*, uint8_t * *macAddr*, uint32_t *srcClock_Hz* )

This function ungates the module clock and initializes it with the ENET configuration.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | ENET handler pointer. |
| *config* | ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods. |
| *bufferConfig* | ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer. |
| *macAddr* | ENET mac address of Ethernet device. This MAC address should be provided. |
| *srcClock_Hz* | The internal module clock source for MII clock. |

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET_Init().

### 19.7.3 void ENET_Deinit ( ENET_Type * *base* )

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

### 19.7.4 static void ENET_Reset ( ENET_Type * *base* ) [inline], [static]

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

### 19.7.5 void ENET_SetMII ( ENET_Type * *base*, enet_mii_speed_t *speed*, enet_mii_duplex_t *duplex* )

This API is provided to dynamically change the speed and dulpex for MAC.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *speed* | The speed of the RMII mode. |
| *duplex* | The duplex of the RMII mode. |

### 19.7.6 void ENET_SetSMI ( ENET_Type * *base*, uint32_t *srcClock_Hz*, bool *isPreambleDisabled* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *srcClock_Hz* | This is the ENET module clock frequency. Normally it's the system clock. See clock distribution. |
| *isPreamble-Disabled* | The preamble disable flag.<br>• true Enables the preamble.<br>• false Disables the preamble. |

### 19.7.7 static bool ENET_GetSMI ( ENET_Type * *base* ) [inline], [static]

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

Returns

The SMI setup status true or false.

### 19.7.8 static uint32_t ENET_ReadSMIData ( ENET_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

Returns

The data read from PHY

### 19.7.9 void ENET_StartSMIRead ( ENET_Type * *base,* uint32_t *phyAddr,* uint32_t *phyReg,* enet_mii_read_t *operation* )

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *phyAddr* | The PHY address. |
| *phyReg* | The PHY register. Range from 0 ~ 31. |
| *operation* | The read operation. |

### 19.7.10 void ENET_StartSMIWrite ( ENET_Type * *base,* uint32_t *phyAddr,* uint32_t *phyReg,* enet_mii_write_t *operation,* uint32_t *data* )

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *phyAddr* | The PHY address. |
| *phyReg* | The PHY register. Range from 0 ∼ 31. |
| *operation* | The write operation. |
| *data* | The data written to PHY. |

### 19.7.11 void ENET_StartExtC45SMIRead ( ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *phyAddr* | The PHY address. |
| *phyReg* | The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) \| regAddr. |

### 19.7.12 void ENET_StartExtC45SMIWrite ( ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, uint32_t *data* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *phyAddr* | The PHY address. |
| *phyReg* | The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) \| regAddr. |
| *data* | The data written to PHY. |

### 19.7.13 void ENET_SetMacAddr ( ENET_Type * *base*, uint8_t * *macAddr* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *macAddr* | The six-byte Mac address pointer. The pointer is allocated by application and input into the API. |

### 19.7.14 void ENET_GetMacAddr ( ENET_Type * *base,* uint8_t * *macAddr* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *macAddr* | The six-byte Mac address pointer. The pointer is allocated by application and input into the API. |

### 19.7.15 void ENET_AddMulticastGroup ( ENET_Type * *base,* uint8_t * *address* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *address* | The six-byte multicast group address which is provided by application. |

### 19.7.16 void ENET_LeaveMulticastGroup ( ENET_Type * *base,* uint8_t * *address* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *address* | The six-byte multicast group address which is provided by application. |

### 19.7.17 static void ENET_ActiveRead ( ENET_Type * *base* ) [inline], [static]

This function is to active the enet read process. It is used for single descriptor ring/queue.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

Note

This must be called after the MAC configuration and state are ready. It must be called after the ENET_Init() and ENET_Ptp1588Configure(). This should be called when the ENET receive required.

### 19.7.18 static void ENET_EnableSleepMode ( ENET_Type * *base*, bool *enable* ) [inline], [static]

This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *enable* | True enable sleep mode, false disable sleep mode. |

### 19.7.19 static void ENET_GetAccelFunction ( ENET_Type * *base*, uint32_t * *txAccelOption*, uint32_t * *rxAccelOption* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *txAccelOption* | The transmit accelerator option. The "enet_tx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option. |
| *rxAccelOption* | The receive accelerator option. The "enet_rx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option. |

### 19.7.20 static void ENET_EnableInterrupts ( ENET_Type * *base*, uint32_t *mask* ) [inline], [static]

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See enet_interrupt_enable_t. For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

## Function Documentation

```
*     ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt |
      kENET_RxFrameInterrupt);
*
```

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *mask* | ENET interrupts to enable. This is a logical OR of the enumeration :: enet_interrupt-_enable_t. |

### 19.7.21 static void ENET_DisableInterrupts ( ENET_Type * *base*, uint32_t *mask* ) [inline], [static]

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See enet_interrupt_enable_t. For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
*     ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt |
      kENET_RxFrameInterrupt);
*
```

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *mask* | ENET interrupts to disable. This is a logical OR of the enumeration :: enet_interrupt-_enable_t. |

### 19.7.22 static uint32_t ENET_GetInterruptStatus ( ENET_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: enet_interrupt_enable_t.

### 19.7.23 static void ENET_ClearInterruptStatus ( ENET_Type * *base*, uint32_t *mask* ) [inline], [static]

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the enet_interrupt_enable_t. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
*     ENET_ClearInterruptStatus(ENET,
      kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
*
```

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *mask* | ENET interrupt source to be cleared. This is the logical OR of members of the enumeration :: enet_interrupt_enable_t. |

### 19.7.24 void ENET_SetCallback ( enet_handle_t * *handle*, enet_callback_t *callback*, void * *userData* )

This API is provided for the application callback required case when ENET interrupt is enabled. This API should be called after calling ENET_Init.

Parameters

| | |
|---|---|
| *handle* | ENET handler pointer. Should be provided by application. |
| *callback* | The ENET callback function. |
| *userData* | The callback function parameter. |

### 19.7.25 void ENET_GetRxErrBeforeReadFrame ( enet_handle_t * *handle*, enet_data_error_stats_t * *eErrorStatic* )

This API must be called after the ENET_GetRxFrameSize and before the ENET_ReadFrame(). If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```
*       status = ENET_GetRxFrameSize(&g_handle, &length);
*       if (status == kStatus_ENET_RxFrameError)
*       {
*           // Get the error information of the received frame.
*           ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic);
*           // update the receive buffer.
*           ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
*       }
*
```

Parameters

| | |
|---|---|
| *handle* | The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init. |
| *eErrorStatic* | The error statistics structure pointer. |

### 19.7.26 status_t ENET_GetTxErrAfterSendFrame ( enet_handle_t * *handle*, enet_data_error_stats_t * *eErrorStatic* )

This interface gets the error statistics of the transmit frame. Because the error information is reported by the uDMA after the data delivery, this interface should be called after the data transmit API. It is recommended to call this function on transmit interrupt handler. After calling the ENET_SendFrame, the transmit interrupt notifies the transmit completion.

Parameters

| | |
|---|---|
| *handle* | The PTP handler pointer. This is the same handler pointer used in the ENET_Init. |
| *eErrorStatic* | The error statistics structure pointer. |

Returns

The execute status.

### 19.7.27 status_t ENET_GetRxFrameSize ( enet_handle_t * *handle*, uint32_t * *length* )

This function gets a received frame size from the ENET buffer descriptors.

Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, ENET_ReadFrame() should be called to update the receive buffers If the result is not "kStatus_ENET_RxFrameEmpty".

Parameters

| | |
|---|---|
| *handle* | The ENET handler structure. This is the same handler pointer used in the ENET_Init. |
| *length* | The length of the valid frame received. |

Return values

| | |
|---|---|
| *kStatus_ENET_RxFrame-Empty* | No frame received. Should not call ENET_ReadFrame to read frame. |
| *kStatus_ENET_RxFrame-Error* | Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers. |
| *kStatus_Success* | Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input. |

### 19.7.28 status_t ENET_ReadFrame ( ENET_Type * *base*, enet_handle_t * *handle*, uint8_t * *data*, uint32_t *length* )

This function reads a frame (both the data and the length) from the ENET buffer descriptors. The ENET-_GetRxFrameSize should be used to get the size of the prepared data buffer. This is an example:

```
*       uint32_t length;
*       enet_handle_t g_handle;
*       //Get the received frame size firstly.
*       status = ENET_GetRxFrameSize(&g_handle, &length);
*       if (length != 0)
*       {
*           //Allocate memory here with the size of "length"
*           uint8_t *data = memory allocate interface;
*           if (!data)
*           {
*               ENET_ReadFrame(ENET, &g_handle, NULL, 0);
*               //Add the console warning log.
*           }
*           else
*           {
*              status = ENET_ReadFrame(ENET, &g_handle, data, length);
*              //Call stack input API to deliver the data to stack
*           }
*       }
*       else if (status == kStatus_ENET_RxFrameError)
*       {
*          //Update the received buffer when a error frame is received.
*           ENET_ReadFrame(ENET, &g_handle, NULL, 0);
*       }
*
```

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET handler structure. This is the same handler pointer used in the ENET_Init. |
| *data* | The data buffer provided by user to store the frame which memory size should be at least "length". |
| *length* | The size of the data buffer which is still the length of the received frame. |

Returns

The execute status, successful or failure.

### 19.7.29 status_t ENET_SendFrame ( ENET_Type * *base,* enet_handle_t * *handle,* const uint8_t * *data,* uint32_t *length* )

Note

The CRC is automatically appended to the data. Input the data to send without the CRC.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| *data* | The data buffer provided by user to be send. |
| *length* | The length of the data to be send. |

Return values

| | |
|---|---|
| *kStatus_Success* | Send frame succeed. |
| *kStatus_ENET_TxFrame-Busy* | Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus-_ENET_TxFrameBusy. |

### 19.7.30 void ENET_TransmitIRQHandler ( ENET_Type * *base,* enet_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET handler pointer. |

### 19.7.31 void ENET_ReceiveIRQHandler ( ENET_Type * *base*, enet_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET handler pointer. |

### 19.7.32 void ENET_ErrorIRQHandler ( ENET_Type * *base*, enet_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET handler pointer. |

### 19.7.33 void ENET_CommonFrame0IRQHandler ( ENET_Type * *base* )

This is used for the combined tx/rx/error interrupt for single/mutli-ring (frame 0).

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

### 19.7.34 void ENET_Ptp1588Configure ( ENET_Type * *base*, enet_handle_t * *handle*, enet_ptp_config_t * *ptpConfig* )

The function sets the clock for PTP 1588 timer and enables time stamp interrupts and transmit interrupts for PTP 1588 features. This API should be called when the 1588 feature is enabled or the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE is defined. ENET_Init should be called before calling this API.

Note

The PTP 1588 time-stamp second increase though time-stamp interrupt handler and the transmit time-stamp store is done through transmit interrupt handler. As a result, the TS interrupt and TX interrupt are enabled when you call this API.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | ENET handler pointer. |
| *ptpConfig* | The ENET PTP1588 configuration. |

### 19.7.35 void ENET_Ptp1588StartTimer ( ENET_Type * *base*, uint32_t *ptpClkSrc* )

This function is used to initialize the PTP timer. After the PTP starts, the PTP timer starts running.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *ptpClkSrc* | The clock source of the PTP timer. |

### 19.7.36 static void ENET_Ptp1588StopTimer ( ENET_Type * *base* ) [inline], [static]

This function is used to stops the ENET PTP timer.

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |

### 19.7.37 void ENET_Ptp1588AdjustTimer ( ENET_Type * *base*, uint32_t *corrIncrease*, uint32_t *corrPeriod* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *corrIncrease* | The correction increment value. This value is added every time the correction timer expires. A value less than the PTP timer frequency(1/ptpClkSrc) slows down the timer, a value greater than the 1/ptpClkSrc speeds up the timer. |
| *corrPeriod* | The PTP timer correction counter wrap-around value. This defines after how many timer clock the correction counter should be reset and trigger a correction increment on the timer. A value of 0 disables the correction counter and no correction occurs. |

### 19.7.38 static void ENET_Ptp1588SetChannelMode ( ENET_Type * *base,* enet_ptp_timer_channel_t *channel,* enet_ptp_timer_channel_mode_t *mode,* bool *intEnable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *channel* | The ENET PTP timer channel number. |
| *mode* | The PTP timer channel mode, see "enet_ptp_timer_channel_mode_t". |
| *intEnable* | Enables or disables the interrupt. |

### 19.7.39 static void ENET_Ptp1588SetChannelOutputPulseWidth ( ENET_Type * *base,* enet_ptp_timer_channel_t *channel,* bool *isOutputLow,* uint8_t *pulseWidth,* bool *intEnable* ) [inline], [static]

For the input "mode" in ENET_Ptp1588SetChannelMode, the kENET_PtpChannelPulseLowonCompare kENET_PtpChannelPulseHighonCompare only support the pulse width for one 1588 clock. this function is extended for control the pulse width from 1 to 32 1588 clock cycles. so call this function if you need to set the timer channel mode for kENET_PtpChannelPulseLowonCompare or kENET_PtpChannelPulse-HighonCompare with pulse width more than one 1588 clock,

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *channel* | The ENET PTP timer channel number. |
| *isOutputLow* | True — timer channel is configured for output compare pulse output low. false — timer channel is configured for output compare pulse output high. |
| *pulseWidth* | The pulse width control value, range from 0 $\sim$ 31. 0 — pulse width is one 1588 clock cycle. 31 — pulse width is thirty two 1588 clock cycles. |
| *intEnable* | Enables or disables the interrupt. |

### 19.7.40 static void ENET_Ptp1588SetChannelCmpValue ( ENET_Type * *base,* enet_ptp_timer_channel_t *channel,* uint32_t *cmpValue* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *channel* | The PTP timer channel, see "enet_ptp_timer_channel_t". |
| *cmpValue* | The compare value for the compare setting. |

### 19.7.41 static bool ENET_Ptp1588GetChannelStatus ( ENET_Type * *base,* enet_ptp_timer_channel_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *channel* | The IEEE 1588 timer channel number. |

Returns

True or false, Compare or capture operation status

### 19.7.42 static void ENET_Ptp1588ClearChannelStatus ( ENET_Type * *base,* enet_ptp_timer_channel_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *channel* | The IEEE 1588 timer channel number. |

### 19.7.43 void ENET_Ptp1588GetTimer ( ENET_Type * *base,* enet_handle_t * *handle,* enet_ptp_time_t * *ptpTime* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET state pointer. This is the same state pointer used in the ENET_Init. |
| *ptpTime* | The PTP timer structure. |

### 19.7.44 void ENET_Ptp1588SetTimer ( ENET_Type * *base*, enet_handle_t * *handle*, enet_ptp_time_t * *ptpTime* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET state pointer. This is the same state pointer used in the ENET_Init. |
| *ptpTime* | The timer to be set to the PTP timer. |

### 19.7.45 void ENET_Ptp1588TimerIRQHandler ( ENET_Type * *base*, enet_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | ENET peripheral base address. |
| *handle* | The ENET state pointer. This is the same state pointer used in the ENET_Init. |

### 19.7.46 status_t ENET_GetRxFrameTime ( enet_handle_t * *handle*, enet_ptp_time_data_t * *ptpTimeData* )

This function is used for PTP stack to get the timestamp captured by the ENET driver.

Parameters

| | |
|---|---|
| *handle* | The ENET handler pointer.This is the same state pointer used in ENET_Init. |
| *ptpTimeData* | The special PTP timestamp data for search the receive timestamp. |

Return values

| | |
|---|---|
| *kStatus_Success* | Get 1588 timestamp success. |
| *kStatus_ENET_PtpTs-RingEmpty* | 1588 timestamp ring empty. |
| *kStatus_ENET_PtpTs-RingFull* | 1588 timestamp ring full. |

### 19.7.47 status_t ENET_GetTxFrameTime ( enet_handle_t * *handle,* enet_ptp_time_data_t * *ptpTimeData* )

This function is used for PTP stack to get the timestamp captured by the ENET driver.

Parameters

| | |
|---|---|
| *handle* | The ENET handler pointer.This is the same state pointer used in ENET_Init. |
| *ptpTimeData* | The special PTP timestamp data for search the receive timestamp. |

Return values

| | |
|---|---|
| *kStatus_Success* | Get 1588 timestamp success. |
| *kStatus_ENET_PtpTs-RingEmpty* | 1588 timestamp ring empty. |
| *kStatus_ENET_PtpTs-RingFull* | 1588 timestamp ring full. |

# Chapter 20
# EWM: External Watchdog Monitor Driver

## 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the External Watchdog (EWM) Driver module of MCUXpresso SDK devices.

## 20.2 Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ewm

## Data Structures

- struct ewm_config_t
  *Describes EWM clock source. More...*

## Enumerations

- enum _ewm_interrupt_enable_t { kEWM_InterruptEnable = EWM_CTRL_INTEN_MASK }
  *EWM interrupt configuration structure with default settings all disabled.*
- enum _ewm_status_flags_t { kEWM_RunningFlag = EWM_CTRL_EWMEN_MASK }
  *EWM status flags.*

## Driver version

- #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *EWM driver version 2.0.1.*

## EWM initialization and de-initialization

- void EWM_Init (EWM_Type *base, const ewm_config_t *config)
  *Initializes the EWM peripheral.*
- void EWM_Deinit (EWM_Type *base)
  *Deinitializes the EWM peripheral.*
- void EWM_GetDefaultConfig (ewm_config_t *config)
  *Initializes the EWM configuration structure.*

## EWM functional Operation

- static void EWM_EnableInterrupts (EWM_Type *base, uint32_t mask)
  *Enables the EWM interrupt.*
- static void EWM_DisableInterrupts (EWM_Type *base, uint32_t mask)
  *Disables the EWM interrupt.*
- static uint32_t EWM_GetStatusFlags (EWM_Type *base)
  *Gets all status flags.*
- void EWM_Refresh (EWM_Type *base)
  *Services the EWM.*

## 20.3 Data Structure Documentation

### 20.3.1 struct ewm_config_t

Data structure for EWM configuration.

This structure is used to configure the EWM.

#### Data Fields

- bool enableEwm
  *Enable EWM module.*
- bool enableEwmInput
  *Enable EWM_in input.*
- bool setInputAssertLogic
  *EWM_in signal assertion state.*
- bool enableInterrupt
  *Enable EWM interrupt.*
- uint8_t compareLowValue
  *Compare low-register value.*
- uint8_t compareHighValue
  *Compare high-register value.*

## 20.4 Macro Definition Documentation

### 20.4.1 #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

## 20.5 Enumeration Type Documentation

### 20.5.1 enum _ewm_interrupt_enable_t

This structure contains the settings for all of EWM interrupt configurations.

Enumerator

***kEWM_InterruptEnable*** Enable the EWM to generate an interrupt.

### 20.5.2 enum _ewm_status_flags_t

This structure contains the constants for the EWM status flags for use in the EWM functions.

Enumerator

***kEWM_RunningFlag*** Running flag, set when EWM is enabled.

## 20.6 Function Documentation

### 20.6.1 void EWM_Init ( EWM_Type * *base,* const ewm_config_t * *config* )

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 |

### 20.6.2 void EWM_Deinit ( EWM_Type * *base* )

This function is used to shut down the EWM.

Parameters

| | |
|---|---|
| *base* | EWM peripheral base address |

### 20.6.3 void EWM_GetDefaultConfig ( ewm_config_t * *config* )

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;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to the EWM configuration structure. |

See Also

ewm_config_t

### 20.6.4 static void EWM_EnableInterrupts ( EWM_Type * *base,* uint32_t *mask* ) [inline], [static]

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<br>• kEWM_InterruptEnable |

### 20.6.5 static void EWM_DisableInterrupts ( EWM_Type * *base,* uint32_t *mask* ) [inline], [static]

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<br>• kEWM_InterruptEnable |

### 20.6.6 static uint32_t EWM_GetStatusFlags ( EWM_Type * *base* ) [inline], [static]

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;
*
```

Parameters

| | |
|---|---|
| *base* | EWM peripheral base address |

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

_ewm_status_flags_t
- True: a related status flag has been set.
- False: a related status flag is not set.

### 20.6.7 void EWM_Refresh ( EWM_Type * *base* )

This function resets the EWM counter to zero.

Parameters

| | |
|---|---|
| *base* | EWM peripheral base address |

# Chapter 21
# FlexCAN: Flex Controller Area Network Driver

## 21.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flex Controller Area Network (FlexCAN) module of MCUXpresso SDK devices.

## Modules

- FlexCAN Driver
- FlexCAN eDMA Driver

## 21.2 FlexCAN Driver

### 21.2.1 Overview

This section describes the programming interface of the FlexCAN driver. The FlexCAN driver configures FlexCAN module and provides functional and transactional interfaces to build the FlexCAN application.

### 21.2.2 Typical use case

#### 21.2.2.1 Message Buffer Send Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/flexcan

#### 21.2.2.2 Message Buffer Receive Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/flexcan

#### 21.2.2.3 Receive FIFO Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/flexcan

### Data Structures

- struct flexcan_frame_t
  *FlexCAN message frame structure.* *More...*
- struct flexcan_timing_config_t
  *FlexCAN protocol timing characteristic configuration structure.* *More...*
- struct flexcan_config_t
  *FlexCAN module configuration structure.* *More...*
- struct flexcan_rx_mb_config_t
  *FlexCAN Receive Message Buffer configuration structure.* *More...*
- struct flexcan_rx_fifo_config_t
  *FlexCAN Rx FIFO configuration structure.* *More...*
- struct flexcan_mb_transfer_t
  *FlexCAN Message Buffer transfer.* *More...*
- struct flexcan_fifo_transfer_t
  *FlexCAN Rx FIFO transfer.* *More...*
- struct flexcan_handle_t
  *FlexCAN handle structure.* *More...*

## Macros

- #define FLEXCAN_ID_STD(id) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)
  *FlexCAN Frame ID helper macro.*
- #define FLEXCAN_ID_EXT(id)
  *Extend Frame ID helper macro.*
- #define FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)
  *FlexCAN Rx Message Buffer Mask helper macro.*
- #define FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)
  *Extend Rx Message Buffer Mask helper macro.*
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)
  *FlexCAN Rx FIFO Mask helper macro.*
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)
  *Standard Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)
  *Standard Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id) (((uint32_t)(id)&0x7F8) << 21)
  *Standard Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id) (((uint32_t)(id)&0x7F8) << 13)
  *Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id) (((uint32_t)(id)&0x7F8) << 5)
  *Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id) (((uint32_t)(id)&0x7F8) >> 3)
  *Standard Rx FIFO Mask helper macro Type C lower part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)
  *Extend Rx FIFO Mask helper macro Type A helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)
  *Extend Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)
  *Extend Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)
  *Extend Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)
  *Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)
  *Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)
  *Extend Rx FIFO Mask helper macro Type C lower part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)
  *FlexCAN Rx FIFO Filter helper macro.*
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)
  *Standard Rx FIFO Filter helper macro Type B upper part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)

*Standard Rx FIFO Filter helper macro Type B lower part helper macro.*

- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)
  *Standard Rx FIFO Filter helper macro Type C upper part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)
  *Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)
  *Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.*
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)
  *Standard Rx FIFO Filter helper macro Type C lower part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)
  *Extend Rx FIFO Filter helper macro Type A helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)
  *Extend Rx FIFO Filter helper macro Type B upper part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)
  *Extend Rx FIFO Filter helper macro Type B lower part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)
  *Extend Rx FIFO Filter helper macro Type C upper part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)
  *Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)
  *Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.*
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)
  *Extend Rx FIFO Filter helper macro Type C lower part helper macro.*

## Typedefs

- typedef void(∗ flexcan_transfer_callback_t )(CAN_Type ∗base, flexcan_handle_t ∗handle, status_t status, uint32_t result, void ∗userData)
  *FlexCAN transfer callback function.*

## Enumerations

- enum _flexcan_status {
  kStatus_FLEXCAN_TxBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 0),
  kStatus_FLEXCAN_TxIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 1),
  kStatus_FLEXCAN_TxSwitchToRx,
  kStatus_FLEXCAN_RxBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 3),
  kStatus_FLEXCAN_RxIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 4),
  kStatus_FLEXCAN_RxOverflow = MAKE_STATUS(kStatusGroup_FLEXCAN, 5),
  kStatus_FLEXCAN_RxFifoBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 6),
  kStatus_FLEXCAN_RxFifoIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 7),
  kStatus_FLEXCAN_RxFifoOverflow = MAKE_STATUS(kStatusGroup_FLEXCAN, 8),
  kStatus_FLEXCAN_RxFifoWarning = MAKE_STATUS(kStatusGroup_FLEXCAN, 9),
  kStatus_FLEXCAN_ErrorStatus = MAKE_STATUS(kStatusGroup_FLEXCAN, 10),
  kStatus_FLEXCAN_WakeUp = MAKE_STATUS(kStatusGroup_FLEXCAN, 11),
  kStatus_FLEXCAN_UnHandled = MAKE_STATUS(kStatusGroup_FLEXCAN, 12) }
  *FlexCAN transfer status.*
- enum flexcan_frame_format_t {
  kFLEXCAN_FrameFormatStandard = 0x0U,
  kFLEXCAN_FrameFormatExtend = 0x1U }
  *FlexCAN frame format.*
- enum flexcan_frame_type_t {
  kFLEXCAN_FrameTypeData = 0x0U,
  kFLEXCAN_FrameTypeRemote = 0x1U }
  *FlexCAN frame type.*
- enum flexcan_clock_source_t {
  kFLEXCAN_ClkSrcOsc = 0x0U,
  kFLEXCAN_ClkSrcPeri = 0x1U }
  *FlexCAN clock source.*
- enum flexcan_wake_up_source_t {
  kFLEXCAN_WakeupSrcUnfiltered = 0x0U,
  kFLEXCAN_WakeupSrcFiltered = 0x1U }
  *FlexCAN wake up source.*
- enum flexcan_rx_fifo_filter_type_t {
  kFLEXCAN_RxFifoFilterTypeA = 0x0U,
  kFLEXCAN_RxFifoFilterTypeB,
  kFLEXCAN_RxFifoFilterTypeC,
  kFLEXCAN_RxFifoFilterTypeD = 0x3U }
  *FlexCAN Rx Fifo Filter type.*
- enum flexcan_rx_fifo_priority_t {
  kFLEXCAN_RxFifoPrioLow = 0x0U,
  kFLEXCAN_RxFifoPrioHigh = 0x1U }
  *FlexCAN Rx FIFO priority.*
- enum _flexcan_interrupt_enable {

kFLEXCAN_BusOffInterruptEnable = CAN_CTRL1_BOFFMSK_MASK,
kFLEXCAN_ErrorInterruptEnable = CAN_CTRL1_ERRMSK_MASK,
kFLEXCAN_RxWarningInterruptEnable = CAN_CTRL1_RWRNMSK_MASK,
kFLEXCAN_TxWarningInterruptEnable = CAN_CTRL1_TWRNMSK_MASK,
kFLEXCAN_WakeUpInterruptEnable = CAN_MCR_WAKMSK_MASK }
*FlexCAN interrupt configuration structure, default settings all disabled.*

- enum _flexcan_flags {
  kFLEXCAN_SynchFlag = CAN_ESR1_SYNCH_MASK,
  kFLEXCAN_TxWarningIntFlag = CAN_ESR1_TWRNINT_MASK,
  kFLEXCAN_RxWarningIntFlag = CAN_ESR1_RWRNINT_MASK,
  kFLEXCAN_TxErrorWarningFlag = CAN_ESR1_TXWRN_MASK,
  kFLEXCAN_RxErrorWarningFlag = CAN_ESR1_RXWRN_MASK,
  kFLEXCAN_IdleFlag = CAN_ESR1_IDLE_MASK,
  kFLEXCAN_FaultConfinementFlag = CAN_ESR1_FLTCONF_MASK,
  kFLEXCAN_TransmittingFlag = CAN_ESR1_TX_MASK,
  kFLEXCAN_ReceivingFlag = CAN_ESR1_RX_MASK,
  kFLEXCAN_BusOffIntFlag = CAN_ESR1_BOFFINT_MASK,
  kFLEXCAN_ErrorIntFlag = CAN_ESR1_ERRINT_MASK,
  kFLEXCAN_WakeUpIntFlag = CAN_ESR1_WAKINT_MASK }
  *FlexCAN status flags.*
- enum _flexcan_error_flags {
  kFLEXCAN_StuffingError = CAN_ESR1_STFERR_MASK,
  kFLEXCAN_FormError = CAN_ESR1_FRMERR_MASK,
  kFLEXCAN_CrcError = CAN_ESR1_CRCERR_MASK,
  kFLEXCAN_AckError = CAN_ESR1_ACKERR_MASK,
  kFLEXCAN_Bit0Error = CAN_ESR1_BIT0ERR_MASK,
  kFLEXCAN_Bit1Error = CAN_ESR1_BIT1ERR_MASK }
  *FlexCAN error status flags.*
- enum _flexcan_rx_fifo_flags {
  kFLEXCAN_RxFifoOverflowFlag = CAN_IFLAG1_BUF7I_MASK,
  kFLEXCAN_RxFifoWarningFlag = CAN_IFLAG1_BUF6I_MASK,
  kFLEXCAN_RxFifoFrameAvlFlag = CAN_IFLAG1_BUF5I_MASK }
  *FlexCAN Rx FIFO status flags.*

## Driver version

- #define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  *FlexCAN driver version 2.4.0.*

## Initialization and deinitialization

- uint32_t FLEXCAN_GetInstance (CAN_Type *base)
  *Get the FlexCAN instance from peripheral base address.*
- bool FLEXCAN_CalculateImprovedTimingValues (uint32_t baudRate, uint32_t sourceClock_Hz, flexcan_timing_config_t *pconfig)

*Calculates the improved timing values by specific baudrates for classical CAN.*
- void FLEXCAN_Init (CAN_Type *base, const flexcan_config_t *config, uint32_t sourceClock_Hz)
  *Initializes a FlexCAN instance.*
- void FLEXCAN_Deinit (CAN_Type *base)
  *De-initializes a FlexCAN instance.*
- void FLEXCAN_GetDefaultConfig (flexcan_config_t *config)
  *Gets the default configuration structure.*

## Configuration.

- void FLEXCAN_SetTimingConfig (CAN_Type *base, const flexcan_timing_config_t *config)
  *Sets the FlexCAN protocol timing characteristic.*
- void FLEXCAN_SetRxMbGlobalMask (CAN_Type *base, uint32_t mask)
  *Sets the FlexCAN receive message buffer global mask.*
- void FLEXCAN_SetRxFifoGlobalMask (CAN_Type *base, uint32_t mask)
  *Sets the FlexCAN receive FIFO global mask.*
- void FLEXCAN_SetRxIndividualMask (CAN_Type *base, uint8_t maskIdx, uint32_t mask)
  *Sets the FlexCAN receive individual mask.*
- void FLEXCAN_SetTxMbConfig (CAN_Type *base, uint8_t mbIdx, bool enable)
  *Configures a FlexCAN transmit message buffer.*
- void FLEXCAN_SetRxMbConfig (CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t *config, bool enable)
  *Configures a FlexCAN Receive Message Buffer.*
- void FLEXCAN_SetRxFifoConfig (CAN_Type *base, const flexcan_rx_fifo_config_t *config, bool enable)
  *Configures the FlexCAN Rx FIFO.*

## Status

- static uint32_t FLEXCAN_GetStatusFlags (CAN_Type *base)
  *Gets the FlexCAN module interrupt flags.*
- static void FLEXCAN_ClearStatusFlags (CAN_Type *base, uint32_t mask)
  *Clears status flags with the provided mask.*
- static void FLEXCAN_GetBusErrCount (CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)
  *Gets the FlexCAN Bus Error Counter value.*
- static uint64_t FLEXCAN_GetMbStatusFlags (CAN_Type *base, uint64_t mask)
  *Gets the FlexCAN Message Buffer interrupt flags.*
- static void FLEXCAN_ClearMbStatusFlags (CAN_Type *base, uint64_t mask)
  *Clears the FlexCAN Message Buffer interrupt flags.*

## Interrupts

- static void FLEXCAN_EnableInterrupts (CAN_Type *base, uint32_t mask)
  *Enables FlexCAN interrupts according to the provided mask.*
- static void FLEXCAN_DisableInterrupts (CAN_Type *base, uint32_t mask)
  *Disables FlexCAN interrupts according to the provided mask.*
- static void FLEXCAN_EnableMbInterrupts (CAN_Type *base, uint64_t mask)

*Enables FlexCAN Message Buffer interrupts.*

- static void FLEXCAN_DisableMbInterrupts (CAN_Type *base, uint64_t mask)
  *Disables FlexCAN Message Buffer interrupts.*

## Bus Operations

- static void FLEXCAN_Enable (CAN_Type *base, bool enable)
  *Enables or disables the FlexCAN module operation.*
- status_t FLEXCAN_WriteTxMb (CAN_Type *base, uint8_t mbIdx, const flexcan_frame_t *txFrame)
  *Writes a FlexCAN Message to the Transmit Message Buffer.*
- status_t FLEXCAN_ReadRxMb (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *rxFrame)
  *Reads a FlexCAN Message from Receive Message Buffer.*
- status_t FLEXCAN_ReadRxFifo (CAN_Type *base, flexcan_frame_t *rxFrame)
  *Reads a FlexCAN Message from Rx FIFO.*

## Transactional

- status_t FLEXCAN_TransferSendBlocking (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *txFrame)
  *Performs a polling send transaction on the CAN bus.*
- status_t FLEXCAN_TransferReceiveBlocking (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *rxFrame)
  *Performs a polling receive transaction on the CAN bus.*
- status_t FLEXCAN_TransferReceiveFifoBlocking (CAN_Type *base, flexcan_frame_t *rxFrame)
  *Performs a polling receive transaction from Rx FIFO on the CAN bus.*
- void FLEXCAN_TransferCreateHandle (CAN_Type *base, flexcan_handle_t *handle, flexcan_transfer_callback_t callback, void *userData)
  *Initializes the FlexCAN handle.*
- status_t FLEXCAN_TransferSendNonBlocking (CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *xfer)
  *Sends a message using IRQ.*
- status_t FLEXCAN_TransferReceiveNonBlocking (CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *xfer)
  *Receives a message using IRQ.*
- status_t FLEXCAN_TransferReceiveFifoNonBlocking (CAN_Type *base, flexcan_handle_t *handle, flexcan_fifo_transfer_t *xfer)
  *Receives a message from Rx FIFO using IRQ.*
- void FLEXCAN_TransferAbortSend (CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
  *Aborts the interrupt driven message send process.*
- void FLEXCAN_TransferAbortReceive (CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
  *Aborts the interrupt driven message receive process.*
- void FLEXCAN_TransferAbortReceiveFifo (CAN_Type *base, flexcan_handle_t *handle)
  *Aborts the interrupt driven message receive from Rx FIFO process.*
- void FLEXCAN_TransferHandleIRQ (CAN_Type *base, flexcan_handle_t *handle)
  *FlexCAN IRQ handle function.*

### 21.2.3 Data Structure Documentation

#### 21.2.3.1 struct flexcan_frame_t

##### 21.2.3.1.0.28 Field Documentation

###### 21.2.3.1.0.28.1 uint32_t flexcan_frame_t::timestamp

###### 21.2.3.1.0.28.2 uint32_t flexcan_frame_t::length

###### 21.2.3.1.0.28.3 uint32_t flexcan_frame_t::type

###### 21.2.3.1.0.28.4 uint32_t flexcan_frame_t::format

###### 21.2.3.1.0.28.5 uint32_t flexcan_frame_t::__pad0__

###### 21.2.3.1.0.28.6 uint32_t flexcan_frame_t::idhit

###### 21.2.3.1.0.28.7 uint32_t flexcan_frame_t::id

###### 21.2.3.1.0.28.8 uint32_t flexcan_frame_t::dataWord0

###### 21.2.3.1.0.28.9 uint32_t flexcan_frame_t::dataWord1

###### 21.2.3.1.0.28.10 uint8_t flexcan_frame_t::dataByte3

###### 21.2.3.1.0.28.11 uint8_t flexcan_frame_t::dataByte2

###### 21.2.3.1.0.28.12 uint8_t flexcan_frame_t::dataByte1

###### 21.2.3.1.0.28.13 uint8_t flexcan_frame_t::dataByte0

###### 21.2.3.1.0.28.14 uint8_t flexcan_frame_t::dataByte7

###### 21.2.3.1.0.28.15 uint8_t flexcan_frame_t::dataByte6

###### 21.2.3.1.0.28.16 uint8_t flexcan_frame_t::dataByte5

###### 21.2.3.1.0.28.17 uint8_t flexcan_frame_t::dataByte4

#### 21.2.3.2 struct flexcan_timing_config_t

**Data Fields**

- uint16_t preDivider
  *Clock Pre-scaler Division Factor.*
- uint8_t rJumpwidth
  *Re-sync Jump Width.*
- uint8_t phaseSeg1
  *Phase Segment 1.*

- uint8_t phaseSeg2
  *Phase Segment 2.*
- uint8_t propSeg
  *Propagation Segment.*

#### 21.2.3.2.0.29 Field Documentation

#### 21.2.3.2.0.29.1 uint16_t flexcan_timing_config_t::preDivider

#### 21.2.3.2.0.29.2 uint8_t flexcan_timing_config_t::rJumpwidth

#### 21.2.3.2.0.29.3 uint8_t flexcan_timing_config_t::phaseSeg1

#### 21.2.3.2.0.29.4 uint8_t flexcan_timing_config_t::phaseSeg2

#### 21.2.3.2.0.29.5 uint8_t flexcan_timing_config_t::propSeg

### 21.2.3.3 struct flexcan_config_t

### Data Fields

- uint32_t baudRate
  *FlexCAN baud rate in bps.*
- flexcan_clock_source_t clkSrc
  *Clock source for FlexCAN Protocol Engine.*
- flexcan_wake_up_source_t wakeupSrc
  *Wake up source selection.*
- uint8_t maxMbNum
  *The maximum number of Message Buffers used by user.*
- bool enableLoopBack
  *Enable or Disable Loop Back Self Test Mode.*
- bool enableTimerSync
  *Enable or Disable Timer Synchronization.*
- bool enableSelfWakeup
  *Enable or Disable Self Wakeup Mode.*
- bool enableIndividMask
  *Enable or Disable Rx Individual Mask.*

#### 21.2.3.3.0.30 Field Documentation

##### 21.2.3.3.0.30.1 uint32_t flexcan_config_t::baudRate

##### 21.2.3.3.0.30.2 flexcan_clock_source_t flexcan_config_t::clkSrc

##### 21.2.3.3.0.30.3 flexcan_wake_up_source_t flexcan_config_t::wakeupSrc

##### 21.2.3.3.0.30.4 uint8_t flexcan_config_t::maxMbNum

##### 21.2.3.3.0.30.5 bool flexcan_config_t::enableLoopBack

##### 21.2.3.3.0.30.6 bool flexcan_config_t::enableTimerSync

##### 21.2.3.3.0.30.7 bool flexcan_config_t::enableSelfWakeup

##### 21.2.3.3.0.30.8 bool flexcan_config_t::enableIndividMask

### 21.2.3.4 struct flexcan_rx_mb_config_t

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

#### Data Fields

- uint32_t id
  *CAN Message Buffer Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.*
- flexcan_frame_format_t format
  *CAN Frame Identifier format(Standard of Extend).*
- flexcan_frame_type_t type
  *CAN Frame Type(Data or Remote).*

#### 21.2.3.4.0.31 Field Documentation

##### 21.2.3.4.0.31.1 uint32_t flexcan_rx_mb_config_t::id

##### 21.2.3.4.0.31.2 flexcan_frame_format_t flexcan_rx_mb_config_t::format

##### 21.2.3.4.0.31.3 flexcan_frame_type_t flexcan_rx_mb_config_t::type

### 21.2.3.5 struct flexcan_rx_fifo_config_t

#### Data Fields

- uint32_t * idFilterTable
  *Pointer to the FlexCAN Rx FIFO identifier filter table.*

- uint8_t idFilterNum
  *The quantity of filter elements.*
- flexcan_rx_fifo_filter_type_t idFilterType
  *The FlexCAN Rx FIFO Filter type.*
- flexcan_rx_fifo_priority_t priority
  *The FlexCAN Rx FIFO receive priority.*

#### 21.2.3.5.0.32 Field Documentation

##### 21.2.3.5.0.32.1 uint32_t* flexcan_rx_fifo_config_t::idFilterTable

##### 21.2.3.5.0.32.2 uint8_t flexcan_rx_fifo_config_t::idFilterNum

##### 21.2.3.5.0.32.3 flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType

##### 21.2.3.5.0.32.4 flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority

### 21.2.3.6 struct flexcan_mb_transfer_t

### Data Fields

- flexcan_frame_t * frame
  *The buffer of CAN Message to be transfer.*
- uint8_t mbIdx
  *The index of Message buffer used to transfer Message.*

#### 21.2.3.6.0.33 Field Documentation

##### 21.2.3.6.0.33.1 flexcan_frame_t* flexcan_mb_transfer_t::frame

##### 21.2.3.6.0.33.2 uint8_t flexcan_mb_transfer_t::mbIdx

### 21.2.3.7 struct flexcan_fifo_transfer_t

### Data Fields

- flexcan_frame_t * frame
  *The buffer of CAN Message to be received from Rx FIFO.*

#### 21.2.3.7.0.34 Field Documentation

##### 21.2.3.7.0.34.1 flexcan_frame_t* flexcan_fifo_transfer_t::frame

### 21.2.3.8 struct _flexcan_handle

FlexCAN handle structure definition.

### Data Fields

- flexcan_transfer_callback_t callback

*Callback function.*

- void * userData
  *FlexCAN callback function parameter.*
- flexcan_frame_t *volatile mbFrameBuf [CAN_WORD1_COUNT]
  *The buffer for received data from Message Buffers.*
- flexcan_frame_t *volatile rxFifoFrameBuf
  *The buffer for received data from Rx FIFO.*
- volatile uint8_t mbState [CAN_WORD1_COUNT]
  *Message Buffer transfer state.*
- volatile uint8_t rxFifoState
  *Rx FIFO transfer state.*

#### 21.2.3.8.0.35 Field Documentation

##### 21.2.3.8.0.35.1 flexcan_transfer_callback_t flexcan_handle_t::callback

##### 21.2.3.8.0.35.2 void* flexcan_handle_t::userData

##### 21.2.3.8.0.35.3 flexcan_frame_t* volatile flexcan_handle_t::mbFrameBuf[CAN_WORD1_COUNT]

##### 21.2.3.8.0.35.4 flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf

##### 21.2.3.8.0.35.5 volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]

##### 21.2.3.8.0.35.6 volatile uint8_t flexcan_handle_t::rxFifoState

### 21.2.4 Macro Definition Documentation

#### 21.2.4.1 #define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

#### 21.2.4.2 #define FLEXCAN_ID_STD( *id* ) (((uint32_t)(((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)

Standard Frame ID helper macro.

#### 21.2.4.3 #define FLEXCAN_ID_EXT( *id* )

**Value:**

```
(((uint32_t)(((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
     (CAN_ID_EXT_MASK | CAN_ID_STD_MASK))
```

#### 21.2.4.4 #define FLEXCAN_RX_MB_STD_MASK( *id, rtr, ide* )

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
     FLEXCAN_ID_STD(id))
```

Standard Rx Message Buffer Mask helper macro.

#### 21.2.4.5 #define FLEXCAN_RX_MB_EXT_MASK( *id, rtr, ide* )

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
     FLEXCAN_ID_EXT(id))
```

#### 21.2.4.6 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_A( *id, rtr, ide* )

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
     (FLEXCAN_ID_STD(id) << 1))
```

Standard Rx FIFO Mask helper macro Type A helper macro.

#### 21.2.4.7 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH( *id, rtr, ide* )

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
     (((uint32_t)(id)&0x7FF) << 19))
```

#### 21.2.4.8 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW( *id, rtr, ide* )

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
     (((uint32_t)(id)&0x7FF) << 3))
```

#### 21.2.4.9 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH( *id* ) (((uint32_t)(id)&0x7F8) << 21)

#### 21.2.4.10 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH( *id* ) (((uint32_t)(id)&0x7F8) << 13)

#### 21.2.4.11 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW( *id* ) (((uint32_t)(id)&0x7F8) << 5)

#### 21.2.4.12 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW( *id* ) (((uint32_t)(id)&0x7F8) >> 3)

#### 21.2.4.13 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A( *id, rtr, ide* )

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
     (FLEXCAN_ID_EXT(id) << 1))
```

#### 21.2.4.14 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH( *id, rtr, ide* )

**Value:**

```
(                                                                          \
        ((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
        ((FLEXCAN_ID_EXT(id) & 0x1FFF8000)                                   \
         << 1))
```

#### 21.2.4.15 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW( *id, rtr, ide* )

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
     ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >>                                 \
      15))
```

#### 21.2.4.16 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH( *id* ) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)

#### 21.2.4.17 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH( *id* )

**Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>           \
     5)
```

#### 21.2.4.18 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW( *id* )

**Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>           \
     13)
```

#### 21.2.4.19 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW( *id* ) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)

#### 21.2.4.20 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A( *id, rtr, ide* ) FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)

Standard Rx FIFO Filter helper macro Type A helper macro.

#### 21.2.4.21 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH( *id, rtr, ide* )

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(                   \
        id, rtr, ide)
```

#### 21.2.4.22 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW( *id, rtr, ide* )

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(                    \
        id, rtr, ide)
```

#### 21.2.4.23 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH( *id* )

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(           \
        id)
```

#### 21.2.4.24 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH( *id* )

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(           \
        id)
```

#### 21.2.4.25 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW( *id* )

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(          \
        id)
```

#### 21.2.4.26 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW( *id* )

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(          \
        id)
```

#### 21.2.4.27 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A( *id, rtr, ide* ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)

#### 21.2.4.28 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH( *id, rtr, ide* )

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(                  \
        id, rtr, ide)
```

#### 21.2.4.29 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW( *id, rtr, ide* )

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(                  \
        id, rtr, ide)
```

#### 21.2.4.30 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH( *id* )

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(          \
        id)
```

#### 21.2.4.31 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH( *id* )

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(          \
        id)
```

#### 21.2.4.32 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW( *id* )

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(          \
        id)
```

#### 21.2.4.33 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW( *id* ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)

### 21.2.5 Typedef Documentation

#### 21.2.5.1 typedef void(∗ flexcan_transfer_callback_t)(CAN_Type ∗base, flexcan_handle_t ∗handle, status_t status, uint32_t result, void ∗userData)

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_FLEXCAN_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

### 21.2.6 Enumeration Type Documentation

#### 21.2.6.1 enum _flexcan_status

Enumerator

***kStatus_FLEXCAN_TxBusy*** Tx Message Buffer is Busy.
***kStatus_FLEXCAN_TxIdle*** Tx Message Buffer is Idle.
***kStatus_FLEXCAN_TxSwitchToRx*** Remote Message is send out and Message buffer changed to Receive one.
***kStatus_FLEXCAN_RxBusy*** Rx Message Buffer is Busy.
***kStatus_FLEXCAN_RxIdle*** Rx Message Buffer is Idle.
***kStatus_FLEXCAN_RxOverflow*** Rx Message Buffer is Overflowed.

*kStatus_FLEXCAN_RxFifoBusy* Rx Message FIFO is Busy.
*kStatus_FLEXCAN_RxFifoIdle* Rx Message FIFO is Idle.
*kStatus_FLEXCAN_RxFifoOverflow* Rx Message FIFO is overflowed.
*kStatus_FLEXCAN_RxFifoWarning* Rx Message FIFO is almost overflowed.
*kStatus_FLEXCAN_ErrorStatus* FlexCAN Module Error and Status.
*kStatus_FLEXCAN_WakeUp* FlexCAN is waken up from STOP mode.
*kStatus_FLEXCAN_UnHandled* UnHadled Interrupt asserted.

#### 21.2.6.2 enum flexcan_frame_format_t

Enumerator

*kFLEXCAN_FrameFormatStandard* Standard frame format attribute.
*kFLEXCAN_FrameFormatExtend* Extend frame format attribute.

#### 21.2.6.3 enum flexcan_frame_type_t

Enumerator

*kFLEXCAN_FrameTypeData* Data frame type attribute.
*kFLEXCAN_FrameTypeRemote* Remote frame type attribute.

#### 21.2.6.4 enum flexcan_clock_source_t

Enumerator

*kFLEXCAN_ClkSrcOsc* FlexCAN Protocol Engine clock from Oscillator.
*kFLEXCAN_ClkSrcPeri* FlexCAN Protocol Engine clock from Peripheral Clock.

#### 21.2.6.5 enum flexcan_wake_up_source_t

Enumerator

*kFLEXCAN_WakeupSrcUnfiltered* FlexCAN uses unfiltered Rx input to detect edge.
*kFLEXCAN_WakeupSrcFiltered* FlexCAN uses filtered Rx input to detect edge.

#### 21.2.6.6 enum flexcan_rx_fifo_filter_type_t

Enumerator

*kFLEXCAN_RxFifoFilterTypeA* One full ID (standard and extended) per ID Filter element.

*kFLEXCAN_RxFifoFilterTypeB* Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.
*kFLEXCAN_RxFifoFilterTypeC* Four partial 8-bit Standard or extended ID slices per ID Filter Table element.
*kFLEXCAN_RxFifoFilterTypeD* All frames rejected.

#### 21.2.6.7 enum flexcan_rx_fifo_priority_t

The matching process starts from the Rx MB(or Rx FIFO) with higher priority. If no MB(or Rx FIFO filter) is satisfied, the matching process goes on with the Rx FIFO(or Rx MB) with lower priority.

Enumerator

*kFLEXCAN_RxFifoPrioLow* Matching process start from Rx Message Buffer first.
*kFLEXCAN_RxFifoPrioHigh* Matching process start from Rx FIFO first.

#### 21.2.6.8 enum _flexcan_interrupt_enable

This structure contains the settings for all of the FlexCAN Module interrupt configurations. Note: FlexCAN Message Buffers and Rx FIFO have their own interrupts.

Enumerator

*kFLEXCAN_BusOffInterruptEnable* Bus Off interrupt.
*kFLEXCAN_ErrorInterruptEnable* Error interrupt.
*kFLEXCAN_RxWarningInterruptEnable* Rx Warning interrupt.
*kFLEXCAN_TxWarningInterruptEnable* Tx Warning interrupt.
*kFLEXCAN_WakeUpInterruptEnable* Wake Up interrupt.

#### 21.2.6.9 enum _flexcan_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions. Note: The CPU read action clears FlEXCAN_ErrorFlag, therefore user need to read FlEXCAN_ErrorFlag and distinguish which error is occur using _flexcan_error_flags enumerations.

Enumerator

*kFLEXCAN_SynchFlag* CAN Synchronization Status.
*kFLEXCAN_TxWarningIntFlag* Tx Warning Interrupt Flag.
*kFLEXCAN_RxWarningIntFlag* Rx Warning Interrupt Flag.
*kFLEXCAN_TxErrorWarningFlag* Tx Error Warning Status.
*kFLEXCAN_RxErrorWarningFlag* Rx Error Warning Status.
*kFLEXCAN_IdleFlag* CAN IDLE Status Flag.

***kFLEXCAN_FaultConfinementFlag*** Fault Confinement State Flag.
***kFLEXCAN_TransmittingFlag*** FlexCAN In Transmission Status.
***kFLEXCAN_ReceivingFlag*** FlexCAN In Reception Status.
***kFLEXCAN_BusOffIntFlag*** Bus Off Interrupt Flag.
***kFLEXCAN_ErrorIntFlag*** Error Interrupt Flag.
***kFLEXCAN_WakeUpIntFlag*** Wake-Up Interrupt Flag.

#### 21.2.6.10 enum _flexcan_error_flags

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in _flexcan_flags enumerations to ditermine which error is generated.

Enumerator

***kFLEXCAN_StuffingError*** Stuffing Error.
***kFLEXCAN_FormError*** Form Error.
***kFLEXCAN_CrcError*** Cyclic Redundancy Check Error.
***kFLEXCAN_AckError*** Received no ACK on transmission.
***kFLEXCAN_Bit0Error*** Unable to send dominant bit.
***kFLEXCAN_Bit1Error*** Unable to send recessive bit.

#### 21.2.6.11 enum _flexcan_rx_fifo_flags

The FlexCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ∼ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Enumerator

***kFLEXCAN_RxFifoOverflowFlag*** Rx FIFO overflow flag.
***kFLEXCAN_RxFifoWarningFlag*** Rx FIFO almost full flag.
***kFLEXCAN_RxFifoFrameAvlFlag*** Frames available in Rx FIFO flag.

### 21.2.7 Function Documentation

#### 21.2.7.1 uint32_t FLEXCAN_GetInstance ( CAN_Type * *base* )

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |

Returns

FlexCAN instance.

#### 21.2.7.2 bool FLEXCAN_CalculateImprovedTimingValues ( uint32_t *baudRate*, uint32_t *sourceClock_Hz*, flexcan_timing_config_t * *pconfig* )

Parameters

| | |
|---|---|
| *baudRate* | The classical CAN speed in bps defined by user |
| *sourceClock_Hz* | The Source clock data speed in bps. Zero to disable baudrate switching |
| *pconfig* | Pointer to the FlexCAN timing configuration structure. |

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

#### 21.2.7.3 void FLEXCAN_Init ( CAN_Type * *base*, const flexcan_config_t * *config*, uint32_t *sourceClock_Hz* )

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
*   flexcan_config_t flexcanConfig;
*   flexcanConfig.clkSrc            = kFLEXCAN_ClkSrcOsc;
*   flexcanConfig.baudRate          = 1000000U;
*   flexcanConfig.maxMbNum          = 16;
*   flexcanConfig.enableLoopBack    = false;
*   flexcanConfig.enableSelfWakeup  = false;
*   flexcanConfig.enableIndividMask = false;
*   flexcanConfig.enableDoze        = false;
*   flexcanConfig.timingConfig      = timingConfig;
*   FLEXCAN_Init(CAN0, &flexcanConfig, 8000000UL);
*
```

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *config* | Pointer to the user-defined configuration structure. |
| *sourceClock_-Hz* | FlexCAN Protocol Engine clock source frequency in Hz. |

#### 21.2.7.4 void FLEXCAN_Deinit ( CAN_Type * *base* )

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |

#### 21.2.7.5 void FLEXCAN_GetDefaultConfig ( flexcan_config_t * *config* )

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrcOsc; flexcanConfig->baudRate = 1000000U; flexcanConfig->baudRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enable-LoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->enableDoze = false; flexcanConfig.timingConfig = timingConfig;

Parameters

| | |
|---|---|
| *config* | Pointer to the FlexCAN configuration structure. |

#### 21.2.7.6 void FLEXCAN_SetTimingConfig ( CAN_Type * *base*, const flexcan_timing_config_t * *config* )

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_Init() and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling FLEXCAN_SetTimingConfig() overrides the baud rate set in FLEXCAN_Init().

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *config* | Pointer to the timing configuration structure. |

#### 21.2.7.7 void FLEXCAN_SetRxMbGlobalMask ( CAN_Type * *base*, uint32_t *mask* )

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | Rx Message Buffer Global Mask value. |

#### 21.2.7.8 void FLEXCAN_SetRxFifoGlobalMask ( CAN_Type * *base*, uint32_t *mask* )

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | Rx Fifo Global Mask value. |

#### 21.2.7.9 void FLEXCAN_SetRxIndividualMask ( CAN_Type * *base*, uint8_t *maskIdx*, uint32_t *mask* )

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *maskIdx* | The Index of individual Mask. |

| | |
|---|---|
| *mask* | Rx Individual Mask value. |

#### 21.2.7.10 void FLEXCAN_SetTxMbConfig ( CAN_Type * *base,* uint8_t *mbIdx,* bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mbIdx* | The Message Buffer index. |
| *enable* | Enable/disable Tx Message Buffer.<br>• true: Enable Tx Message Buffer.<br>• false: Disable Tx Message Buffer. |

#### 21.2.7.11 void FLEXCAN_SetRxMbConfig ( CAN_Type * *base,* uint8_t *mbIdx,* const flexcan_rx_mb_config_t * *config,* bool *enable* )

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mbIdx* | The Message Buffer index. |
| *config* | Pointer to the FlexCAN Message Buffer configuration structure. |
| *enable* | Enable/disable Rx Message Buffer.<br>• true: Enable Rx Message Buffer.<br>• false: Disable Rx Message Buffer. |

#### 21.2.7.12 void FLEXCAN_SetRxFifoConfig ( CAN_Type * *base,* const flexcan_rx_fifo_config_t * *config,* bool *enable* )

This function configures the Rx FIFO with given Rx FIFO configuration.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *config* | Pointer to the FlexCAN Rx FIFO configuration structure. |
| *enable* | Enable/disable Rx FIFO.<br>• true: Enable Rx FIFO.<br>• false: Disable Rx FIFO. |

#### 21.2.7.13 static uint32_t FLEXCAN_GetStatusFlags ( CAN_Type * *base* ) [inline], [static]

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators _flexcan_flags. To check the specific status, compare the return value with enumerators in _flexcan_flags.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |

Returns

FlexCAN status flags which are ORed by the enumerators in the _flexcan_flags.

#### 21.2.7.14 static void FLEXCAN_ClearStatusFlags ( CAN_Type * *base,* uint32_t *mask* ) [inline], [static]

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | The status flags to be cleared, it is logical OR value of _flexcan_flags. |

#### 21.2.7.15 static void FLEXCAN_GetBusErrCount ( CAN_Type * *base,* uint8_t * *txErrBuf,* uint8_t * *rxErrBuf* ) [inline], [static]

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *txErrBuf* | Buffer to store Tx Error Counter value. |
| *rxErrBuf* | Buffer to store Rx Error Counter value. |

#### 21.2.7.16 static uint64_t FLEXCAN_GetMbStatusFlags ( CAN_Type * *base*, uint64_t *mask* ) [inline], [static]

This function gets the interrupt flags of a given Message Buffers.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | The ORed FlexCAN Message Buffer mask. |

Returns

The status of given Message Buffers.

#### 21.2.7.17 static void FLEXCAN_ClearMbStatusFlags ( CAN_Type * *base*, uint64_t *mask* ) [inline], [static]

This function clears the interrupt flags of a given Message Buffers.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | The ORed FlexCAN Message Buffer mask. |

#### 21.2.7.18 static void FLEXCAN_EnableInterrupts ( CAN_Type * *base*, uint32_t *mask* ) [inline], [static]

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | The interrupts to enable. Logical OR of _flexcan_interrupt_enable. |

#### 21.2.7.19 static void FLEXCAN_DisableInterrupts ( CAN_Type * *base*, uint32_t *mask* ) [inline], [static]

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | The interrupts to disable. Logical OR of _flexcan_interrupt_enable. |

#### 21.2.7.20 static void FLEXCAN_EnableMbInterrupts ( CAN_Type * *base*, uint64_t *mask* ) [inline], [static]

This function enables the interrupts of given Message Buffers.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | The ORed FlexCAN Message Buffer mask. |

#### 21.2.7.21 static void FLEXCAN_DisableMbInterrupts ( CAN_Type * *base*, uint64_t *mask* ) [inline], [static]

This function disables the interrupts of given Message Buffers.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mask* | The ORed FlexCAN Message Buffer mask. |

#### 21.2.7.22 static void FLEXCAN_Enable ( CAN_Type * *base*, bool *enable* ) [inline], [static]

This function enables or disables the FlexCAN module.

Parameters

| | |
|---|---|
| *base* | FlexCAN base pointer. |
| *enable* | true to enable, false to disable. |

#### 21.2.7.23 status_t FLEXCAN_WriteTxMb ( CAN_Type * *base*, uint8_t *mbIdx*, const flexcan_frame_t * *txFrame* )

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mbIdx* | The FlexCAN Message Buffer index. |
| *txFrame* | 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. |

#### 21.2.7.24 status_t FLEXCAN_ReadRxMb ( CAN_Type * *base*, uint8_t *mbIdx*, flexcan_frame_t * *rxFrame* )

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *mbIdx* | The FlexCAN Message Buffer index. |
| *rxFrame* | Pointer to CAN message frame structure for reception. |

Return values

| | |
|---|---|
| *kStatus_Success* | - Rx Message Buffer is full and has been read successfully. |
| *kStatus_FLEXCAN_Rx-Overflow* | - Rx Message Buffer is already overflowed and has been read successfully. |
| *kStatus_Fail* | - Rx Message Buffer is empty. |

#### 21.2.7.25 status_t FLEXCAN_ReadRxFifo ( CAN_Type * *base*, flexcan_frame_t * *rxFrame* )

This function reads a CAN message from the FlexCAN build-in Rx FIFO.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *rxFrame* | Pointer to CAN message frame structure for reception. |

Return values

| | |
|---|---|
| *kStatus_Success* | - Read Message from Rx FIFO successfully. |
| *kStatus_Fail* | - Rx FIFO is not enabled. |

#### 21.2.7.26 status_t FLEXCAN_TransferSendBlocking ( CAN_Type * *base*, uint8_t *mbIdx*, flexcan_frame_t * *txFrame* )

Note that a transfer handle does not need to be created before calling this API.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base pointer. |
| *mbIdx* | The FlexCAN Message Buffer index. |
| *txFrame* | 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. |

#### 21.2.7.27 status_t FLEXCAN_TransferReceiveBlocking ( CAN_Type * *base,* uint8_t *mbIdx,* flexcan_frame_t * *rxFrame* )

Note that a transfer handle does not need to be created before calling this API.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base pointer. |
| *mbIdx* | The FlexCAN Message Buffer index. |
| *rxFrame* | Pointer to CAN message frame structure for reception. |

Return values

| | |
|---|---|
| *kStatus_Success* | - Rx Message Buffer is full and has been read successfully. |
| *kStatus_FLEXCAN_RxOverflow* | - Rx Message Buffer is already overflowed and has been read successfully. |
| *kStatus_Fail* | - Rx Message Buffer is empty. |

### 21.2.7.28 status_t FLEXCAN_TransferReceiveFifoBlocking ( CAN_Type * *base*, flexcan_frame_t * *rxFrame* )

Note that a transfer handle does not need to be created before calling this API.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base pointer. |
| *rxFrame* | Pointer to CAN message frame structure for reception. |

Return values

| | |
|---|---|
| *kStatus_Success* | - Read Message from Rx FIFO successfully. |
| *kStatus_Fail* | - Rx FIFO is not enabled. |

### 21.2.7.29 void FLEXCAN_TransferCreateHandle ( CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_transfer_callback_t *callback*, void * *userData* )

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

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |

#### 21.2.7.30 status_t FLEXCAN_TransferSendNonBlocking ( CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_mb_transfer_t * *xfer* )

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |
| *xfer* | FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start Tx Message Buffer sending process successfully. |
| *kStatus_Fail* | Write Tx Message Buffer failed. |
| *kStatus_FLEXCAN_TxBusy* | Tx Message Buffer is in use. |

#### 21.2.7.31 status_t FLEXCAN_TransferReceiveNonBlocking ( CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_mb_transfer_t * *xfer* )

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |
| *xfer* | FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | - Start Rx Message Buffer receiving process successfully. |
| *kStatus_FLEXCAN_RxBusy* | - Rx Message Buffer is in use. |

### 21.2.7.32 status_t FLEXCAN_TransferReceiveFifoNonBlocking ( CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_fifo_transfer_t * *xfer* )

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |
| *xfer* | FlexCAN Rx FIFO transfer structure. See the flexcan_fifo_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | - Start Rx FIFO receiving process successfully. |
| *kStatus_FLEXCAN_RxFifoBusy* | - Rx FIFO is currently in use. |

### 21.2.7.33 void FLEXCAN_TransferAbortSend ( CAN_Type * *base*, flexcan_handle_t * *handle*, uint8_t *mbIdx* )

This function aborts the interrupt driven message send process.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |
| *mbIdx* | The FlexCAN Message Buffer index. |

### 21.2.7.34 void FLEXCAN_TransferAbortReceive ( CAN_Type * *base*, flexcan_handle_t * *handle*, uint8_t *mbIdx* )

This function aborts the interrupt driven message receive process.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |
| *mbIdx* | The FlexCAN Message Buffer index. |

#### 21.2.7.35 void FLEXCAN_TransferAbortReceiveFifo ( CAN_Type * *base,* flexcan_handle_t * *handle* )

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |

#### 21.2.7.36 void FLEXCAN_TransferHandleIRQ ( CAN_Type * *base,* flexcan_handle_t * *handle* )

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | FlexCAN handle pointer. |

## 21.3 FlexCAN eDMA Driver

### 21.3.1 Overview

#### Data Structures

- struct flexcan_edma_handle_t
  *FlexCAN eDMA handle. More...*

#### Typedefs

- typedef void(∗ flexcan_edma_transfer_callback_t )(CAN_Type ∗base, flexcan_edma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexCAN transfer callback function.*

#### Driver version

- #define FSL_FLEXCAN_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  *FlexCAN EDMA driver version 2.4.0.*

#### eDMA transactional

- void FLEXCAN_TransferCreateHandleEDMA (CAN_Type ∗base, flexcan_edma_handle_t ∗handle, flexcan_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗rxFifoEdmaHandle)
  *Initializes the FlexCAN handle, which is used in transactional functions.*
- status_t FLEXCAN_TransferReceiveFifoEDMA (CAN_Type ∗base, flexcan_edma_handle_t ∗handle, flexcan_fifo_transfer_t ∗xfer)
  *Receives the CAN Message from the Rx FIFO using eDMA.*
- void FLEXCAN_TransferAbortReceiveFifoEDMA (CAN_Type ∗base, flexcan_edma_handle_t ∗handle)
  *Aborts the receive process which used eDMA.*

### 21.3.2 Data Structure Documentation

#### 21.3.2.1 struct _flexcan_edma_handle

#### Data Fields

- flexcan_edma_transfer_callback_t callback
  *Callback function.*
- void ∗ userData
  *FlexCAN callback function parameter.*
- edma_handle_t ∗ rxFifoEdmaHandle

*The EDMA Rx FIFO channel used.*

- volatile uint8_t rxFifoState
  *Rx FIFO transfer state.*

#### 21.3.2.1.0.36 Field Documentation

##### 21.3.2.1.0.36.1 flexcan_edma_transfer_callback_t flexcan_edma_handle_t::callback

##### 21.3.2.1.0.36.2 void* flexcan_edma_handle_t::userData

##### 21.3.2.1.0.36.3 edma_handle_t* flexcan_edma_handle_t::rxFifoEdmaHandle

##### 21.3.2.1.0.36.4 volatile uint8_t flexcan_edma_handle_t::rxFifoState

### 21.3.3 Macro Definition Documentation

#### 21.3.3.1 #define FSL_FLEXCAN_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

### 21.3.4 Typedef Documentation

#### 21.3.4.1 typedef void(* flexcan_edma_transfer_callback_t)(CAN_Type *base, flexcan_edma_handle_t *handle, status_t status, void *userData)

### 21.3.5 Function Documentation

#### 21.3.5.1 void FLEXCAN_TransferCreateHandleEDMA ( CAN_Type * *base*, flexcan_edma_handle_t * *handle*, flexcan_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *rxFifoEdmaHandle* )

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | Pointer to flexcan_edma_handle_t structure. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |
| *rxFifoEdma-Handle* | User-requested DMA handle for Rx FIFO DMA transfer. |

#### 21.3.5.2 status_t FLEXCAN_TransferReceiveFifoEDMA ( CAN_Type * *base*, flexcan_edma_handle_t * *handle*, flexcan_fifo_transfer_t * *xfer* )

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | Pointer to flexcan_edma_handle_t structure. |
| *xfer* | FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_FLEXCAN_RxFifoBusy* | Previous transfer ongoing. |

#### 21.3.5.3 void FLEXCAN_TransferAbortReceiveFifoEDMA ( CAN_Type * *base*, flexcan_edma_handle_t * *handle* )

This function aborts the receive process which used eDMA.

Parameters

| | |
|---|---|
| *base* | FlexCAN peripheral base address. |
| *handle* | Pointer to flexcan_edma_handle_t structure. |

# Chapter 22
# FLEXRAM: on-chip RAM manager

## 22.1 Overview

The MCUXpresso SDK provides a driver for the FLEXRAM module of MCUXpresso SDK devices.

The FLEXRAM module intergrates the ITCM, DTCM, and OCRAM controller, and supports parameterized RAM array and RAM array portioning.

This example code shows how to allocate RAM using the FLEXRAM driver.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/flexram

## Data Structures

- struct flexram_allocate_ram_t
  *FLEXRAM allocate ocram, itcm, dtcm size. More...*

## Enumerations

- enum _flexram_wr_rd_sel {
  kFLEXRAM_Read = 0U,
  kFLEXRAM_Write = 1U }
  *flexram write read sel*
- enum _flexram_interrupt_status {
  kFLEXRAM_OCRAMAccessError = FLEXRAM_INT_STATUS_OCRAM_ERR_STATUS_MASK,
  kFLEXRAM_DTCMAccessError = FLEXRAM_INT_STATUS_DTCM_ERR_STATUS_MASK,
  kFLEXRAM_ITCMAccessError = FLEXRAM_INT_STATUS_ITCM_ERR_STATUS_MASK,
  kFLEXRAM_InterruptStatusAll }
  *Interrupt status flag mask.*
- enum flexram_tcm_access_mode_t {
  kFLEXRAM_TCMAccessFastMode = 0U,
  kFLEXRAM_TCMAccessWaitMode = 1U }
  *FLEXRAM TCM access mode Fast access mode expected to be finished in 1-cycle Wait access mode expected to be finished in 2-cycle Wait access mode is a feature of the flexram and it should be used when the cpu clock too fast to finish tcm access in 1-cycle.*
- enum _flexram_bank_type {
  kFLEXRAM_BankNotUsed = 0U,
  kFLEXRAM_BankOCRAM = 1U,
  kFLEXRAM_BankDTCM = 2U,
  kFLEXRAM_BankITCM = 3U }
  *FLEXRAM bank type.*
- enum _flexram_tcm_size {

kFLEXRAM_TCMSize32KB = 32 ∗ 1024U,
kFLEXRAM_TCMSize64KB = 64 ∗ 1024U,
kFLEXRAM_TCMSize128KB = 128 ∗ 1024U,
kFLEXRAM_TCMSize256KB = 256 ∗ 1024U,
kFLEXRAM_TCMSize512KB = 512 ∗ 1024U }
*FLEXRAM tcm support size.*

- enum flexram_bank_allocate_src_t {
  kFLEXRAM_BankAllocateThroughHardwareFuse = 0U,
  kFLEXRAM_BankAllocateThroughBankCfg = 1U }
  *FLEXRAM bank allocate source.*

## Variables

- const uint8_t flexram_allocate_ram_t::ocramBankNum
  *ocram banknumber which the SOC support*
- const uint8_t flexram_allocate_ram_t::dtcmBankNum
  *dtcm bank number to allocate, the number should be power of 2*
- const uint8_t flexram_allocate_ram_t::itcmBankNum
  *itcm bank number to allocate, the number should be power of 2*

## Driver version

- #define FSL_FLEXRAM_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 4U))
  *Driver version 2.0.4.*

## Initialization and deinitialization

- void FLEXRAM_Init (FLEXRAM_Type ∗base)
  *FLEXRAM module initialization function.*
- void FLEXRAN_Deinit (FLEXRAM_Type ∗base)
  *Deinitializes the FLEXRAM.*

## Status

- static uint32_t FLEXRAM_GetInterruptStatus (FLEXRAM_Type ∗base)
  *FLEXRAM module get interrupt status.*
- static void FLEXRAM_ClearInterruptStatus (FLEXRAM_Type ∗base, uint32_t status)
  *FLEXRAM module clear interrupt status.*
- static void FLEXRAM_EnableInterruptStatus (FLEXRAM_Type ∗base, uint32_t status)
  *FLEXRAM module enable interrupt status.*
- static void FLEXRAM_DisableInterruptStatus (FLEXRAM_Type ∗base, uint32_t status)
  *FLEXRAM module disable interrupt status.*

## Interrupts

- static void FLEXRAM_EnableInterruptSignal (FLEXRAM_Type ∗base, uint32_t status)
  *FLEXRAM module enable interrupt.*
- static void FLEXRAM_DisableInterruptSignal (FLEXRAM_Type ∗base, uint32_t status)
  *FLEXRAM module disable interrupt.*

### functional

- static void FLEXRAM_SetTCMReadAccessMode (FLEXRAM_Type *base, flexram_tcm_access_mode_t mode)
  *FLEXRAM module set TCM read access mode.*
- static void FLEXRAM_SetTCMWriteAccessMode (FLEXRAM_Type *base, flexram_tcm_access_mode_t mode)
  *FLEXRAM module set TCM write access mode.*
- static void FLEXRAM_EnableForceRamClockOn (FLEXRAM_Type *base, bool enable)
  *FLEXRAM module force ram clock on.*
- status_t FLEXRAM_AllocateRam (flexram_allocate_ram_t *config)
  *FLEXRAM allocate on-chip ram for OCRAM,ITCM,DTCM This function is independent of FLEXRAM_Init, it can be called directly if ram re-allocate is needed.*
- static void FLEXRAM_SetAllocateRamSrc (flexram_bank_allocate_src_t src)
  *FLEXRAM set allocate on-chip ram source.*
- void FLEXRAM_SetTCMSize (uint8_t itcmBankNum, uint8_t dtcmBankNum)
  *FLEXRAM configure TCM size This function is used to set the TCM to the target size.*

## 22.2 Data Structure Documentation

### 22.2.1 struct flexram_allocate_ram_t

#### Data Fields

- const uint8_t ocramBankNum
  *ocram banknumber which the SOC support*
- const uint8_t dtcmBankNum
  *dtcm bank number to allocate, the number should be power of 2*
- const uint8_t itcmBankNum
  *itcm bank number to allocate, the number should be power of 2*

## 22.3 Macro Definition Documentation

### 22.3.1 #define FSL_FLEXRAM_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 4U))

## 22.4 Enumeration Type Documentation

### 22.4.1 enum _flexram_wr_rd_sel

Enumerator

***kFLEXRAM_Read*** read
***kFLEXRAM_Write*** write

### 22.4.2 enum _flexram_interrupt_status

Enumerator

***kFLEXRAM_OCRAMAccessError*** ocram access unallocated address
***kFLEXRAM_DTCMAccessError*** dtcm access unallocated address
***kFLEXRAM_ITCMAccessError*** itcm access unallocated address
***kFLEXRAM_InterruptStatusAll*** all the interrupt status mask

### 22.4.3 enum flexram_tcm_access_mode_t

Normally, fast mode is the default mode, the efficiency of the tcm access will better.

Enumerator

***kFLEXRAM_TCMAccessFastMode*** fast access mode
***kFLEXRAM_TCMAccessWaitMode*** wait access mode

### 22.4.4 enum _flexram_bank_type

Enumerator

***kFLEXRAM_BankNotUsed*** bank is not used
***kFLEXRAM_BankOCRAM*** bank is OCRAM
***kFLEXRAM_BankDTCM*** bank is DTCM
***kFLEXRAM_BankITCM*** bank is ITCM

### 22.4.5 enum _flexram_tcm_size

Enumerator

***kFLEXRAM_TCMSize32KB*** TCM total size 32KB.
***kFLEXRAM_TCMSize64KB*** TCM total size 64KB.
***kFLEXRAM_TCMSize128KB*** TCM total size 128KB.
***kFLEXRAM_TCMSize256KB*** TCM total size 256KB.
***kFLEXRAM_TCMSize512KB*** TCM total size 512KB.

### 22.4.6 enum flexram_bank_allocate_src_t

Enumerator

***kFLEXRAM_BankAllocateThroughHardwareFuse*** allocate ram through hardware fuse value
***kFLEXRAM_BankAllocateThroughBankCfg*** allocate ram through FLEXRAM_BANK_CFG

## 22.5 Function Documentation

### 22.5.1 void FLEXRAM_Init ( FLEXRAM_Type * *base* )

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |

### 22.5.2 static uint32_t FLEXRAM_GetInterruptStatus ( FLEXRAM_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |

### 22.5.3 static void FLEXRAM_ClearInterruptStatus ( FLEXRAM_Type * *base,* uint32_t *status* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *status* | status to clear. |

### 22.5.4 static void FLEXRAM_EnableInterruptStatus ( FLEXRAM_Type * *base,* uint32_t *status* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *status* | status to enable. |

### 22.5.5 static void FLEXRAM_DisableInterruptStatus ( FLEXRAM_Type * *base,* uint32_t *status* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *status* | status to disable. |

### 22.5.6 static void FLEXRAM_EnableInterruptSignal ( FLEXRAM_Type * *base*, uint32_t *status* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *status* | status interrupt to enable. |

### 22.5.7 static void FLEXRAM_DisableInterruptSignal ( FLEXRAM_Type * *base*, uint32_t *status* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *status* | status interrupt to disable. |

### 22.5.8 static void FLEXRAM_SetTCMReadAccessMode ( FLEXRAM_Type * *base*, flexram_tcm_access_mode_t *mode* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *mode* | access mode. |

### 22.5.9 static void FLEXRAM_SetTCMWriteAccessMode ( FLEXRAM_Type * *base*, flexram_tcm_access_mode_t *mode* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *mode* | access mode. |

### 22.5.10 static void FLEXRAM_EnableForceRamClockOn ( FLEXRAM_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXRAM base address. |
| *enable* | enable or disable clock force on. |

### 22.5.11 status_t FLEXRAM_AllocateRam ( flexram_allocate_ram_t * *config* )

Parameters

| | |
|---|---|
| *config* | allocate configuration. |

Return values

| | |
|---|---|
| *kStatus_InvalidArgument* | the argument is invalid kStatus_Success allocate success |

### 22.5.12 static void FLEXRAM_SetAllocateRamSrc ( flexram_bank_allocate_src_t *src* ) [inline], [static]

Parameters

| | |
|---|---|
| *src* | bank config source select value. |

### 22.5.13 void FLEXRAM_SetTCMSize ( uint8_t *itcmBankNum,* uint8_t *dtcmBankNum* )

If a odd bank number is used, a new banknumber will be used which is bigger than target value, application can set tcm size to the biggest bank number always, then boundary access error can be captured by flexram only. When access to the TCM memory boundary ,hardfault will raised by core.

Parameters

| | |
|---|---|
| *itcmBankNum* | itcm bank number to allocate |
| *dtcmBankNum* | dtcm bank number to allocate |

# Chapter 23
# FlexIO: FlexIO Driver

## 23.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FlexIO drivers for the FlexIO module of MCUXpresso SDK devices.

## Modules

- FlexIO Camera Driver
- FlexIO Driver
- FlexIO I2C Master Driver
- FlexIO I2S Driver
- FlexIO MCU Interface LCD Driver
- FlexIO SPI Driver
- FlexIO UART Driver

## 23.2 FlexIO Driver

### 23.2.1 Overview

### Data Structures

- struct flexio_config_t
  *Define FlexIO user configuration structure. More...*
- struct flexio_timer_config_t
  *Define FlexIO timer configuration structure. More...*
- struct flexio_shifter_config_t
  *Define FlexIO shifter configuration structure. More...*

### Macros

- #define FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x) ((uint32_t)(x) << 1U)
  *Calculate FlexIO timer trigger.*

### Typedefs

- typedef void(* flexio_isr_t )(void *base, void *handle)
  *typedef for FlexIO simulated driver interrupt handler.*

### Enumerations

- enum flexio_timer_trigger_polarity_t {
  kFLEXIO_TimerTriggerPolarityActiveHigh = 0x0U,
  kFLEXIO_TimerTriggerPolarityActiveLow = 0x1U }
  *Define time of timer trigger polarity.*
- enum flexio_timer_trigger_source_t {
  kFLEXIO_TimerTriggerSourceExternal = 0x0U,
  kFLEXIO_TimerTriggerSourceInternal = 0x1U }
  *Define type of timer trigger source.*
- enum flexio_pin_config_t {
  kFLEXIO_PinConfigOutputDisabled = 0x0U,
  kFLEXIO_PinConfigOpenDrainOrBidirection = 0x1U,
  kFLEXIO_PinConfigBidirectionOutputData = 0x2U,
  kFLEXIO_PinConfigOutput = 0x3U }
  *Define type of timer/shifter pin configuration.*
- enum flexio_pin_polarity_t {
  kFLEXIO_PinActiveHigh = 0x0U,
  kFLEXIO_PinActiveLow = 0x1U }
  *Definition of pin polarity.*

- enum flexio_timer_mode_t {
  kFLEXIO_TimerModeDisabled = 0x0U,
  kFLEXIO_TimerModeDual8BitBaudBit = 0x1U,
  kFLEXIO_TimerModeDual8BitPWM = 0x2U,
  kFLEXIO_TimerModeSingle16Bit = 0x3U }
  *Define type of timer work mode.*
- enum flexio_timer_output_t {
  kFLEXIO_TimerOutputOneNotAffectedByReset = 0x0U,
  kFLEXIO_TimerOutputZeroNotAffectedByReset = 0x1U,
  kFLEXIO_TimerOutputOneAffectedByReset = 0x2U,
  kFLEXIO_TimerOutputZeroAffectedByReset = 0x3U }
  *Define type of timer initial output or timer reset condition.*
- enum flexio_timer_decrement_source_t {
  kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput = 0x0U,
  kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput = 0x1U,
  kFLEXIO_TimerDecSrcOnPinInputShiftPinInput = 0x2U,
  kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput = 0x3U }
  *Define type of timer decrement.*
- enum flexio_timer_reset_condition_t {
  kFLEXIO_TimerResetNever = 0x0U,
  kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput = 0x2U,
  kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput = 0x3U,
  kFLEXIO_TimerResetOnTimerPinRisingEdge = 0x4U,
  kFLEXIO_TimerResetOnTimerTriggerRisingEdge = 0x6U,
  kFLEXIO_TimerResetOnTimerTriggerBothEdge = 0x7U }
  *Define type of timer reset condition.*
- enum flexio_timer_disable_condition_t {
  kFLEXIO_TimerDisableNever = 0x0U,
  kFLEXIO_TimerDisableOnPreTimerDisable = 0x1U,
  kFLEXIO_TimerDisableOnTimerCompare = 0x2U,
  kFLEXIO_TimerDisableOnTimerCompareTriggerLow = 0x3U,
  kFLEXIO_TimerDisableOnPinBothEdge = 0x4U,
  kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh = 0x5U,
  kFLEXIO_TimerDisableOnTriggerFallingEdge = 0x6U }
  *Define type of timer disable condition.*
- enum flexio_timer_enable_condition_t {
  kFLEXIO_TimerEnabledAlways = 0x0U,
  kFLEXIO_TimerEnableOnPrevTimerEnable = 0x1U,
  kFLEXIO_TimerEnableOnTriggerHigh = 0x2U,
  kFLEXIO_TimerEnableOnTriggerHighPinHigh = 0x3U,
  kFLEXIO_TimerEnableOnPinRisingEdge = 0x4U,
  kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh = 0x5U,
  kFLEXIO_TimerEnableOnTriggerRisingEdge = 0x6U,
  kFLEXIO_TimerEnableOnTriggerBothEdge = 0x7U }
  *Define type of timer enable condition.*
- enum flexio_timer_stop_bit_condition_t {

kFLEXIO_TimerStopBitDisabled = 0x0U,
kFLEXIO_TimerStopBitEnableOnTimerCompare = 0x1U,
kFLEXIO_TimerStopBitEnableOnTimerDisable = 0x2U,
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable = 0x3U }
*Define type of timer stop bit generate condition.*

- enum flexio_timer_start_bit_condition_t {
kFLEXIO_TimerStartBitDisabled = 0x0U,
kFLEXIO_TimerStartBitEnabled = 0x1U }
*Define type of timer start bit generate condition.*
- enum flexio_shifter_timer_polarity_t {
kFLEXIO_ShifterTimerPolarityOnPositive = 0x0U,
kFLEXIO_ShifterTimerPolarityOnNegitive = 0x1U }
*Define type of timer polarity for shifter control.*
- enum flexio_shifter_mode_t {
kFLEXIO_ShifterDisabled = 0x0U,
kFLEXIO_ShifterModeReceive = 0x1U,
kFLEXIO_ShifterModeTransmit = 0x2U,
kFLEXIO_ShifterModeMatchStore = 0x4U,
kFLEXIO_ShifterModeMatchContinuous = 0x5U }
*Define type of shifter working mode.*
- enum flexio_shifter_input_source_t {
kFLEXIO_ShifterInputFromPin = 0x0U,
kFLEXIO_ShifterInputFromNextShifterOutput = 0x1U }
*Define type of shifter input source.*
- enum flexio_shifter_stop_bit_t {
kFLEXIO_ShifterStopBitDisable = 0x0U,
kFLEXIO_ShifterStopBitLow = 0x2U,
kFLEXIO_ShifterStopBitHigh = 0x3U }
*Define of STOP bit configuration.*
- enum flexio_shifter_start_bit_t {
kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable = 0x0U,
kFLEXIO_ShifterStartBitDisabledLoadDataOnShift = 0x1U,
kFLEXIO_ShifterStartBitLow = 0x2U,
kFLEXIO_ShifterStartBitHigh = 0x3U }
*Define type of START bit configuration.*
- enum flexio_shifter_buffer_type_t {
kFLEXIO_ShifterBuffer = 0x0U,
kFLEXIO_ShifterBufferBitSwapped = 0x1U,
kFLEXIO_ShifterBufferByteSwapped = 0x2U,
kFLEXIO_ShifterBufferBitByteSwapped = 0x3U }
*Define FlexIO shifter buffer type.*

## Variables

- FLEXIO_Type *const s_flexioBases [ ]
*Pointers to flexio bases for each instance.*

- const clock_ip_name_t s_flexioClocks [ ]
  *Pointers to flexio clocks for each instance.*

### Driver version

- #define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *FlexIO driver version 2.0.2.*

### FlexIO Initialization and De-initialization

- void FLEXIO_GetDefaultConfig (flexio_config_t *userConfig)
  *Gets the default configuration to configure the FlexIO module.*
- void FLEXIO_Init (FLEXIO_Type *base, const flexio_config_t *userConfig)
  *Configures the FlexIO with a FlexIO configuration.*
- void FLEXIO_Deinit (FLEXIO_Type *base)
  *Gates the FlexIO clock.*
- uint32_t FLEXIO_GetInstance (FLEXIO_Type *base)
  *Get instance number for FLEXIO module.*

### FlexIO Basic Operation

- void FLEXIO_Reset (FLEXIO_Type *base)
  *Resets the FlexIO module.*
- static void FLEXIO_Enable (FLEXIO_Type *base, bool enable)
  *Enables the FlexIO module operation.*
- void FLEXIO_SetShifterConfig (FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
  *Configures the shifter with the shifter configuration.*
- void FLEXIO_SetTimerConfig (FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t *timerConfig)
  *Configures the timer with the timer configuration.*

### FlexIO Interrupt Operation

- static void FLEXIO_EnableShifterStatusInterrupts (FLEXIO_Type *base, uint32_t mask)
  *Enables the shifter status interrupt.*
- static void FLEXIO_DisableShifterStatusInterrupts (FLEXIO_Type *base, uint32_t mask)
  *Disables the shifter status interrupt.*
- static void FLEXIO_EnableShifterErrorInterrupts (FLEXIO_Type *base, uint32_t mask)
  *Enables the shifter error interrupt.*
- static void FLEXIO_DisableShifterErrorInterrupts (FLEXIO_Type *base, uint32_t mask)
  *Disables the shifter error interrupt.*
- static void FLEXIO_EnableTimerStatusInterrupts (FLEXIO_Type *base, uint32_t mask)
  *Enables the timer status interrupt.*
- static void FLEXIO_DisableTimerStatusInterrupts (FLEXIO_Type *base, uint32_t mask)

*Disables the timer status interrupt.*

### FlexIO Status Operation

- static uint32_t FLEXIO_GetShifterStatusFlags (FLEXIO_Type *base)
  *Gets the shifter status flags.*
- static void FLEXIO_ClearShifterStatusFlags (FLEXIO_Type *base, uint32_t mask)
  *Clears the shifter status flags.*
- static uint32_t FLEXIO_GetShifterErrorFlags (FLEXIO_Type *base)
  *Gets the shifter error flags.*
- static void FLEXIO_ClearShifterErrorFlags (FLEXIO_Type *base, uint32_t mask)
  *Clears the shifter error flags.*
- static uint32_t FLEXIO_GetTimerStatusFlags (FLEXIO_Type *base)
  *Gets the timer status flags.*
- static void FLEXIO_ClearTimerStatusFlags (FLEXIO_Type *base, uint32_t mask)
  *Clears the timer status flags.*

### FlexIO DMA Operation

- static void FLEXIO_EnableShifterStatusDMA (FLEXIO_Type *base, uint32_t mask, bool enable)
  *Enables/disables the shifter status DMA.*
- uint32_t FLEXIO_GetShifterBufferAddress (FLEXIO_Type *base, flexio_shifter_buffer_type_-t type, uint8_t index)
  *Gets the shifter buffer address for the DMA transfer usage.*
- status_t FLEXIO_RegisterHandleIRQ (void *base, void *handle, flexio_isr_t isr)
  *Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.*
- status_t FLEXIO_UnregisterHandleIRQ (void *base)
  *Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.*

## 23.2.2 Data Structure Documentation

### 23.2.2.1 struct flexio_config_t

### Data Fields

- bool enableFlexio
  *Enable/disable FlexIO module.*
- bool enableInDoze
  *Enable/disable FlexIO operation in doze mode.*
- bool enableInDebug
  *Enable/disable FlexIO operation in debug mode.*
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers, fast access requires` *the FlexIO clock to be at least twice the frequency of the bus clock.*

#### 23.2.2.1.0.37 Field Documentation

#### 23.2.2.1.0.37.1 bool flexio_config_t::enableFastAccess

### 23.2.2.2 struct flexio_timer_config_t

## Data Fields

- uint32_t triggerSelect
  *The internal trigger selection number using MACROs.*
- flexio_timer_trigger_polarity_t triggerPolarity
  *Trigger Polarity.*
- flexio_timer_trigger_source_t triggerSource
  *Trigger Source, internal (see 'trgsel') or external.*
- flexio_pin_config_t pinConfig
  *Timer Pin Configuration.*
- uint32_t pinSelect
  *Timer Pin number Select.*
- flexio_pin_polarity_t pinPolarity
  *Timer Pin Polarity.*
- flexio_timer_mode_t timerMode
  *Timer work Mode.*
- flexio_timer_output_t timerOutput
  `Configures the initial state of the Timer Output and` *whether it is affected by the Timer reset.*
- flexio_timer_decrement_source_t timerDecrement
  `Configures the source of the Timer decrement and the` *source of the Shift clock.*
- flexio_timer_reset_condition_t timerReset
  `Configures the condition that causes the timer counter` *(and optionally the timer output) to be reset.*
- flexio_timer_disable_condition_t timerDisable
  `Configures the condition that causes the Timer to be` *disabled and stop decrementing.*
- flexio_timer_enable_condition_t timerEnable
  `Configures the condition that causes the Timer to be` *enabled and start decrementing.*
- flexio_timer_stop_bit_condition_t timerStop
  *Timer STOP Bit generation.*
- flexio_timer_start_bit_condition_t timerStart
  *Timer STRAT Bit generation.*
- uint32_t timerCompare
  *Value for Timer Compare N Register.*

#### 23.2.2.2.0.38 Field Documentation

##### 23.2.2.2.0.38.1 uint32_t flexio_timer_config_t::triggerSelect

##### 23.2.2.2.0.38.2 flexio_timer_trigger_polarity_t flexio_timer_config_t::triggerPolarity

##### 23.2.2.2.0.38.3 flexio_timer_trigger_source_t flexio_timer_config_t::triggerSource

##### 23.2.2.2.0.38.4 flexio_pin_config_t flexio_timer_config_t::pinConfig

##### 23.2.2.2.0.38.5 uint32_t flexio_timer_config_t::pinSelect

##### 23.2.2.2.0.38.6 flexio_pin_polarity_t flexio_timer_config_t::pinPolarity

##### 23.2.2.2.0.38.7 flexio_timer_mode_t flexio_timer_config_t::timerMode

##### 23.2.2.2.0.38.8 flexio_timer_output_t flexio_timer_config_t::timerOutput

##### 23.2.2.2.0.38.9 flexio_timer_decrement_source_t flexio_timer_config_t::timerDecrement

##### 23.2.2.2.0.38.10 flexio_timer_reset_condition_t flexio_timer_config_t::timerReset

##### 23.2.2.2.0.38.11 flexio_timer_disable_condition_t flexio_timer_config_t::timerDisable

##### 23.2.2.2.0.38.12 flexio_timer_enable_condition_t flexio_timer_config_t::timerEnable

##### 23.2.2.2.0.38.13 flexio_timer_stop_bit_condition_t flexio_timer_config_t::timerStop

##### 23.2.2.2.0.38.14 flexio_timer_start_bit_condition_t flexio_timer_config_t::timerStart

##### 23.2.2.2.0.38.15 uint32_t flexio_timer_config_t::timerCompare

### 23.2.2.3 struct flexio_shifter_config_t

### Data Fields

- uint32_t timerSelect
  `Selects which Timer is used for controlling the`
  *logic/shift register and generating the Shift clock.*
- flexio_shifter_timer_polarity_t timerPolarity
  *Timer Polarity.*
- flexio_pin_config_t pinConfig
  *Shifter Pin Configuration.*
- uint32_t pinSelect
  *Shifter Pin number Select.*
- flexio_pin_polarity_t pinPolarity
  *Shifter Pin Polarity.*
- flexio_shifter_mode_t shifterMode
  *Configures the mode of the Shifter.*
- flexio_shifter_input_source_t inputSource
  *Selects the input source for the shifter.*

- flexio_shifter_stop_bit_t shifterStop
  *Shifter STOP bit.*
- flexio_shifter_start_bit_t shifterStart
  *Shifter START bit.*

#### 23.2.2.3.0.39 Field Documentation

##### 23.2.2.3.0.39.1 uint32_t flexio_shifter_config_t::timerSelect

##### 23.2.2.3.0.39.2 flexio_shifter_timer_polarity_t flexio_shifter_config_t::timerPolarity

##### 23.2.2.3.0.39.3 flexio_pin_config_t flexio_shifter_config_t::pinConfig

##### 23.2.2.3.0.39.4 uint32_t flexio_shifter_config_t::pinSelect

##### 23.2.2.3.0.39.5 flexio_pin_polarity_t flexio_shifter_config_t::pinPolarity

##### 23.2.2.3.0.39.6 flexio_shifter_mode_t flexio_shifter_config_t::shifterMode

##### 23.2.2.3.0.39.7 flexio_shifter_input_source_t flexio_shifter_config_t::inputSource

##### 23.2.2.3.0.39.8 flexio_shifter_stop_bit_t flexio_shifter_config_t::shifterStop

##### 23.2.2.3.0.39.9 flexio_shifter_start_bit_t flexio_shifter_config_t::shifterStart

### 23.2.3 Macro Definition Documentation

#### 23.2.3.1 #define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

#### 23.2.3.2 #define FLEXIO_TIMER_TRIGGER_SEL_PININPUT( *x* ) ((uint32_t)(x) << 1U)

### 23.2.4 Typedef Documentation

#### 23.2.4.1 typedef void(* flexio_isr_t)(void *base, void *handle)

### 23.2.5 Enumeration Type Documentation

#### 23.2.5.1 enum flexio_timer_trigger_polarity_t

Enumerator

***kFLEXIO_TimerTriggerPolarityActiveHigh*** Active high.
***kFLEXIO_TimerTriggerPolarityActiveLow*** Active low.

#### 23.2.5.2 enum flexio_timer_trigger_source_t

Enumerator

***kFLEXIO_TimerTriggerSourceExternal*** External trigger selected.
***kFLEXIO_TimerTriggerSourceInternal*** Internal trigger selected.

#### 23.2.5.3 enum flexio_pin_config_t

Enumerator

***kFLEXIO_PinConfigOutputDisabled*** Pin output disabled.
***kFLEXIO_PinConfigOpenDrainOrBidirection*** Pin open drain or bidirectional output enable.
***kFLEXIO_PinConfigBidirectionOutputData*** Pin bidirectional output data.
***kFLEXIO_PinConfigOutput*** Pin output.

#### 23.2.5.4 enum flexio_pin_polarity_t

Enumerator

***kFLEXIO_PinActiveHigh*** Active high.
***kFLEXIO_PinActiveLow*** Active low.

#### 23.2.5.5 enum flexio_timer_mode_t

Enumerator

***kFLEXIO_TimerModeDisabled*** Timer Disabled.
***kFLEXIO_TimerModeDual8BitBaudBit*** Dual 8-bit counters baud/bit mode.
***kFLEXIO_TimerModeDual8BitPWM*** Dual 8-bit counters PWM mode.
***kFLEXIO_TimerModeSingle16Bit*** Single 16-bit counter mode.

#### 23.2.5.6 enum flexio_timer_output_t

Enumerator

***kFLEXIO_TimerOutputOneNotAffectedByReset*** Logic one when enabled and is not affected by timer reset.
***kFLEXIO_TimerOutputZeroNotAffectedByReset*** Logic zero when enabled and is not affected by timer reset.
***kFLEXIO_TimerOutputOneAffectedByReset*** Logic one when enabled and on timer reset.
***kFLEXIO_TimerOutputZeroAffectedByReset*** Logic zero when enabled and on timer reset.

#### 23.2.5.7 enum flexio_timer_decrement_source_t

Enumerator

***kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput*** Decrement counter on FlexIO clock, Shift clock equals Timer output.

***kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput*** Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

***kFLEXIO_TimerDecSrcOnPinInputShiftPinInput*** Decrement counter on Pin input (both edges), Shift clock equals Pin input.

***kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput*** Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

#### 23.2.5.8 enum flexio_timer_reset_condition_t

Enumerator

***kFLEXIO_TimerResetNever*** Timer never reset.

***kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput*** Timer reset on Timer Pin equal to Timer Output.

***kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput*** Timer reset on Timer Trigger equal to Timer Output.

***kFLEXIO_TimerResetOnTimerPinRisingEdge*** Timer reset on Timer Pin rising edge.

***kFLEXIO_TimerResetOnTimerTriggerRisingEdge*** Timer reset on Trigger rising edge.

***kFLEXIO_TimerResetOnTimerTriggerBothEdge*** Timer reset on Trigger rising or falling edge.

#### 23.2.5.9 enum flexio_timer_disable_condition_t

Enumerator

***kFLEXIO_TimerDisableNever*** Timer never disabled.

***kFLEXIO_TimerDisableOnPreTimerDisable*** Timer disabled on Timer N-1 disable.

***kFLEXIO_TimerDisableOnTimerCompare*** Timer disabled on Timer compare.

***kFLEXIO_TimerDisableOnTimerCompareTriggerLow*** Timer disabled on Timer compare and Trigger Low.

***kFLEXIO_TimerDisableOnPinBothEdge*** Timer disabled on Pin rising or falling edge.

***kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh*** Timer disabled on Pin rising or falling edge provided Trigger is high.

***kFLEXIO_TimerDisableOnTriggerFallingEdge*** Timer disabled on Trigger falling edge.

#### 23.2.5.10 enum flexio_timer_enable_condition_t

Enumerator

***kFLEXIO_TimerEnabledAlways*** Timer always enabled.

*kFLEXIO_TimerEnableOnPrevTimerEnable* Timer enabled on Timer N-1 enable.
*kFLEXIO_TimerEnableOnTriggerHigh* Timer enabled on Trigger high.
*kFLEXIO_TimerEnableOnTriggerHighPinHigh* Timer enabled on Trigger high and Pin high.
*kFLEXIO_TimerEnableOnPinRisingEdge* Timer enabled on Pin rising edge.
*kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh* Timer enabled on Pin rising edge and Trigger high.
*kFLEXIO_TimerEnableOnTriggerRisingEdge* Timer enabled on Trigger rising edge.
*kFLEXIO_TimerEnableOnTriggerBothEdge* Timer enabled on Trigger rising or falling edge.

#### 23.2.5.11 enum flexio_timer_stop_bit_condition_t

Enumerator

*kFLEXIO_TimerStopBitDisabled* Stop bit disabled.
*kFLEXIO_TimerStopBitEnableOnTimerCompare* Stop bit is enabled on timer compare.
*kFLEXIO_TimerStopBitEnableOnTimerDisable* Stop bit is enabled on timer disable.
*kFLEXIO_TimerStopBitEnableOnTimerCompareDisable* Stop bit is enabled on timer compare and timer disable.

#### 23.2.5.12 enum flexio_timer_start_bit_condition_t

Enumerator

*kFLEXIO_TimerStartBitDisabled* Start bit disabled.
*kFLEXIO_TimerStartBitEnabled* Start bit enabled.

#### 23.2.5.13 enum flexio_shifter_timer_polarity_t

Enumerator

*kFLEXIO_ShifterTimerPolarityOnPositive* Shift on positive edge of shift clock.
*kFLEXIO_ShifterTimerPolarityOnNegitive* Shift on negative edge of shift clock.

#### 23.2.5.14 enum flexio_shifter_mode_t

Enumerator

*kFLEXIO_ShifterDisabled* Shifter is disabled.
*kFLEXIO_ShifterModeReceive* Receive mode.
*kFLEXIO_ShifterModeTransmit* Transmit mode.
*kFLEXIO_ShifterModeMatchStore* Match store mode.
*kFLEXIO_ShifterModeMatchContinuous* Match continuous mode.

#### 23.2.5.15 enum flexio_shifter_input_source_t

Enumerator

***kFLEXIO_ShifterInputFromPin*** Shifter input from pin.
***kFLEXIO_ShifterInputFromNextShifterOutput*** Shifter input from Shifter N+1.

#### 23.2.5.16 enum flexio_shifter_stop_bit_t

Enumerator

***kFLEXIO_ShifterStopBitDisable*** Disable shifter stop bit.
***kFLEXIO_ShifterStopBitLow*** Set shifter stop bit to logic low level.
***kFLEXIO_ShifterStopBitHigh*** Set shifter stop bit to logic high level.

#### 23.2.5.17 enum flexio_shifter_start_bit_t

Enumerator

***kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable*** Disable shifter start bit, transmitter loads data on enable.
***kFLEXIO_ShifterStartBitDisabledLoadDataOnShift*** Disable shifter start bit, transmitter loads data on first shift.
***kFLEXIO_ShifterStartBitLow*** Set shifter start bit to logic low level.
***kFLEXIO_ShifterStartBitHigh*** Set shifter start bit to logic high level.

#### 23.2.5.18 enum flexio_shifter_buffer_type_t

Enumerator

***kFLEXIO_ShifterBuffer*** Shifter Buffer N Register.
***kFLEXIO_ShifterBufferBitSwapped*** Shifter Buffer N Bit Byte Swapped Register.
***kFLEXIO_ShifterBufferByteSwapped*** Shifter Buffer N Byte Swapped Register.
***kFLEXIO_ShifterBufferBitByteSwapped*** Shifter Buffer N Bit Swapped Register.

### 23.2.6 Function Documentation

#### 23.2.6.1 void FLEXIO_GetDefaultConfig ( flexio_config_t * *userConfig* )

The configuration can used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

| | |
|---|---|
| *userConfig* | pointer to flexio_config_t structure |

### 23.2.6.2 void FLEXIO_Init ( FLEXIO_Type * *base*, const flexio_config_t * *userConfig* )

The configuration structure can be filled by the user or be set with default values by FLEXIO_GetDefaultConfig().

Example

```
flexio_config_t config = {
.enableFlexio = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *userConfig* | pointer to flexio_config_t structure |

### 23.2.6.3 void FLEXIO_Deinit ( FLEXIO_Type * *base* )

Call this API to stop the FlexIO clock.

Note

After calling this API, call the FLEXO_Init to use the FlexIO module.

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |

### 23.2.6.4 uint32_t FLEXIO_GetInstance ( FLEXIO_Type * *base* )

Parameters

| | |
|---|---|
| *base* | FLEXIO peripheral base address. |

#### 23.2.6.5 void FLEXIO_Reset ( FLEXIO_Type * *base* )

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |

#### 23.2.6.6 static void FLEXIO_Enable ( FLEXIO_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *enable* | true to enable, false to disable. |

#### 23.2.6.7 void FLEXIO_SetShifterConfig ( FLEXIO_Type * *base,* uint8_t *index,* const flexio_shifter_config_t * *shifterConfig* )

The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
.timerSelect = 0,
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *index* | Shifter index |
| *shifterConfig* | Pointer to flexio_shifter_config_t structure |

### 23.2.6.8 void FLEXIO_SetTimerConfig ( FLEXIO_Type * *base*, uint8_t *index*, const flexio_timer_config_t * *timerConfig* )

The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
      ,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *index* | Timer index |
| *timerConfig* | Pointer to the flexio_timer_config_t structure |

### 23.2.6.9 static void FLEXIO_EnableShifterStatusInterrupts ( FLEXIO_Type * *base*, uint32_t *mask* ) [inline], [static]

The interrupt generates when the corresponding SSF is set.

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The shifter status mask which can be calculated by (1 << shifter index) |

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

#### 23.2.6.10 static void FLEXIO_DisableShifterStatusInterrupts ( FLEXIO_Type * *base*, uint32_t *mask* ) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The shifter status mask which can be calculated by (1 << shifter index) |

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

#### 23.2.6.11 static void FLEXIO_EnableShifterErrorInterrupts ( FLEXIO_Type * *base*, uint32_t *mask* ) [inline], [static]

The interrupt generates when the corresponding SEF is set.

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The shifter error mask which can be calculated by (1 << shifter index) |

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

#### 23.2.6.12 static void FLEXIO_DisableShifterErrorInterrupts ( FLEXIO_Type * *base,* uint32_t *mask* ) [inline], [static]

The interrupt won't generate when the corresponding SEF is set.

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The shifter error mask which can be calculated by (1 << shifter index) |

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

#### 23.2.6.13 static void FLEXIO_EnableTimerStatusInterrupts ( FLEXIO_Type * *base,* uint32_t *mask* ) [inline], [static]

The interrupt generates when the corresponding SSF is set.

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The timer status mask which can be calculated by (1 << timer index) |

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 << timer index0) | (1 << timer index1))

#### 23.2.6.14 static void FLEXIO_DisableTimerStatusInterrupts ( FLEXIO_Type * *base,* uint32_t *mask* ) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The timer status mask which can be calculated by (1 << timer index) |

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 << timer index0) | (1 << timer index1))

#### 23.2.6.15 static uint32_t FLEXIO_GetShifterStatusFlags ( FLEXIO_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |

Returns

Shifter status flags

#### 23.2.6.16 static void FLEXIO_ClearShifterStatusFlags ( FLEXIO_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The shifter status mask which can be calculated by (1 << shifter index) |

Note

For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

#### 23.2.6.17 static uint32_t FLEXIO_GetShifterErrorFlags ( FLEXIO_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |

Returns

Shifter error flags

#### 23.2.6.18 static void FLEXIO_ClearShifterErrorFlags ( FLEXIO_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The shifter error mask which can be calculated by (1 << shifter index) |

Note

For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

#### 23.2.6.19 static uint32_t FLEXIO_GetTimerStatusFlags ( FLEXIO_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |

Returns

Timer status flags

#### 23.2.6.20 static void FLEXIO_ClearTimerStatusFlags ( FLEXIO_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The timer status mask which can be calculated by (1 << timer index) |

Note

For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using ((1 << timer index0) | (1 << timer index1))

#### 23.2.6.21 static void FLEXIO_EnableShifterStatusDMA ( FLEXIO_Type * *base,* uint32_t *mask,* bool *enable* ) [inline], [static]

The DMA request generates when the corresponding SSF is set.

Note

For multiple shifter status DMA enables, for example, calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *mask* | The shifter status mask which can be calculated by (1 << shifter index) |
| *enable* | True to enable, false to disable. |

#### 23.2.6.22 uint32_t FLEXIO_GetShifterBufferAddress ( FLEXIO_Type * *base*, flexio_shifter_buffer_type_t *type*, uint8_t *index* )

Parameters

| | |
|---|---|
| *base* | FlexIO peripheral base address |
| *type* | Shifter type of flexio_shifter_buffer_type_t |
| *index* | Shifter index |

Returns

Corresponding shifter buffer index

#### 23.2.6.23 status_t FLEXIO_RegisterHandleIRQ ( void * *base*, void * *handle*, flexio_isr_t *isr* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FlexIO simulated peripheral type. |
| *handle* | Pointer to the handler for FlexIO simulated peripheral. |
| *isr* | FlexIO simulated peripheral interrupt handler. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO type/handle/ISR table out of range. |

#### 23.2.6.24 status_t FLEXIO_UnregisterHandleIRQ ( void * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FlexIO simulated peripheral type. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO type/handle/ISR table out of range. |

### 23.2.7 Variable Documentation

#### 23.2.7.1 FLEXIO_Type* const s_flexioBases[]

#### 23.2.7.2 const clock_ip_name_t s_flexioClocks[]

## 23.3 FlexIO Camera Driver

### 23.3.1 Overview

The MCUXpresso SDK provides a driver for the camera function using Flexible I/O.

FlexIO Camera driver includes functional APIs and eDMA transactional APIs. Functional APIs target low level APIs. Users can use functional APIs for FlexIO Camera initialization/configuration/operation purpose. Using the functional API requires knowledge of the FlexIO Camera peripheral and how to organize functional APIs to meet the requirements of the application. All functional API use the FLEXIO_CAMERA_Type * as the first parameter. FlexIO Camera functional operation groups provide the functional APIs set.

eDMA transactional APIs target high-level APIs. Users can use the transactional API to enable the peripheral quickly and can also use in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the flexio_camera_edma_handle_t as the second parameter. Users need to initialize the handle by calling the FLEXIO_CAMERA_TransferCreateHandleEDMA() API.

eDMA transactional APIs support asynchronous receive. This means that the functions FLEXIO_CAMERA_TransferReceiveEDMA() set up an interrupt for data receive. When the receive is complete, the upper layer is notified through a callback function with the status kStatus_FLEXIO_CAMERA_RxIdle.

### 23.3.2 Typical use case

#### 23.3.2.1 FlexIO Camera Receive using eDMA method

```
volatile uint32_t isEDMAGetOnePictureFinish = false;
edma_handle_t g_edmaHandle;
flexio_camera_edma_handle_t g_cameraEdmaHandle;
edma_config_t edmaConfig;
FLEXIO_CAMERA_Type g_FlexioCameraDevice = {.flexioBase = FLEXIO0,
                                           .datPinStartIdx = 24U, /* fxio_pin 24 -31 are used. */
                                           .pclkPinIdx = 1U,      /* fxio_pin 1 is used as pclk pin. */
                                           .hrefPinIdx = 18U,     /* flexio_pin 18 is used as href pin. */
                                           .shifterStartIdx = 0U, /* Shifter 0 = 7 are used. */
                                           .shifterCount = 8U,
                                           .timerIdx = 0U};
flexio_camera_config_t cameraConfig;

/* Configure DMAMUX */
DMAMUX_Init(DMAMUX0);
/* Configure DMA */
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(DMA0, &edmaConfig);

DMAMUX_SetSource(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF, (g_FlexioCameraDevice.
      shifterStartIdx + 1U));
DMAMUX_EnableChannel(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);
EDMA_CreateHandle(&g_edmaHandle, DMA0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);

FLEXIO_CAMERA_GetDefaultConfig(&cameraConfig);
FLEXIO_CAMERA_Init(&g_FlexioCameraDevice, &cameraConfig);
/* Clear all the flag. */
```

```
FLEXIO_CAMERA_ClearStatusFlags(&g_FlexioCameraDevice,
                               kFLEXIO_CAMERA_RxDataRegFullFlag |
      kFLEXIO_CAMERA_RxErrorFlag);
FLEXIO_ClearTimerStatusFlags(FLEXIO0, 0xFF);
FLEXIO_CAMERA_TransferCreateHandleEDMA(&g_FlexioCameraDevice, &
      g_cameraEdmaHandle, FLEXIO_CAMERA_UserCallback, NULL,
                               &g_edmaHandle);
cameraTransfer.dataAddress = (uint32_t)u16CameraFrameBuffer;
cameraTransfer.dataNum = sizeof(u16CameraFrameBuffer);
FLEXIO_CAMERA_TransferReceiveEDMA(&g_FlexioCameraDevice, &
      g_cameraEdmaHandle, &cameraTransfer);
while (!(isEDMAGetOnePictureFinish))
{
    ;
}

/* A callback function is also needed */
void FLEXIO_CAMERA_UserCallback(FLEXIO_CAMERA_Type *base,
                            flexio_camera_edma_handle_t *handle,
                            status_t status,
                            void *userData)
{
    userData = userData;
    /* eDMA Transfer finished */
    if (kStatus_FLEXIO_CAMERA_RxIdle == status)
    {
        isEDMAGetOnePictureFinish = true;
    }
}
```

## Modules

- FlexIO eDMA Camera Driver

## Data Structures

- struct FLEXIO_CAMERA_Type
  *Define structure of configuring the FlexIO Camera device. More...*
- struct flexio_camera_config_t
  *Define FlexIO Camera user configuration structure. More...*
- struct flexio_camera_transfer_t
  *Define FlexIO Camera transfer structure. More...*

## Macros

- #define FLEXIO_CAMERA_PARALLEL_DATA_WIDTH (8U)
  *Define the Camera CPI interface is constantly 8-bit width.*

## Enumerations

- enum _flexio_camera_status {
  kStatus_FLEXIO_CAMERA_RxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_CAMERA,

0),
kStatus_FLEXIO_CAMERA_RxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_CAMERA, 1)
}
*Error codes for the Camera driver.*

- enum _flexio_camera_status_flags {
kFLEXIO_CAMERA_RxDataRegFullFlag = 0x1U,
kFLEXIO_CAMERA_RxErrorFlag = 0x2U }
*Define FlexIO Camera status mask.*

## Driver version

- #define FSL_FLEXIO_CAMERA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
*FlexIO Camera driver version 2.1.2.*

## Initialization and configuration

- void FLEXIO_CAMERA_Init (FLEXIO_CAMERA_Type *base, const flexio_camera_config_t *config)
*Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO Camera.*
- void FLEXIO_CAMERA_Deinit (FLEXIO_CAMERA_Type *base)
*Resets the FLEXIO_CAMERA shifer and timer config.*
- void FLEXIO_CAMERA_GetDefaultConfig (flexio_camera_config_t *config)
*Gets the default configuration to configure the FlexIO Camera.*
- static void FLEXIO_CAMERA_Enable (FLEXIO_CAMERA_Type *base, bool enable)
*Enables/disables the FlexIO Camera module operation.*

## Status

- uint32_t FLEXIO_CAMERA_GetStatusFlags (FLEXIO_CAMERA_Type *base)
*Gets the FlexIO Camera status flags.*
- void FLEXIO_CAMERA_ClearStatusFlags (FLEXIO_CAMERA_Type *base, uint32_t mask)
*Clears the receive buffer full flag manually.*

## Interrupts

- void FLEXIO_CAMERA_EnableInterrupt (FLEXIO_CAMERA_Type *base)
*Switches on the interrupt for receive buffer full event.*
- void FLEXIO_CAMERA_DisableInterrupt (FLEXIO_CAMERA_Type *base)
*Switches off the interrupt for receive buffer full event.*

## DMA support

- static void FLEXIO_CAMERA_EnableRxDMA (FLEXIO_CAMERA_Type *base, bool enable)

*Enables/disables the FlexIO Camera receive DMA.*
- static uint32_t FLEXIO_CAMERA_GetRxBufferAddress (FLEXIO_CAMERA_Type *base)
  *Gets the data from the receive buffer.*

## 23.3.3 Data Structure Documentation

### 23.3.3.1 struct FLEXIO_CAMERA_Type

#### Data Fields

- FLEXIO_Type * flexioBase
  *FlexIO module base address.*
- uint32_t datPinStartIdx
  *First data pin (D0) index for flexio_camera.*
- uint32_t pclkPinIdx
  *Pixel clock pin (PCLK) index for flexio_camera.*
- uint32_t hrefPinIdx
  *Horizontal sync pin (HREF) index for flexio_camera.*
- uint32_t shifterStartIdx
  *First shifter index used for flexio_camera data FIFO.*
- uint32_t shifterCount
  *The count of shifters that are used as flexio_camera data FIFO.*
- uint32_t timerIdx
  *Timer index used for flexio_camera in FlexIO.*

#### 23.3.3.1.0.40 Field Documentation

#### 23.3.3.1.0.40.1 FLEXIO_Type* FLEXIO_CAMERA_Type::flexioBase

#### 23.3.3.1.0.40.2 uint32_t FLEXIO_CAMERA_Type::datPinStartIdx

Then the successive following FLEXIO_CAMERA_DATA_WIDTH-1 pins are used as D1-D7.

#### 23.3.3.1.0.40.3 uint32_t FLEXIO_CAMERA_Type::pclkPinIdx

#### 23.3.3.1.0.40.4 uint32_t FLEXIO_CAMERA_Type::hrefPinIdx

#### 23.3.3.1.0.40.5 uint32_t FLEXIO_CAMERA_Type::shifterStartIdx

#### 23.3.3.1.0.40.6 uint32_t FLEXIO_CAMERA_Type::shifterCount

#### 23.3.3.1.0.40.7 uint32_t FLEXIO_CAMERA_Type::timerIdx

### 23.3.3.2 struct flexio_camera_config_t

#### Data Fields

- bool enablecamera
  *Enable/disable FlexIO Camera TX & RX.*

- bool enableInDoze
  *Enable/disable FlexIO operation in doze mode.*
- bool enableInDebug
  *Enable/disable FlexIO operation in debug mode.*
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  *fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

#### 23.3.3.2.0.41 Field Documentation

#### 23.3.3.2.0.41.1 bool flexio_camera_config_t::enablecamera

#### 23.3.3.2.0.41.2 bool flexio_camera_config_t::enableFastAccess

### 23.3.3.3 struct flexio_camera_transfer_t

### Data Fields

- uint32_t dataAddress
  *Transfer buffer.*
- uint32_t dataNum
  *Transfer num.*

## 23.3.4 Macro Definition Documentation

### 23.3.4.1 #define FSL_FLEXIO_CAMERA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

### 23.3.4.2 #define FLEXIO_CAMERA_PARALLEL_DATA_WIDTH (8U)

## 23.3.5 Enumeration Type Documentation

### 23.3.5.1 enum _flexio_camera_status

Enumerator

***kStatus_FLEXIO_CAMERA_RxBusy*** Receiver is busy.
***kStatus_FLEXIO_CAMERA_RxIdle*** Camera receiver is idle.

### 23.3.5.2 enum _flexio_camera_status_flags

Enumerator

***kFLEXIO_CAMERA_RxDataRegFullFlag*** Receive buffer full flag.
***kFLEXIO_CAMERA_RxErrorFlag*** Receive buffer error flag.

### 23.3.6 Function Documentation

#### 23.3.6.1 void FLEXIO_CAMERA_Init ( FLEXIO_CAMERA_Type * *base,* const flexio_camera_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_CAMERA_Type structure |
| *config* | Pointer to flexio_camera_config_t structure |

#### 23.3.6.2 void FLEXIO_CAMERA_Deinit ( FLEXIO_CAMERA_Type * *base* )

Note

After calling this API, call FLEXO_CAMERA_Init to use the FlexIO Camera module.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_CAMERA_Type structure |

#### 23.3.6.3 void FLEXIO_CAMERA_GetDefaultConfig ( flexio_camera_config_t * *config* )

The configuration can be used directly for calling the FLEXIO_CAMERA_Init(). Example:

```
flexio_camera_config_t config;
FLEXIO_CAMERA_GetDefaultConfig(&userConfig);
```

Parameters

| | |
|---|---|
| *config* | Pointer to the flexio_camera_config_t structure |

#### 23.3.6.4 static void FLEXIO_CAMERA_Enable ( FLEXIO_CAMERA_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_CAMERA_Type |
| *enable* | True to enable, false does not have any effect. |

#### 23.3.6.5 uint32_t FLEXIO_CAMERA_GetStatusFlags ( FLEXIO_CAMERA_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_CAMERA_Type structure |

Returns

FlexIO shifter status flags
- FLEXIO_SHIFTSTAT_SSF_MASK
- 0

#### 23.3.6.6 void FLEXIO_CAMERA_ClearStatusFlags ( FLEXIO_CAMERA_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Pointer to the device. |
| *mask* | status flag The parameter can be any combination of the following values:<br>• kFLEXIO_CAMERA_RxDataRegFullFlag<br>• kFLEXIO_CAMERA_RxErrorFlag |

#### 23.3.6.7 void FLEXIO_CAMERA_EnableInterrupt ( FLEXIO_CAMERA_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the device. |

#### 23.3.6.8 void FLEXIO_CAMERA_DisableInterrupt ( FLEXIO_CAMERA_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the device. |

#### 23.3.6.9 static void FLEXIO_CAMERA_EnableRxDMA ( FLEXIO_CAMERA_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_CAMERA_Type structure |
| *enable* | True to enable, false to disable. |

The FlexIO Camera mode can't work without the DMA or eDMA support, Usually, it needs at least two DMA or eDMA channels, one for transferring data from Camera, such as 0V7670 to FlexIO buffer, another is for transferring data from FlexIO buffer to LCD.

#### 23.3.6.10 static uint32_t FLEXIO_CAMERA_GetRxBufferAddress ( FLEXIO_CAMERA_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the device. |

Returns

data Pointer to the buffer that keeps the data with count of base->shifterCount .

### 23.3.7 FlexIO eDMA Camera Driver

#### 23.3.7.1 Overview

#### Data Structures

- struct flexio_camera_edma_handle_t
  *Camera eDMA handle. More...*

#### Typedefs

- typedef void(∗ flexio_camera_edma_transfer_callback_t )(FLEXIO_CAMERA_Type ∗base, flexio_camera_edma_handle_t ∗handle, status_t status, void ∗userData)
  *Camera transfer callback function.*

#### Driver version

- #define FSL_FLEXIO_CAMERA_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  *FlexIO Camera EDMA driver version 2.1.2.*

#### eDMA transactional

- status_t FLEXIO_CAMERA_TransferCreateHandleEDMA (FLEXIO_CAMERA_Type ∗base, flexio_camera_edma_handle_t ∗handle, flexio_camera_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗rxEdmaHandle)
  *Initializes the Camera handle, which is used in transactional functions.*
- status_t FLEXIO_CAMERA_TransferReceiveEDMA (FLEXIO_CAMERA_Type ∗base, flexio_camera_edma_handle_t ∗handle, flexio_camera_transfer_t ∗xfer)
  *Receives data using eDMA.*
- void FLEXIO_CAMERA_TransferAbortReceiveEDMA (FLEXIO_CAMERA_Type ∗base, flexio_camera_edma_handle_t ∗handle)
  *Aborts the receive data which used the eDMA.*
- status_t FLEXIO_CAMERA_TransferGetReceiveCountEDMA (FLEXIO_CAMERA_Type ∗base, flexio_camera_edma_handle_t ∗handle, size_t ∗count)
  *Gets the remaining bytes to be received.*

#### 23.3.7.2 Data Structure Documentation

##### 23.3.7.2.1 struct _flexio_camera_edma_handle

Forward declaration of the handle typedef.

#### Data Fields

- flexio_camera_edma_transfer_callback_t callback

*Callback function.*

- void * userData
  *Camera callback function parameter.*
- size_t rxSize
  *Total bytes to be received.*
- edma_handle_t * rxEdmaHandle
  *The eDMA RX channel used.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- volatile uint8_t rxState
  *RX transfer state.*

#### 23.3.7.2.1.1 Field Documentation

##### 23.3.7.2.1.1.1 flexio_camera_edma_transfer_callback_t flexio_camera_edma_handle_t::callback

##### 23.3.7.2.1.1.2 void* flexio_camera_edma_handle_t::userData

##### 23.3.7.2.1.1.3 size_t flexio_camera_edma_handle_t::rxSize

##### 23.3.7.2.1.1.4 edma_handle_t* flexio_camera_edma_handle_t::rxEdmaHandle

##### 23.3.7.2.1.1.5 uint8_t flexio_camera_edma_handle_t::nbytes

### 23.3.7.3 Macro Definition Documentation

#### 23.3.7.3.1 #define FSL_FLEXIO_CAMERA_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

### 23.3.7.4 Typedef Documentation

#### 23.3.7.4.1 typedef void(* flexio_camera_edma_transfer_callback_t)(FLEXIO_CAMERA_Type *base, flexio_camera_edma_handle_t *handle, status_t status, void *userData)

### 23.3.7.5 Function Documentation

#### 23.3.7.5.1 status_t FLEXIO_CAMERA_TransferCreateHandleEDMA ( FLEXIO_CAMERA_Type * *base*, flexio_camera_edma_handle_t * *handle*, flexio_camera_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *rxEdmaHandle* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_CAMERA_Type. |

| | |
|---|---|
| *handle* | Pointer to flexio_camera_edma_handle_t structure. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |
| *rxEdmaHandle* | User requested DMA handle for RX DMA transfer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO Camera eDMA type/handle table out of range. |

#### 23.3.7.5.2 status_t FLEXIO_CAMERA_TransferReceiveEDMA ( FLEXIO_CAMERA_Type * *base*, flexio_camera_edma_handle_t * *handle*, flexio_camera_transfer_t * *xfer* )

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_CAMERA_Type. |
| *handle* | Pointer to the flexio_camera_edma_handle_t structure. |
| *xfer* | Camera eDMA transfer structure, see flexio_camera_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeeded, others failed. |
| *kStatus_CAMERA_RxBusy* | Previous transfer on going. |

#### 23.3.7.5.3 void FLEXIO_CAMERA_TransferAbortReceiveEDMA ( FLEXIO_CAMERA_Type * *base*, flexio_camera_edma_handle_t * *handle* )

This function aborts the receive data which used the eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_CAMERA_Type. |

| | |
|---|---|
| *handle* | Pointer to the flexio_camera_edma_handle_t structure. |

#### 23.3.7.5.4 status_t FLEXIO_CAMERA_TransferGetReceiveCountEDMA ( FLEXIO_CAMERA_Type * *base*, flexio_camera_edma_handle_t * *handle*, size_t * *count* )

This function gets the number of bytes still not received.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_CAMERA_Type. |
| *handle* | Pointer to the flexio_camera_edma_handle_t structure. |
| *count* | Number of bytes sent so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_InvalidArgument* | The count parameter is invalid. |

## 23.4 FlexIO I2C Master Driver

### 23.4.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2C master function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO I2C master driver includes functional APIs and transactional APIs.

Functional APIs target low level APIs. Functional APIs can be used for the FlexIO I2C master initialization/configuration/operation for the optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO I2C master peripheral and how to organize functional APIs to meet the application requirements. The FlexIO I2C master functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support an asynchronous transfer. This means that the functions FLEXIO_I2C_MasterTransferNonBlocking() set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_Success status.

### 23.4.2 Typical use case

#### 23.4.2.1 FlexIO I2C master transfer using an interrupt method

```
flexio_i2c_master_handle_t g_m_handle;
flexio_i2c_master_config_t masterConfig;
flexio_i2c_master_transfer_t masterXfer;
volatile bool completionFlag = false;
const uint8_t sendData[] = [......];
FLEXIO_I2C_Type i2cDev;

void FLEXIO_I2C_MasterCallback(FLEXIO_I2C_Type *base, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_Success == status)
    {
        completionFlag = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2C_MasterGetDefaultConfig(&masterConfig);

    FLEXIO_I2C_MasterInit(&i2cDev, &user_config);
    FLEXIO_I2C_MasterTransferCreateHandle(&i2cDev, &g_m_handle,
      FLEXIO_I2C_MasterCallback, NULL);

    // Prepares to send.
```

```
    masterXfer.slaveAddress = g_accel_address[0];
    masterXfer.direction = kI2C_Read;
    masterXfer.subaddress = &who_am_i_reg;
    masterXfer.subaddressSize = 1;
    masterXfer.data = &who_am_i_value;
    masterXfer.dataSize = 1;
    masterXfer.flags = kI2C_TransferDefaultFlag;

    // Sends out.
    FLEXIO_I2C_MasterTransferNonBlocking(&i2cDev, &g_m_handle, &
      masterXfer);

    // Wait for sending is complete.
    while (!completionFlag)
    {
    }

    // ...
}
```

## Data Structures

- struct FLEXIO_I2C_Type
  *Define FlexIO I2C master access structure typedef.* *More...*
- struct flexio_i2c_master_config_t
  *Define FlexIO I2C master user configuration structure.* *More...*
- struct flexio_i2c_master_transfer_t
  *Define FlexIO I2C master transfer structure.* *More...*
- struct flexio_i2c_master_handle_t
  *Define FlexIO I2C master handle structure.* *More...*

## Typedefs

- typedef void(∗ flexio_i2c_master_transfer_callback_t )(FLEXIO_I2C_Type ∗base, flexio_i2c_master_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO I2C master transfer callback typedef.*

## Enumerations

- enum _flexio_i2c_status {
  kStatus_FLEXIO_I2C_Busy = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 0),
  kStatus_FLEXIO_I2C_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 1),
  kStatus_FLEXIO_I2C_Nak = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 2) }
  *FlexIO I2C transfer status.*
- enum _flexio_i2c_master_interrupt {
  kFLEXIO_I2C_TxEmptyInterruptEnable = 0x1U,
  kFLEXIO_I2C_RxFullInterruptEnable = 0x2U }
  *Define FlexIO I2C master interrupt mask.*
- enum _flexio_i2c_master_status_flags {

kFLEXIO_I2C_TxEmptyFlag = 0x1U,
kFLEXIO_I2C_RxFullFlag = 0x2U,
kFLEXIO_I2C_ReceiveNakFlag = 0x4U }
*Define FlexIO I2C master status mask.*

- enum flexio_i2c_direction_t {
kFLEXIO_I2C_Write = 0x0U,
kFLEXIO_I2C_Read = 0x1U }
*Direction of master transfer.*

## Driver version

- #define FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))
*FlexIO I2C master driver version 2.1.7.*

## Initialization and deinitialization

- status_t FLEXIO_I2C_MasterInit (FLEXIO_I2C_Type *base, flexio_i2c_master_config_t *masterConfig, uint32_t srcClock_Hz)
*Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.*
- void FLEXIO_I2C_MasterDeinit (FLEXIO_I2C_Type *base)
*De-initializes the FlexIO I2C master peripheral.*
- void FLEXIO_I2C_MasterGetDefaultConfig (flexio_i2c_master_config_t *masterConfig)
*Gets the default configuration to configure the FlexIO module.*
- static void FLEXIO_I2C_MasterEnable (FLEXIO_I2C_Type *base, bool enable)
*Enables/disables the FlexIO module operation.*

## Status

- uint32_t FLEXIO_I2C_MasterGetStatusFlags (FLEXIO_I2C_Type *base)
*Gets the FlexIO I2C master status flags.*
- void FLEXIO_I2C_MasterClearStatusFlags (FLEXIO_I2C_Type *base, uint32_t mask)
*Clears the FlexIO I2C master status flags.*

## Interrupts

- void FLEXIO_I2C_MasterEnableInterrupts (FLEXIO_I2C_Type *base, uint32_t mask)
*Enables the FlexIO i2c master interrupt requests.*
- void FLEXIO_I2C_MasterDisableInterrupts (FLEXIO_I2C_Type *base, uint32_t mask)
*Disables the FlexIO I2C master interrupt requests.*

## Bus Operations

- void FLEXIO_I2C_MasterSetBaudRate (FLEXIO_I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  *Sets the FlexIO I2C master transfer baudrate.*
- void FLEXIO_I2C_MasterStart (FLEXIO_I2C_Type *base, uint8_t address, flexio_i2c_direction_t direction)
  *Sends START + 7-bit address to the bus.*
- void FLEXIO_I2C_MasterStop (FLEXIO_I2C_Type *base)
  *Sends the stop signal on the bus.*
- void FLEXIO_I2C_MasterRepeatedStart (FLEXIO_I2C_Type *base)
  *Sends the repeated start signal on the bus.*
- void FLEXIO_I2C_MasterAbortStop (FLEXIO_I2C_Type *base)
  *Sends the stop signal when transfer is still on-going.*
- void FLEXIO_I2C_MasterEnableAck (FLEXIO_I2C_Type *base, bool enable)
  *Configures the sent ACK/NAK for the following byte.*
- status_t FLEXIO_I2C_MasterSetTransferCount (FLEXIO_I2C_Type *base, uint8_t count)
  *Sets the number of bytes to be transferred from a start signal to a stop signal.*
- static void FLEXIO_I2C_MasterWriteByte (FLEXIO_I2C_Type *base, uint32_t data)
  *Writes one byte of data to the I2C bus.*
- static uint8_t FLEXIO_I2C_MasterReadByte (FLEXIO_I2C_Type *base)
  *Reads one byte of data from the I2C bus.*
- status_t FLEXIO_I2C_MasterWriteBlocking (FLEXIO_I2C_Type *base, const uint8_t *txBuff, uint8_t txSize)
  *Sends a buffer of data in bytes.*
- void FLEXIO_I2C_MasterReadBlocking (FLEXIO_I2C_Type *base, uint8_t *rxBuff, uint8_t rxSize)
  *Receives a buffer of bytes.*
- status_t FLEXIO_I2C_MasterTransferBlocking (FLEXIO_I2C_Type *base, flexio_i2c_master_transfer_t *xfer)
  *Performs a master polling transfer on the I2C bus.*

## Transactional

- status_t FLEXIO_I2C_MasterTransferCreateHandle (FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, flexio_i2c_master_transfer_callback_t callback, void *userData)
  *Initializes the I2C handle which is used in transactional functions.*
- status_t FLEXIO_I2C_MasterTransferNonBlocking (FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, flexio_i2c_master_transfer_t *xfer)
  *Performs a master interrupt non-blocking transfer on the I2C bus.*
- status_t FLEXIO_I2C_MasterTransferGetCount (FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, size_t *count)
  *Gets the master transfer status during a interrupt non-blocking transfer.*
- void FLEXIO_I2C_MasterTransferAbort (FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle)
  *Aborts an interrupt non-blocking transfer early.*
- void FLEXIO_I2C_MasterTransferHandleIRQ (void *i2cType, void *i2cHandle)
  *Master interrupt handler.*

### 23.4.3 Data Structure Documentation

#### 23.4.3.1 struct FLEXIO_I2C_Type

**Data Fields**

- FLEXIO_Type ∗ flexioBase
  *FlexIO base pointer.*
- uint8_t SDAPinIndex
  *Pin select for I2C SDA.*
- uint8_t SCLPinIndex
  *Pin select for I2C SCL.*
- uint8_t shifterIndex [2]
  *Shifter index used in FlexIO I2C.*
- uint8_t timerIndex [2]
  *Timer index used in FlexIO I2C.*

##### 23.4.3.1.0.1 Field Documentation

###### 23.4.3.1.0.1.1 FLEXIO_Type∗ FLEXIO_I2C_Type::flexioBase

###### 23.4.3.1.0.1.2 uint8_t FLEXIO_I2C_Type::SDAPinIndex

###### 23.4.3.1.0.1.3 uint8_t FLEXIO_I2C_Type::SCLPinIndex

###### 23.4.3.1.0.1.4 uint8_t FLEXIO_I2C_Type::shifterIndex[2]

###### 23.4.3.1.0.1.5 uint8_t FLEXIO_I2C_Type::timerIndex[2]

#### 23.4.3.2 struct flexio_i2c_master_config_t

**Data Fields**

- bool enableMaster
  *Enables the FlexIO I2C peripheral at initialization time.*
- bool enableInDoze
  *Enable/disable FlexIO operation in doze mode.*
- bool enableInDebug
  *Enable/disable FlexIO operation in debug mode.*
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers, fast access requires`
  *the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32_t baudRate_Bps
  *Baud rate in Bps.*

#### 23.4.3.2.0.2 Field Documentation

##### 23.4.3.2.0.2.1 bool flexio_i2c_master_config_t::enableMaster

##### 23.4.3.2.0.2.2 bool flexio_i2c_master_config_t::enableInDoze

##### 23.4.3.2.0.2.3 bool flexio_i2c_master_config_t::enableInDebug

##### 23.4.3.2.0.2.4 bool flexio_i2c_master_config_t::enableFastAccess

##### 23.4.3.2.0.2.5 uint32_t flexio_i2c_master_config_t::baudRate_Bps

### 23.4.3.3 struct flexio_i2c_master_transfer_t

### Data Fields

- uint32_t flags
  *Transfer flag which controls the transfer, reserved for FlexIO I2C.*
- uint8_t slaveAddress
  *7-bit slave address.*
- flexio_i2c_direction_t direction
  *Transfer direction, read or write.*
- uint32_t subaddress
  *Sub address.*
- uint8_t subaddressSize
  *Size of command buffer.*
- uint8_t volatile * data
  *Transfer buffer.*
- volatile size_t dataSize
  *Transfer size.*

#### 23.4.3.3.0.3 Field Documentation

##### 23.4.3.3.0.3.1 uint32_t flexio_i2c_master_transfer_t::flags

##### 23.4.3.3.0.3.2 uint8_t flexio_i2c_master_transfer_t::slaveAddress

##### 23.4.3.3.0.3.3 flexio_i2c_direction_t flexio_i2c_master_transfer_t::direction

##### 23.4.3.3.0.3.4 uint32_t flexio_i2c_master_transfer_t::subaddress

Transferred MSB first.

#### 23.4.3.3.0.3.5 uint8_t flexio_i2c_master_transfer_t::subaddressSize

#### 23.4.3.3.0.3.6 uint8_t volatile* flexio_i2c_master_transfer_t::data

#### 23.4.3.3.0.3.7 volatile size_t flexio_i2c_master_transfer_t::dataSize

### 23.4.3.4 struct _flexio_i2c_master_handle

FlexIO I2C master handle typedef.

### Data Fields

- flexio_i2c_master_transfer_t transfer
  *FlexIO I2C master transfer copy.*
- size_t transferSize
  *Total bytes to be transferred.*
- uint8_t state
  *Transfer state maintained during transfer.*
- flexio_i2c_master_transfer_callback_t completionCallback
  *Callback function called at transfer event.*
- void * userData
  *Callback parameter passed to callback function.*

#### 23.4.3.4.0.4 Field Documentation

#### 23.4.3.4.0.4.1 flexio_i2c_master_transfer_t flexio_i2c_master_handle_t::transfer

#### 23.4.3.4.0.4.2 size_t flexio_i2c_master_handle_t::transferSize

#### 23.4.3.4.0.4.3 uint8_t flexio_i2c_master_handle_t::state

#### 23.4.3.4.0.4.4 flexio_i2c_master_transfer_callback_t flexio_i2c_master_handle_t::completionCallback

Callback function called at transfer event.

#### 23.4.3.4.0.4.5 void* flexio_i2c_master_handle_t::userData

### 23.4.4 Macro Definition Documentation

#### 23.4.4.1 #define FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))

### 23.4.5 Typedef Documentation

#### 23.4.5.1 typedef void(* flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, status_t status, void *userData)

### 23.4.6 Enumeration Type Documentation

#### 23.4.6.1 enum _flexio_i2c_status

Enumerator

***kStatus_FLEXIO_I2C_Busy*** I2C is busy doing transfer.
***kStatus_FLEXIO_I2C_Idle*** I2C is busy doing transfer.
***kStatus_FLEXIO_I2C_Nak*** NAK received during transfer.

#### 23.4.6.2 enum _flexio_i2c_master_interrupt

Enumerator

***kFLEXIO_I2C_TxEmptyInterruptEnable*** Tx buffer empty interrupt enable.
***kFLEXIO_I2C_RxFullInterruptEnable*** Rx buffer full interrupt enable.

#### 23.4.6.3 enum _flexio_i2c_master_status_flags

Enumerator

***kFLEXIO_I2C_TxEmptyFlag*** Tx shifter empty flag.
***kFLEXIO_I2C_RxFullFlag*** Rx shifter full/Transfer complete flag.
***kFLEXIO_I2C_ReceiveNakFlag*** Receive NAK flag.

#### 23.4.6.4 enum flexio_i2c_direction_t

Enumerator

***kFLEXIO_I2C_Write*** Master send to slave.
***kFLEXIO_I2C_Read*** Master receive from slave.

### 23.4.7 Function Documentation

#### 23.4.7.1 status_t FLEXIO_I2C_MasterInit ( FLEXIO_I2C_Type * *base*, flexio_i2c_master_config_t * *masterConfig*, uint32_t *srcClock_Hz* )

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *masterConfig* | Pointer to flexio_i2c_master_config_t structure. |
| *srcClock_Hz* | FlexIO source clock in Hz. |

Return values

| | |
|---|---|
| *kStatus_Success* | Initialization successful |
| *kStatus_InvalidArgument* | The source clock exceed upper range limitation |

#### 23.4.7.2 void FLEXIO_I2C_MasterDeinit ( FLEXIO_I2C_Type * *base* )

Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_I2C_Type structure. |

#### 23.4.7.3 void FLEXIO_I2C_MasterGetDefaultConfig ( flexio_i2c_master_config_t * *masterConfig* )

The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

| | |
|---|---|
| *masterConfig* | Pointer to flexio_i2c_master_config_t structure. |

#### 23.4.7.4 static void FLEXIO_I2C_MasterEnable ( FLEXIO_I2C_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *enable* | Pass true to enable module, false does not have any effect. |

#### 23.4.7.5 uint32_t FLEXIO_I2C_MasterGetStatusFlags ( FLEXIO_I2C_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure |

Returns

Status flag, use status flag to AND _flexio_i2c_master_status_flags can get the related status.

#### 23.4.7.6 void FLEXIO_I2C_MasterClearStatusFlags ( FLEXIO_I2C_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *mask* | Status flag. The parameter can be any combination of the following values:<br>• kFLEXIO_I2C_RxFullFlag<br>• kFLEXIO_I2C_ReceiveNakFlag |

#### 23.4.7.7 void FLEXIO_I2C_MasterEnableInterrupts ( FLEXIO_I2C_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *mask* | Interrupt source. Currently only one interrupt request source:<br>• kFLEXIO_I2C_TransferCompleteInterruptEnable |

#### 23.4.7.8 void FLEXIO_I2C_MasterDisableInterrupts ( FLEXIO_I2C_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *mask* | Interrupt source. |

#### 23.4.7.9 void FLEXIO_I2C_MasterSetBaudRate ( FLEXIO_I2C_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure |
| *baudRate_Bps* | the baud rate value in HZ |
| *srcClock_Hz* | source clock in HZ |

#### 23.4.7.10 void FLEXIO_I2C_MasterStart ( FLEXIO_I2C_Type * *base*, uint8_t *address*, flexio_i2c_direction_t *direction* )

Note

This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_-RxFullFlag status is asserted before calling this API.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *address* | 7-bit address. |
| *direction* | transfer direction. This parameter is one of the values in flexio_i2c_direction_t:<br>• kFLEXIO_I2C_Write: Transmit<br>• kFLEXIO_I2C_Read: Receive |

#### 23.4.7.11 void FLEXIO_I2C_MasterStop ( FLEXIO_I2C_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |

#### 23.4.7.12 void FLEXIO_I2C_MasterRepeatedStart ( FLEXIO_I2C_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |

#### 23.4.7.13 void FLEXIO_I2C_MasterAbortStop ( FLEXIO_I2C_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |

#### 23.4.7.14 void FLEXIO_I2C_MasterEnableAck ( FLEXIO_I2C_Type * *base*, bool *enable* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *enable* | True to configure send ACK, false configure to send NAK. |

#### 23.4.7.15 status_t FLEXIO_I2C_MasterSetTransferCount ( FLEXIO_I2C_Type * *base*, uint8_t *count* )

Note

Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *count* | Number of bytes need to be transferred from a start signal to a re-start/stop signal |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully configured the count. |
| *kStatus_InvalidArgument* | Input argument is invalid. |

#### 23.4.7.16 static void FLEXIO_I2C_MasterWriteByte ( FLEXIO_I2C_Type * *base*, uint32_t *data* ) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *data* | a byte of data. |

#### 23.4.7.17 static uint8_t FLEXIO_I2C_MasterReadByte ( FLEXIO_I2C_Type * *base* ) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |

Returns

data byte read.

### 23.4.7.18 status_t FLEXIO_I2C_MasterWriteBlocking ( FLEXIO_I2C_Type * *base*, const uint8_t * *txBuff*, uint8_t *txSize* )

Note

This function blocks via polling until all bytes have been sent.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *txBuff* | The data bytes to send. |
| *txSize* | The number of data bytes to send. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully write data. |
| *kStatus_FLEXIO_I2C_-Nak* | Receive NAK during writing data. |

### 23.4.7.19 void FLEXIO_I2C_MasterReadBlocking ( FLEXIO_I2C_Type * *base*, uint8_t * *rxBuff*, uint8_t *rxSize* )

Note

This function blocks via polling until all bytes have been received.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *rxBuff* | The buffer to store the received bytes. |
| *rxSize* | The number of data bytes to be received. |

#### 23.4.7.20 status_t FLEXIO_I2C_MasterTransferBlocking ( FLEXIO_I2C_Type * *base*, flexio_i2c_master_transfer_t * *xfer* )

Note

The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_I2C_Type structure. |
| *xfer* | pointer to flexio_i2c_master_transfer_t structure. |

Returns

status of status_t.

#### 23.4.7.21 status_t FLEXIO_I2C_MasterTransferCreateHandle ( FLEXIO_I2C_Type * *base*, flexio_i2c_master_handle_t * *handle*, flexio_i2c_master_transfer_callback_t *callback*, void * *userData* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *handle* | Pointer to flexio_i2c_master_handle_t structure to store the transfer state. |
| *callback* | Pointer to user callback function. |
| *userData* | User param passed to the callback function. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO type/handle/isr table out of range. |

#### 23.4.7.22 status_t FLEXIO_I2C_MasterTransferNonBlocking ( FLEXIO_I2C_Type * *base*, flexio_i2c_master_handle_t * *handle*, flexio_i2c_master_transfer_t * *xfer* )

Note

The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure |
| *handle* | Pointer to flexio_i2c_master_handle_t structure which stores the transfer state |
| *xfer* | pointer to flexio_i2c_master_transfer_t structure |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_FLEXIO_I2C_Busy* | FlexIO I2C is not idle, is running another transfer. |

#### 23.4.7.23 status_t FLEXIO_I2C_MasterTransferGetCount ( FLEXIO_I2C_Type * *base*, flexio_i2c_master_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure. |
| *handle* | Pointer to flexio_i2c_master_handle_t structure which stores the transfer state. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_InvalidArgument* | count is Invalid. |
| *kStatus_Success* | Successfully return the count. |

#### 23.4.7.24 void FLEXIO_I2C_MasterTransferAbort ( FLEXIO_I2C_Type * *base*, flexio_i2c_master_handle_t * *handle* )

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2C_Type structure |
| *handle* | Pointer to flexio_i2c_master_handle_t structure which stores the transfer state |

#### 23.4.7.25 void FLEXIO_I2C_MasterTransferHandleIRQ ( void * *i2cType*, void * *i2cHandle* )

Parameters

| | |
|---|---|
| *i2cType* | Pointer to FLEXIO_I2C_Type structure |
| *i2cHandle* | Pointer to flexio_i2c_master_transfer_t structure |

## 23.5 FlexIO I2S Driver

### 23.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2S function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO I2S driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs.

Functional APIs can be used for FlexIO I2S initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO I2S peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO I2S functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the sai_handle_t as the first parameter. Initialize the handle by calling the FlexIO_I2S_TransferTxCreateHandle() or FlexIO_I2S_TransferRxCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions FLEXIO_I2S_TransferSendNonBlocking() and FLEXIO_I2S_TransferReceiveNonBlocking() set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_FLEXIO_I2S_TxIdle and kStatus_FLEXIO_I2S_RxIdle status.

### 23.5.2 Typical use case

#### 23.5.2.1 FlexIO I2S send/receive using an interrupt method

```
sai_handle_t g_saiTxHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
volatile bool rxFinished;
const uint8_t sendData[] = [......];

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);
```

```
    FLEXIO_I2S_TxInit(FLEXIO I2S0, &user_config);
    FLEXIO_I2S_TransferTxCreateHandle(FLEXIO I2S0, &g_saiHandle,
      FLEXIO_I2S_UserCallback, NULL);

    //Configures the SAI format.
    FLEXIO_I2S_TransferTxSetTransferFormat(FLEXIO I2S0, &g_saiHandle, mclkSource, mclk);

    // Prepares to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_I2S_TransferSendNonBlocking(FLEXIO I2S0, &g_saiHandle, &
      sendXfer);

    // Waiting to send is finished.
    while (!txFinished)
    {
    }

    // ...
}
```

#### 23.5.2.2 FLEXIO_I2S send/receive using a DMA method

```
sai_handle_t g_saiHandle;
dma_handle_t g_saiTxDmaHandle;
dma_handle_t g_saiRxDmaHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
uint8_t sendData[] = ...;

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);
    FLEXIO_I2S_TxInit(FLEXIO I2S0, &user_config);

    // Sets up the DMA.
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL, FLEXIO_I2S_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL);

    DMA_Init(DMA0);

    /* Creates the DMA handle. */
    DMA_TransferTxCreateHandle(&g_saiTxDmaHandle, DMA0, FLEXIO_I2S_TX_DMA_CHANNEL);

    FLEXIO_I2S_TransferTxCreateHandleDMA(FLEXIO I2S0, &g_saiTxDmaHandle
      , FLEXIO_I2S_UserCallback, NULL);
```

```
    // Prepares to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_I2S_TransferSendDMA(&g_saiHandle, &sendXfer);

    // Waiting to send is finished.
    while (!txFinished)
    {
    }

    // ...
}
```

## Modules

- FlexIO DMA I2S Driver
- FlexIO eDMA I2S Driver

## Data Structures

- struct FLEXIO_I2S_Type
  *Define FlexIO I2S access structure typedef.* *More...*
- struct flexio_i2s_config_t
  *FlexIO I2S configure structure.* *More...*
- struct flexio_i2s_format_t
  *FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.* *More...*
- struct flexio_i2s_transfer_t
  *Define FlexIO I2S transfer structure.* *More...*
- struct flexio_i2s_handle_t
  *Define FlexIO I2S handle structure.* *More...*

## Macros

- #define FLEXIO_I2S_XFER_QUEUE_SIZE (4)
  *FlexIO I2S transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(∗ flexio_i2s_callback_t )(FLEXIO_I2S_Type ∗base, flexio_i2s_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO I2S xfer callback prototype.*

## Enumerations

- enum _flexio_i2s_status {
  kStatus_FLEXIO_I2S_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 0),
  kStatus_FLEXIO_I2S_TxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 1),
  kStatus_FLEXIO_I2S_RxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 2),
  kStatus_FLEXIO_I2S_Error = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 3),
  kStatus_FLEXIO_I2S_QueueFull = MAKE_STATUS(kStatusGroup_FLEXIO_I2S, 4) }
  *FlexIO I2S transfer status.*
- enum flexio_i2s_master_slave_t {
  kFLEXIO_I2S_Master = 0x0U,
  kFLEXIO_I2S_Slave = 0x1U }
  *Master or slave mode.*
- enum _flexio_i2s_interrupt_enable {
  kFLEXIO_I2S_TxDataRegEmptyInterruptEnable = 0x1U,
  kFLEXIO_I2S_RxDataRegFullInterruptEnable = 0x2U }
  *Define FlexIO FlexIO I2S interrupt mask.*
- enum _flexio_i2s_status_flags {
  kFLEXIO_I2S_TxDataRegEmptyFlag = 0x1U,
  kFLEXIO_I2S_RxDataRegFullFlag = 0x2U }
  *Define FlexIO FlexIO I2S status mask.*
- enum flexio_i2s_sample_rate_t {
  kFLEXIO_I2S_SampleRate8KHz = 8000U,
  kFLEXIO_I2S_SampleRate11025Hz = 11025U,
  kFLEXIO_I2S_SampleRate12KHz = 12000U,
  kFLEXIO_I2S_SampleRate16KHz = 16000U,
  kFLEXIO_I2S_SampleRate22050Hz = 22050U,
  kFLEXIO_I2S_SampleRate24KHz = 24000U,
  kFLEXIO_I2S_SampleRate32KHz = 32000U,
  kFLEXIO_I2S_SampleRate44100Hz = 44100U,
  kFLEXIO_I2S_SampleRate48KHz = 48000U,
  kFLEXIO_I2S_SampleRate96KHz = 96000U }
  *Audio sample rate.*
- enum flexio_i2s_word_width_t {
  kFLEXIO_I2S_WordWidth8bits = 8U,
  kFLEXIO_I2S_WordWidth16bits = 16U,
  kFLEXIO_I2S_WordWidth24bits = 24U,
  kFLEXIO_I2S_WordWidth32bits = 32U }
  *Audio word width.*

## Driver version

- #define FSL_FLEXIO_I2S_DRIVER_VERSION (MAKE_VERSION(2, 1, 6))
  *FlexIO I2S driver version 2.1.6.*

### Initialization and deinitialization

- void FLEXIO_I2S_Init (FLEXIO_I2S_Type *base, const flexio_i2s_config_t *config)
  *Initializes the FlexIO I2S.*
- void FLEXIO_I2S_GetDefaultConfig (flexio_i2s_config_t *config)
  *Sets the FlexIO I2S configuration structure to default values.*
- void FLEXIO_I2S_Deinit (FLEXIO_I2S_Type *base)
  *De-initializes the FlexIO I2S.*
- static void FLEXIO_I2S_Enable (FLEXIO_I2S_Type *base, bool enable)
  *Enables/disables the FlexIO I2S module operation.*

### Status

- uint32_t FLEXIO_I2S_GetStatusFlags (FLEXIO_I2S_Type *base)
  *Gets the FlexIO I2S status flags.*

### Interrupts

- void FLEXIO_I2S_EnableInterrupts (FLEXIO_I2S_Type *base, uint32_t mask)
  *Enables the FlexIO I2S interrupt.*
- void FLEXIO_I2S_DisableInterrupts (FLEXIO_I2S_Type *base, uint32_t mask)
  *Disables the FlexIO I2S interrupt.*

### DMA Control

- static void FLEXIO_I2S_TxEnableDMA (FLEXIO_I2S_Type *base, bool enable)
  *Enables/disables the FlexIO I2S Tx DMA requests.*
- static void FLEXIO_I2S_RxEnableDMA (FLEXIO_I2S_Type *base, bool enable)
  *Enables/disables the FlexIO I2S Rx DMA requests.*
- static uint32_t FLEXIO_I2S_TxGetDataRegisterAddress (FLEXIO_I2S_Type *base)
  *Gets the FlexIO I2S send data register address.*
- static uint32_t FLEXIO_I2S_RxGetDataRegisterAddress (FLEXIO_I2S_Type *base)
  *Gets the FlexIO I2S receive data register address.*

### Bus Operations

- void FLEXIO_I2S_MasterSetFormat (FLEXIO_I2S_Type *base, flexio_i2s_format_t *format, uint32_t srcClock_Hz)
  *Configures the FlexIO I2S audio format in master mode.*
- void FLEXIO_I2S_SlaveSetFormat (FLEXIO_I2S_Type *base, flexio_i2s_format_t *format)
  *Configures the FlexIO I2S audio format in slave mode.*
- void FLEXIO_I2S_WriteBlocking (FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *txData, size_t size)
  *Sends data using a blocking method.*
- static void FLEXIO_I2S_WriteData (FLEXIO_I2S_Type *base, uint8_t bitWidth, uint32_t data)

*Writes data into a data register.*
- void FLEXIO_I2S_ReadBlocking (FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *rxData, size_t size)
  *Receives a piece of data using a blocking method.*
- static uint32_t FLEXIO_I2S_ReadData (FLEXIO_I2S_Type *base)
  *Reads a data from the data register.*

### Transactional

- void FLEXIO_I2S_TransferTxCreateHandle (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_callback_t callback, void *userData)
  *Initializes the FlexIO I2S handle.*
- void FLEXIO_I2S_TransferSetFormat (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_format_t *format, uint32_t srcClock_Hz)
  *Configures the FlexIO I2S audio format.*
- void FLEXIO_I2S_TransferRxCreateHandle (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_callback_t callback, void *userData)
  *Initializes the FlexIO I2S receive handle.*
- status_t FLEXIO_I2S_TransferSendNonBlocking (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_transfer_t *xfer)
  *Performs an interrupt non-blocking send transfer on FlexIO I2S.*
- status_t FLEXIO_I2S_TransferReceiveNonBlocking (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_transfer_t *xfer)
  *Performs an interrupt non-blocking receive transfer on FlexIO I2S.*
- void FLEXIO_I2S_TransferAbortSend (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)
  *Aborts the current send.*
- void FLEXIO_I2S_TransferAbortReceive (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)
  *Aborts the current receive.*
- status_t FLEXIO_I2S_TransferGetSendCount (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)
  *Gets the remaining bytes to be sent.*
- status_t FLEXIO_I2S_TransferGetReceiveCount (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)
  *Gets the remaining bytes to be received.*
- void FLEXIO_I2S_TransferTxHandleIRQ (void *i2sBase, void *i2sHandle)
  *Tx interrupt handler.*
- void FLEXIO_I2S_TransferRxHandleIRQ (void *i2sBase, void *i2sHandle)
  *Rx interrupt handler.*

## 23.5.3 Data Structure Documentation

### 23.5.3.1 struct FLEXIO_I2S_Type

### Data Fields

- FLEXIO_Type * flexioBase
  *FlexIO base pointer.*

- uint8_t txPinIndex
  *Tx data pin index in FlexIO pins.*
- uint8_t rxPinIndex
  *Rx data pin index.*
- uint8_t bclkPinIndex
  *Bit clock pin index.*
- uint8_t fsPinIndex
  *Frame sync pin index.*
- uint8_t txShifterIndex
  *Tx data shifter index.*
- uint8_t rxShifterIndex
  *Rx data shifter index.*
- uint8_t bclkTimerIndex
  *Bit clock timer index.*
- uint8_t fsTimerIndex
  *Frame sync timer index.*

#### 23.5.3.2 struct flexio_i2s_config_t

### Data Fields

- bool enableI2S
  *Enable FlexIO I2S.*
- flexio_i2s_master_slave_t masterSlave
  *Master or slave.*
- flexio_pin_polarity_t txPinPolarity
  *Tx data pin polarity, active high or low.*
- flexio_pin_polarity_t rxPinPolarity
  *Rx data pin polarity.*
- flexio_pin_polarity_t bclkPinPolarity
  *Bit clock pin polarity.*
- flexio_pin_polarity_t fsPinPolarity
  *Frame sync pin polarity.*
- flexio_shifter_timer_polarity_t txTimerPolarity
  *Tx data valid on bclk rising or falling edge.*
- flexio_shifter_timer_polarity_t rxTimerPolarity
  *Rx data valid on bclk rising or falling edge.*

#### 23.5.3.3 struct flexio_i2s_format_t

### Data Fields

- uint8_t bitWidth
  *Bit width of audio data, always 8/16/24/32 bits.*
- uint32_t sampleRate_Hz
  *Sample rate of the audio data.*

#### 23.5.3.4 struct flexio_i2s_transfer_t

### Data Fields

- uint8_t * data
  *Data buffer start pointer.*
- size_t dataSize
  *Bytes to be transferred.*

##### 23.5.3.4.0.5 Field Documentation

###### 23.5.3.4.0.5.1 size_t flexio_i2s_transfer_t::dataSize

#### 23.5.3.5 struct _flexio_i2s_handle

### Data Fields

- uint32_t state
  *Internal state.*
- flexio_i2s_callback_t callback
  *Callback function called at transfer event.*
- void * userData
  *Callback parameter passed to callback function.*
- uint8_t bitWidth
  *Bit width for transfer, 8/16/24/32bits.*
- flexio_i2s_transfer_t queue [FLEXIO_I2S_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [FLEXIO_I2S_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*

## 23.5.4 Macro Definition Documentation

#### 23.5.4.1 #define FSL_FLEXIO_I2S_DRIVER_VERSION (MAKE_VERSION(2, 1, 6))

#### 23.5.4.2 #define FLEXIO_I2S_XFER_QUEUE_SIZE (4)

## 23.5.5 Enumeration Type Documentation

#### 23.5.5.1 enum _flexio_i2s_status

Enumerator

***kStatus_FLEXIO_I2S_Idle*** FlexIO I2S is in idle state.
***kStatus_FLEXIO_I2S_TxBusy*** FlexIO I2S Tx is busy.

*kStatus_FLEXIO_I2S_RxBusy* FlexIO I2S Tx is busy.
*kStatus_FLEXIO_I2S_Error* FlexIO I2S error occurred.
*kStatus_FLEXIO_I2S_QueueFull* FlexIO I2S transfer queue is full.

#### 23.5.5.2 enum flexio_i2s_master_slave_t

Enumerator

*kFLEXIO_I2S_Master* Master mode.
*kFLEXIO_I2S_Slave* Slave mode.

#### 23.5.5.3 enum _flexio_i2s_interrupt_enable

Enumerator

*kFLEXIO_I2S_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.
*kFLEXIO_I2S_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

#### 23.5.5.4 enum _flexio_i2s_status_flags

Enumerator

*kFLEXIO_I2S_TxDataRegEmptyFlag* Transmit buffer empty flag.
*kFLEXIO_I2S_RxDataRegFullFlag* Receive buffer full flag.

#### 23.5.5.5 enum flexio_i2s_sample_rate_t

Enumerator

*kFLEXIO_I2S_SampleRate8KHz* Sample rate 8000Hz.
*kFLEXIO_I2S_SampleRate11025Hz* Sample rate 11025Hz.
*kFLEXIO_I2S_SampleRate12KHz* Sample rate 12000Hz.
*kFLEXIO_I2S_SampleRate16KHz* Sample rate 16000Hz.
*kFLEXIO_I2S_SampleRate22050Hz* Sample rate 22050Hz.
*kFLEXIO_I2S_SampleRate24KHz* Sample rate 24000Hz.
*kFLEXIO_I2S_SampleRate32KHz* Sample rate 32000Hz.
*kFLEXIO_I2S_SampleRate44100Hz* Sample rate 44100Hz.
*kFLEXIO_I2S_SampleRate48KHz* Sample rate 48000Hz.
*kFLEXIO_I2S_SampleRate96KHz* Sample rate 96000Hz.

#### 23.5.5.6 enum flexio_i2s_word_width_t

Enumerator

***kFLEXIO_I2S_WordWidth8bits*** Audio data width 8 bits.
***kFLEXIO_I2S_WordWidth16bits*** Audio data width 16 bits.
***kFLEXIO_I2S_WordWidth24bits*** Audio data width 24 bits.
***kFLEXIO_I2S_WordWidth32bits*** Audio data width 32 bits.

### 23.5.6 Function Documentation

#### 23.5.6.1 void FLEXIO_I2S_Init ( FLEXIO_I2S_Type * *base*, const flexio_i2s_config_t * *config* )

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO_I2S_GetDefaultConfig().

Note

This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer |
| *config* | FlexIO I2S configure structure. |

#### 23.5.6.2 void FLEXIO_I2S_GetDefaultConfig ( flexio_i2s_config_t * *config* )

The purpose of this API is to get the configuration structure initialized for use in FLEXIO_I2S_Init(). Users may use the initialized structure unchanged in FLEXIO_I2S_Init() or modify some fields of the structure before calling FLEXIO_I2S_Init().

Parameters

| | |
|---|---|
| *config* | pointer to master configuration structure |

#### 23.5.6.3 void FLEXIO_I2S_Deinit ( FLEXIO_I2S_Type * *base* )

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXO_I2S_Init to use the FlexIO I2S module.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer |

#### 23.5.6.4 static void FLEXIO_I2S_Enable ( FLEXIO_I2S_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type |
| *enable* | True to enable, false dose not have any effect. |

#### 23.5.6.5 uint32_t FLEXIO_I2S_GetStatusFlags ( FLEXIO_I2S_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure |

Returns

Status flag, which are ORed by the enumerators in the _flexio_i2s_status_flags.

#### 23.5.6.6 void FLEXIO_I2S_EnableInterrupts ( FLEXIO_I2S_Type * *base*, uint32_t *mask* )

This function enables the FlexIO UART interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure |
| *mask* | interrupt source |

#### 23.5.6.7 void FLEXIO_I2S_DisableInterrupts ( FLEXIO_I2S_Type * *base*, uint32_t *mask* )

This function enables the FlexIO UART interrupt.

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_I2S_Type structure |
| *mask* | interrupt source |

#### 23.5.6.8 static void FLEXIO_I2S_TxEnableDMA ( FLEXIO_I2S_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer |
| *enable* | True means enable DMA, false means disable DMA. |

#### 23.5.6.9 static void FLEXIO_I2S_RxEnableDMA ( FLEXIO_I2S_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer |
| *enable* | True means enable DMA, false means disable DMA. |

#### 23.5.6.10 static uint32_t FLEXIO_I2S_TxGetDataRegisterAddress ( FLEXIO_I2S_Type * *base* ) [inline], [static]

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure |

Returns

FlexIO i2s send data register address.

#### 23.5.6.11 static uint32_t FLEXIO_I2S_RxGetDataRegisterAddress ( FLEXIO_I2S_Type * *base* ) [inline], [static]

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure |

Returns

FlexIO i2s receive data register address.

### 23.5.6.12 void FLEXIO_I2S_MasterSetFormat ( FLEXIO_I2S_Type * *base*, flexio_i2s_format_t * *format*, uint32_t *srcClock_Hz* )

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure |
| *format* | Pointer to FlexIO I2S audio data format structure. |
| *srcClock_Hz* | I2S master clock source frequency in Hz. |

### 23.5.6.13 void FLEXIO_I2S_SlaveSetFormat ( FLEXIO_I2S_Type * *base*, flexio_i2s_format_t * *format* )

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure |
| *format* | Pointer to FlexIO I2S audio data format structure. |

### 23.5.6.14 void FLEXIO_I2S_WriteBlocking ( FLEXIO_I2S_Type * *base*, uint8_t *bitWidth*, uint8_t * *txData*, size_t *size* )

Note

This function blocks via polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer. |
| *bitWidth* | How many bits in a audio word, usually 8/16/24/32 bits. |
| *txData* | Pointer to the data to be written. |
| *size* | Bytes to be written. |

#### 23.5.6.15 static void FLEXIO_I2S_WriteData ( FLEXIO_I2S_Type * *base*, uint8_t *bitWidth*, uint32_t *data* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer. |
| *bitWidth* | How many bits in a audio word, usually 8/16/24/32 bits. |
| *data* | Data to be written. |

#### 23.5.6.16 void FLEXIO_I2S_ReadBlocking ( FLEXIO_I2S_Type * *base*, uint8_t *bitWidth*, uint8_t * *rxData*, size_t *size* )

Note

This function blocks via polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer |
| *bitWidth* | How many bits in a audio word, usually 8/16/24/32 bits. |
| *rxData* | Pointer to the data to be read. |
| *size* | Bytes to be read. |

#### 23.5.6.17 static uint32_t FLEXIO_I2S_ReadData ( FLEXIO_I2S_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FlexIO I2S base pointer |

Returns

Data read from data register.

### 23.5.6.18 void FLEXIO_I2S_TransferTxCreateHandle ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_callback_t *callback*, void * *userData* )

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure |
| *handle* | Pointer to flexio_i2s_handle_t structure to store the transfer state. |
| *callback* | FlexIO I2S callback function, which is called while finished a block. |
| *userData* | User parameter for the FlexIO I2S callback. |

### 23.5.6.19 void FLEXIO_I2S_TransferSetFormat ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_format_t * *format*, uint32_t *srcClock_Hz* )

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | FlexIO I2S handle pointer. |
| *format* | Pointer to audio data format structure. |
| *srcClock_Hz* | FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode. |

### 23.5.6.20 void FLEXIO_I2S_TransferRxCreateHandle ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_callback_t *callback*, void * *userData* )

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | Pointer to flexio_i2s_handle_t structure to store the transfer state. |
| *callback* | FlexIO I2S callback function, which is called while finished a block. |
| *userData* | User parameter for the FlexIO I2S callback. |

#### 23.5.6.21 status_t FLEXIO_I2S_TransferSendNonBlocking ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_transfer_t * *xfer* )

Note

The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | Pointer to flexio_i2s_handle_t structure which stores the transfer state |
| *xfer* | Pointer to flexio_i2s_transfer_t structure |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start the data transmission. |
| *kStatus_FLEXIO_I2S_TxBusy* | Previous transmission still not finished, data not all written to TX register yet. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

#### 23.5.6.22 status_t FLEXIO_I2S_TransferReceiveNonBlocking ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, flexio_i2s_transfer_t * *xfer* )

Note

The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | Pointer to flexio_i2s_handle_t structure which stores the transfer state |
| *xfer* | Pointer to flexio_i2s_transfer_t structure |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start the data receive. |
| *kStatus_FLEXIO_I2S_-RxBusy* | Previous receive still not finished. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

#### 23.5.6.23 void FLEXIO_I2S_TransferAbortSend ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle* )

Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | Pointer to flexio_i2s_handle_t structure which stores the transfer state |

#### 23.5.6.24 void FLEXIO_I2S_TransferAbortReceive ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle* )

Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | Pointer to flexio_i2s_handle_t structure which stores the transfer state |

#### 23.5.6.25 status_t FLEXIO_I2S_TransferGetSendCount ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | Pointer to flexio_i2s_handle_t structure which stores the transfer state |
| *count* | Bytes sent. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

#### 23.5.6.26 status_t FLEXIO_I2S_TransferGetReceiveCount ( FLEXIO_I2S_Type * *base*, flexio_i2s_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_I2S_Type structure. |
| *handle* | Pointer to flexio_i2s_handle_t structure which stores the transfer state |

Returns

count Bytes received.

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

#### 23.5.6.27 void FLEXIO_I2S_TransferTxHandleIRQ ( void * *i2sBase*, void * *i2sHandle* )

Parameters

| | |
|---|---|
| *i2sBase* | Pointer to FLEXIO_I2S_Type structure. |
| *i2sHandle* | Pointer to flexio_i2s_handle_t structure |

### 23.5.6.28 void FLEXIO_I2S_TransferRxHandleIRQ ( void * *i2sBase*, void * *i2sHandle* )

Parameters

| | |
|---|---|
| *i2sBase* | Pointer to FLEXIO_I2S_Type structure. |
| *i2sHandle* | Pointer to flexio_i2s_handle_t structure. |

### 23.5.7 FlexIO eDMA I2S Driver

#### 23.5.7.1 Overview

#### Data Structures

- struct flexio_i2s_edma_handle_t
  *FlexIO I2S DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef void(∗ flexio_i2s_edma_callback_t )(FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO I2S eDMA transfer callback function for finish and error.*

#### Driver version

- #define FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))
  *FlexIO I2S EDMA driver version 2.1.5.*

#### eDMA Transactional

- void FLEXIO_I2S_TransferTxCreateHandleEDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle, flexio_i2s_edma_callback_t callback, void ∗userData, edma_handle_t ∗dmaHandle)
  *Initializes the FlexIO I2S eDMA handle.*
- void FLEXIO_I2S_TransferRxCreateHandleEDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle, flexio_i2s_edma_callback_t callback, void ∗userData, edma_handle_t ∗dmaHandle)
  *Initializes the FlexIO I2S Rx eDMA handle.*
- void FLEXIO_I2S_TransferSetFormatEDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle, flexio_i2s_format_t ∗format, uint32_t srcClock_Hz)
  *Configures the FlexIO I2S Tx audio format.*
- status_t FLEXIO_I2S_TransferSendEDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle, flexio_i2s_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO I2S transfer using DMA.*
- status_t FLEXIO_I2S_TransferReceiveEDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle, flexio_i2s_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO I2S receive using eDMA.*
- void FLEXIO_I2S_TransferAbortSendEDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle)
  *Aborts a FlexIO I2S transfer using eDMA.*
- void FLEXIO_I2S_TransferAbortReceiveEDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_edma_handle_t ∗handle)
  *Aborts a FlexIO I2S receive using eDMA.*

- status_t FLEXIO_I2S_TransferGetSendCountEDMA (FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t *handle, size_t *count)
  *Gets the remaining bytes to be sent.*
- status_t FLEXIO_I2S_TransferGetReceiveCountEDMA (FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t *handle, size_t *count)
  *Get the remaining bytes to be received.*

### 23.5.7.2 Data Structure Documentation

#### 23.5.7.2.1 struct _flexio_i2s_edma_handle

**Data Fields**

- edma_handle_t * dmaHandle
  *DMA handler for FlexIO I2S send.*
- uint8_t bytesPerFrame
  *Bytes in a frame.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- uint32_t state
  *Internal state for FlexIO I2S eDMA transfer.*
- flexio_i2s_edma_callback_t callback
  *Callback for users while transfer finish or error occurred.*
- void * userData
  *User callback parameter.*
- edma_tcd_t tcd [FLEXIO_I2S_XFER_QUEUE_SIZE+1U]
  *TCD pool for eDMA transfer.*
- flexio_i2s_transfer_t queue [FLEXIO_I2S_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [FLEXIO_I2S_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*

#### 23.5.7.2.1.1 Field Documentation

##### 23.5.7.2.1.1.1 uint8_t flexio_i2s_edma_handle_t::nbytes

##### 23.5.7.2.1.1.2 edma_tcd_t flexio_i2s_edma_handle_t::tcd[FLEXIO_I2S_XFER_QUEUE_SIZE+1U]

##### 23.5.7.2.1.1.3 flexio_i2s_transfer_t flexio_i2s_edma_handle_t::queue[FLEXIO_I2S_XFER_QUEUE_SIZE]

##### 23.5.7.2.1.1.4 volatile uint8_t flexio_i2s_edma_handle_t::queueUser

### 23.5.7.3 Macro Definition Documentation

#### 23.5.7.3.1 #define FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))

### 23.5.7.4 Function Documentation

#### 23.5.7.4.1 void FLEXIO_I2S_TransferTxCreateHandleEDMA ( FLEXIO_I2S_Type * *base,* flexio_i2s_edma_handle_t * *handle,* flexio_i2s_edma_callback_t *callback,* void * *userData,* edma_handle_t * *dmaHandle* )

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S eDMA handle pointer. |
| *callback* | FlexIO I2S eDMA callback function called while finished a block. |
| *userData* | User parameter for callback. |
| *dmaHandle* | eDMA handle for FlexIO I2S. This handle is a static value allocated by users. |

#### 23.5.7.4.2 void FLEXIO_I2S_TransferRxCreateHandleEDMA ( FLEXIO_I2S_Type * *base,* flexio_i2s_edma_handle_t * *handle,* flexio_i2s_edma_callback_t *callback,* void * *userData,* edma_handle_t * *dmaHandle* )

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S eDMA handle pointer. |
| *callback* | FlexIO I2S eDMA callback function called while finished a block. |
| *userData* | User parameter for callback. |
| *dmaHandle* | eDMA handle for FlexIO I2S. This handle is a static value allocated by users. |

#### 23.5.7.4.3 void FLEXIO_I2S_TransferSetFormatEDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, flexio_i2s_format_t * *format*, uint32_t *srcClock_Hz* )

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S eDMA handle pointer |
| *format* | Pointer to FlexIO I2S audio data format structure. |
| *srcClock_Hz* | FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |

#### 23.5.7.4.4 status_t FLEXIO_I2S_TransferSendEDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, flexio_i2s_transfer_t * *xfer* )

Note

This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a FlexIO I2S eDMA send successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |
| *kStatus_TxBusy* | FlexIO I2S is busy sending data. |

#### 23.5.7.4.5 status_t FLEXIO_I2S_TransferReceiveEDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, flexio_i2s_transfer_t * *xfer* )

Note

This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a FlexIO I2S eDMA receive successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |
| *kStatus_RxBusy* | FlexIO I2S is busy receiving data. |

#### 23.5.7.4.6 void FLEXIO_I2S_TransferAbortSendEDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |

#### 23.5.7.4.7 void FLEXIO_I2S_TransferAbortReceiveEDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |

#### 23.5.7.4.8 status_t FLEXIO_I2S_TransferGetSendCountEDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *count* | Bytes sent. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

#### 23.5.7.4.9 status_t FLEXIO_I2S_TransferGetReceiveCountEDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |

| | |
|---|---|
| *count* | Bytes received. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferIn-Progress* | There is not a non-blocking transaction currently in progress. |

## 23.5.8 FlexIO DMA I2S Driver

### 23.5.8.1 Overview

#### Data Structures

- struct flexio_i2s_dma_handle_t
  *FlexIO I2S DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef void(∗ flexio_i2s_dma_callback_t )(FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO I2S DMA transfer callback function for finish and error.*

#### Driver version

- #define FSL_FLEXIO_I2S_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))
  *FlexIO I2S DMA driver version 2.1.5.*

#### DMA Transactional

- void FLEXIO_I2S_TransferTxCreateHandleDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle, flexio_i2s_dma_callback_t callback, void ∗userData, dma_handle_t ∗dmaHandle)
  *Initializes the FlexIO I2S DMA handle.*
- void FLEXIO_I2S_TransferRxCreateHandleDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle, flexio_i2s_dma_callback_t callback, void ∗userData, dma_handle_t ∗dmaHandle)
  *Initializes the FlexIO I2S Rx DMA handle.*
- void FLEXIO_I2S_TransferSetFormatDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle, flexio_i2s_format_t ∗format, uint32_t srcClock_Hz)
  *Configures the FlexIO I2S Tx audio format.*
- status_t FLEXIO_I2S_TransferSendDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle, flexio_i2s_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO I2S transfer using DMA.*
- status_t FLEXIO_I2S_TransferReceiveDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle, flexio_i2s_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO I2S receive using DMA.*
- void FLEXIO_I2S_TransferAbortSendDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle)
  *Aborts a FlexIO I2S transfer using DMA.*
- void FLEXIO_I2S_TransferAbortReceiveDMA (FLEXIO_I2S_Type ∗base, flexio_i2s_dma_handle_t ∗handle)
  *Aborts a FlexIO I2S receive using DMA.*

- status_t FLEXIO_I2S_TransferGetSendCountDMA (FLEXIO_I2S_Type *base, flexio_i2s_dma_handle_t *handle, size_t *count)
  *Gets the remaining bytes to be sent.*
- status_t FLEXIO_I2S_TransferGetReceiveCountDMA (FLEXIO_I2S_Type *base, flexio_i2s_dma_handle_t *handle, size_t *count)
  *Gets the remaining bytes to be received.*

#### 23.5.8.2 Data Structure Documentation

##### 23.5.8.2.1 struct _flexio_i2s_dma_handle

**Data Fields**

- dma_handle_t * dmaHandle
  *DMA handler for FlexIO I2S send.*
- uint8_t bytesPerFrame
  *Bytes in a frame.*
- uint32_t state
  *Internal state for FlexIO I2S DMA transfer.*
- flexio_i2s_dma_callback_t callback
  *Callback for users while transfer finish or error occurred.*
- void * userData
  *User callback parameter.*
- flexio_i2s_transfer_t queue [FLEXIO_I2S_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [FLEXIO_I2S_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*

#### 23.5.8.2.1.1 Field Documentation

##### 23.5.8.2.1.1.1 flexio_i2s_transfer_t flexio_i2s_dma_handle_t::queue[FLEXIO_I2S_XFER_QUEUE_SIZE]

##### 23.5.8.2.1.1.2 volatile uint8_t flexio_i2s_dma_handle_t::queueUser

### 23.5.8.3 Macro Definition Documentation

#### 23.5.8.3.1 #define FSL_FLEXIO_I2S_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))

### 23.5.8.4 Function Documentation

#### 23.5.8.4.1 void FLEXIO_I2S_TransferTxCreateHandleDMA ( FLEXIO_I2S_Type * *base,* flexio_i2s_dma_handle_t * *handle,* flexio_i2s_dma_callback_t *callback,* void * *userData,* dma_handle_t * *dmaHandle* )

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *callback* | FlexIO I2S DMA callback function called while finished a block. |
| *userData* | User parameter for callback. |
| *dmaHandle* | DMA handle for FlexIO I2S. This handle is a static value allocated by users. |

#### 23.5.8.4.2 void FLEXIO_I2S_TransferRxCreateHandleDMA ( FLEXIO_I2S_Type * *base,* flexio_i2s_dma_handle_t * *handle,* flexio_i2s_dma_callback_t *callback,* void * *userData,* dma_handle_t * *dmaHandle* )

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *callback* | FlexIO I2S DMA callback function called while finished a block. |
| *userData* | User parameter for callback. |
| *dmaHandle* | DMA handle for FlexIO I2S. This handle is a static value allocated by users. |

#### 23.5.8.4.3 void FLEXIO_I2S_TransferSetFormatDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_dma_handle_t * *handle*, flexio_i2s_format_t * *format*, uint32_t *srcClock_Hz* )

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the DMA parameter according to the format.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer |
| *format* | Pointer to FlexIO I2S audio data format structure. |
| *srcClock_Hz* | FlexIO I2S clock source frequency in Hz. It should be 0 while in slave mode. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |

#### 23.5.8.4.4 status_t FLEXIO_I2S_TransferSendDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_dma_handle_t * *handle*, flexio_i2s_transfer_t * *xfer* )

Note

This interface returns immediately after transfer initiates. Call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether FLEXIO I2S transfer finished.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a FlexIO I2S DMA send successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |
| *kStatus_TxBusy* | FlexIO I2S is busy sending data. |

#### 23.5.8.4.5 status_t FLEXIO_I2S_TransferReceiveDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_dma_handle_t * *handle*, flexio_i2s_transfer_t * *xfer* )

Note

This interface returns immediately after transfer initiates. Call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status to check whether the FlexIO I2S transfer is finished.

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a FlexIO I2S DMA receive successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |
| *kStatus_RxBusy* | FlexIO I2S is busy receiving data. |

#### 23.5.8.4.6 void FLEXIO_I2S_TransferAbortSendDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_dma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |

#### 23.5.8.4.7 void FLEXIO_I2S_TransferAbortReceiveDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_dma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |

#### 23.5.8.4.8 status_t FLEXIO_I2S_TransferGetSendCountDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_dma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |
| *count* | Bytes sent. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

#### 23.5.8.4.9 status_t FLEXIO_I2S_TransferGetReceiveCountDMA ( FLEXIO_I2S_Type * *base*, flexio_i2s_dma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | FlexIO I2S peripheral base address. |
| *handle* | FlexIO I2S DMA handle pointer. |

| | |
|---|---|
| *count* | Bytes received. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

## 23.6 FlexIO MCU Interface LCD Driver

### 23.6.1 Overview

The MCUXpresso SDK provides a peripheral driver for LCD (8080 or 6800 interface) function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO LCD driver supports both 8-bit and 16-bit data bus, 8080 and 6800 interface. User could change the macro FLEXIO_MCULCD_DATA_BUS_WIDTH to choose 8-bit data bus or 16-bit data bus.

The FlexIO LCD driver supports three kinds of data transfer:

1. Send a data array. For example, send the LCD image data to the LCD controller.
2. Send a value many times. For example, send 0 many times to clean the LCD screen.
3. Read data into a data array. For example, read image from LCD controller.

The FlexIO LCD driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FlexIO LCD initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO LCD peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO LCD functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code.

Transactional APIs support asynchronous transfer. This means that the function FLEXIO_MCULCD_-TransferNonBlocking sets up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_FLEXIO_MCULCD_Idle status.

### 23.6.2 Typical use case

#### 23.6.2.1 FlexIO LCD send/receive using functional APIs

This example shows how to send command, or write and read data using the functional APIs. The data bus is 16-bit.

```
uint16_t dataToSend[] = { ... };
uint16_t dataToReceive[] = { ... };

FLEXIO_MCULCD_Type flexioLcdDev;
flexio_MCULCD_transfer_t xfer;
flexio_MCULCD_config_t config;

FLEXIO_MCULCD_GetDefaultConfig(&config);
FLEXIO_MCULCD_Init(&flexioLcdDev, &config, 120000000);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
```

```
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command1);
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command1;
xfer.dataCount = 0; // Only send command, no data transfer.
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteDataArrayBlocking(&flexioLcdDev, dataToSend, sizeof(
      dataToSend));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
xfer.dataCount = sizeof(dataToSend);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteSameValueBlocking(&flexioLcdDev, value, 1000); //
       Send value 1000 times
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
xfer.dataAddrOrSameValue = value;
xfer.dataCount = 1000;
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command3);
FLEXIO_MCULCD_ReadDataArrayBlocking(&flexioLcdDev, dataToReceive, sizeof
      (dataToReceive));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);
```

#### 23.6.2.2 FlexIO LCD send/receive using interrupt transactional APIs

```
flexio_MCULCD_handle_t handle;
volatile bool completeFlag = false;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_handle_t *handle, status_t
       status, void *userData)
{
    if (kStatus_FLEXIO_MCULCD_Idle == status)
    {
        completeFlag = true;
    }
}

void main(void)
```

```
{
    // Init the FlexIO LCD driver.
    FLEXIO_MCULCD_Init(...);

    // Create the transactional handle.
    FLEXIO_MCULCD_TransferCreateHandle(&flexioLcdDev, &handle,
      flexioLcdCallback, NULL);

    xfer.command = command1;
    xfer.dataCount = 0; // Only send command, no data transfer.
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &xfer);

    // When only send method, it is not necessary to wait for the callback,
    // because the command is sent using a blocking method internally. The
    // command has been sent out after the function FLEXIO_MCULCD_TransferNonBlocking
    // returns.
    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
    xfer.dataCount = sizeof(dataToSend);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
    xfer.dataAddrOrSameValue = value;
    xfer.dataCount = 1000;
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command3;
    xfer.mode = kFLEXIO_MCULCD_ReadArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
    xfer.dataCount = sizeof(dataToReceive);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }
}
```

## Modules

- FlexIO DMA MCU Interface LCD Driver
- FlexIO SMARTDMA MCU Interface LCD Driver
- FlexIO eDMA MCU Interface LCD Driver

  *SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.*

## Data Structures

- struct FLEXIO_MCULCD_Type
  *Define FlexIO MCULCD access structure typedef. More...*
- struct flexio_mculcd_config_t
  *Define FlexIO MCULCD configuration structure. More...*
- struct flexio_mculcd_transfer_t
  *Define FlexIO MCULCD transfer structure. More...*
- struct flexio_mculcd_handle_t
  *Define FlexIO MCULCD handle structure. More...*

## Macros

- #define FLEXIO_MCULCD_WAIT_COMPLETE_TIME 512
  *The delay time to wait for FLEXIO transmit complete.*
- #define FLEXIO_MCULCD_DATA_BUS_WIDTH 16
  *The data bus width, must be 8 or 16.*

## Typedefs

- typedef void(* flexio_mculcd_pin_func_t )(bool set)
  *Function to set or clear the CS and RS pin.*
- typedef void(* flexio_mculcd_transfer_callback_t )(FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, status_t status, void *userData)
  *FlexIO MCULCD callback for finished transfer.*

## Enumerations

- enum _flexio_mculcd_status {
  kStatus_FLEXIO_MCULCD_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 0),
  kStatus_FLEXIO_MCULCD_Busy = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 1),
  kStatus_FLEXIO_MCULCD_Error = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 2) }
  *FlexIO LCD transfer status.*
- enum flexio_mculcd_pixel_format_t {
  kFLEXIO_MCULCD_RGB565 = 0,
  kFLEXIO_MCULCD_BGR565,
  kFLEXIO_MCULCD_RGB888,
  kFLEXIO_MCULCD_BGR888 }
  *Define FlexIO MCULCD pixel format.*
- enum flexio_mculcd_bus_t {
  kFLEXIO_MCULCD_8080,
  kFLEXIO_MCULCD_6800 }
  *Define FlexIO MCULCD bus type.*
- enum _flexio_mculcd_interrupt_enable {
  kFLEXIO_MCULCD_TxEmptyInterruptEnable = (1U << 0U),

kFLEXIO_MCULCD_RxFullInterruptEnable = (1U << 1U) }
*Define FlexIO MCULCD interrupt mask.*
- enum _flexio_mculcd_status_flags {
kFLEXIO_MCULCD_TxEmptyFlag = (1U << 0U),
kFLEXIO_MCULCD_RxFullFlag = (1U << 1U) }
*Define FlexIO MCULCD status mask.*
- enum _flexio_mculcd_dma_enable {
kFLEXIO_MCULCD_TxDmaEnable = 0x1U,
kFLEXIO_MCULCD_RxDmaEnable = 0x2U }
*Define FlexIO MCULCD DMA mask.*
- enum flexio_mculcd_transfer_mode_t {
kFLEXIO_MCULCD_ReadArray,
kFLEXIO_MCULCD_WriteArray,
kFLEXIO_MCULCD_WriteSameValue }
*Transfer mode.*

## Driver version

- #define FSL_FLEXIO_MCULCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
*FlexIO MCULCD driver version 2.0.2.*

## FlexIO MCULCD Configuration

- status_t FLEXIO_MCULCD_Init (FLEXIO_MCULCD_Type *base, flexio_mculcd_config_t *config, uint32_t srcClock_Hz)
*Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with FlexIO MCULCD configuration.*
- void FLEXIO_MCULCD_Deinit (FLEXIO_MCULCD_Type *base)
*Resets the FLEXIO_MCULCD timer and shifter configuration.*
- void FLEXIO_MCULCD_GetDefaultConfig (flexio_mculcd_config_t *config)
*Gets the default configuration to configure the FlexIO MCULCD.*

## Status

- uint32_t FLEXIO_MCULCD_GetStatusFlags (FLEXIO_MCULCD_Type *base)
*Gets FlexIO MCULCD status flags.*
- void FLEXIO_MCULCD_ClearStatusFlags (FLEXIO_MCULCD_Type *base, uint32_t mask)
*Clears FlexIO MCULCD status flags.*

## Interrupts

- void FLEXIO_MCULCD_EnableInterrupts (FLEXIO_MCULCD_Type *base, uint32_t mask)
*Enables the FlexIO MCULCD interrupt.*
- void FLEXIO_MCULCD_DisableInterrupts (FLEXIO_MCULCD_Type *base, uint32_t mask)

*Disables the FlexIO MCULCD interrupt.*

## DMA Control

- static void FLEXIO_MCULCD_EnableTxDMA (FLEXIO_MCULCD_Type *base, bool enable)
  *Enables/disables the FlexIO MCULCD transmit DMA.*
- static void FLEXIO_MCULCD_EnableRxDMA (FLEXIO_MCULCD_Type *base, bool enable)
  *Enables/disables the FlexIO MCULCD receive DMA.*
- static uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress (FLEXIO_MCULCD_Type *base)
  *Gets the FlexIO MCULCD transmit data register address.*
- static uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress (FLEXIO_MCULCD_Type *base)
  *Gets the FlexIO MCULCD receive data register address.*

## Bus Operations

- status_t FLEXIO_MCULCD_SetBaudRate (FLEXIO_MCULCD_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  *Set desired baud rate.*
- void FLEXIO_MCULCD_SetSingleBeatWriteConfig (FLEXIO_MCULCD_Type *base)
  *Configures the FLEXIO MCULCD to multiple beats write mode.*
- void FLEXIO_MCULCD_ClearSingleBeatWriteConfig (FLEXIO_MCULCD_Type *base)
  *Clear the FLEXIO MCULCD multiple beats write mode configuration.*
- void FLEXIO_MCULCD_SetSingleBeatReadConfig (FLEXIO_MCULCD_Type *base)
  *Configures the FLEXIO MCULCD to multiple beats read mode.*
- void FLEXIO_MCULCD_ClearSingleBeatReadConfig (FLEXIO_MCULCD_Type *base)
  *Clear the FLEXIO MCULCD multiple beats read mode configuration.*
- void FLEXIO_MCULCD_SetMultiBeatsWriteConfig (FLEXIO_MCULCD_Type *base)
  *Configures the FLEXIO MCULCD to multiple beats write mode.*
- void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig (FLEXIO_MCULCD_Type *base)
  *Clear the FLEXIO MCULCD multiple beats write mode configuration.*
- void FLEXIO_MCULCD_SetMultiBeatsReadConfig (FLEXIO_MCULCD_Type *base)
  *Configures the FLEXIO MCULCD to multiple beats read mode.*
- void FLEXIO_MCULCD_ClearMultiBeatsReadConfig (FLEXIO_MCULCD_Type *base)
  *Clear the FLEXIO MCULCD multiple beats read mode configuration.*
- static void FLEXIO_MCULCD_Enable (FLEXIO_MCULCD_Type *base, bool enable)
  *Enables/disables the FlexIO MCULCD module operation.*
- uint32_t FLEXIO_MCULCD_ReadData (FLEXIO_MCULCD_Type *base)
  *Read data from the FLEXIO MCULCD RX shifter buffer.*
- static void FLEXIO_MCULCD_WriteData (FLEXIO_MCULCD_Type *base, uint32_t data)
  *Write data into the FLEXIO MCULCD TX shifter buffer.*
- static void FLEXIO_MCULCD_StartTransfer (FLEXIO_MCULCD_Type *base)
  *Assert the nCS to start transfer.*
- static void FLEXIO_MCULCD_StopTransfer (FLEXIO_MCULCD_Type *base)
  *De-assert the nCS to stop transfer.*
- void FLEXIO_MCULCD_WaitTransmitComplete (void)
  *Wait for transmit data send out finished.*

- void FLEXIO_MCULCD_WriteCommandBlocking (FLEXIO_MCULCD_Type *base, uint32_t command)
  *Send command in blocking way.*
- void FLEXIO_MCULCD_WriteDataArrayBlocking (FLEXIO_MCULCD_Type *base, void *data, size_t size)
  *Send data array in blocking way.*
- void FLEXIO_MCULCD_ReadDataArrayBlocking (FLEXIO_MCULCD_Type *base, void *data, size_t size)
  *Read data into array in blocking way.*
- void FLEXIO_MCULCD_WriteSameValueBlocking (FLEXIO_MCULCD_Type *base, uint32_t sameValue, size_t size)
  *Send the same value many times in blocking way.*
- void FLEXIO_MCULCD_TransferBlocking (FLEXIO_MCULCD_Type *base, flexio_mculcd_transfer_t *xfer)
  *Performs a polling transfer.*

### Transactional

- status_t FLEXIO_MCULCD_TransferCreateHandle (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, flexio_mculcd_transfer_callback_t callback, void *userData)
  *Initializes the FlexIO MCULCD handle, which is used in transactional functions.*
- status_t FLEXIO_MCULCD_TransferNonBlocking (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, flexio_mculcd_transfer_t *xfer)
  *Transfer data using IRQ.*
- void FLEXIO_MCULCD_TransferAbort (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle)
  *Aborts the data transfer, which used IRQ.*
- status_t FLEXIO_MCULCD_TransferGetCount (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, size_t *count)
  *Gets the data transfer status which used IRQ.*
- void FLEXIO_MCULCD_TransferHandleIRQ (void *base, void *handle)
  *FlexIO MCULCD IRQ handler function.*

## 23.6.3 Data Structure Documentation

### 23.6.3.1 struct FLEXIO_MCULCD_Type

### Data Fields

- FLEXIO_Type * flexioBase
  *FlexIO base pointer.*
- flexio_mculcd_bus_t busType
  *The bus type, 8080 or 6800.*
- uint8_t dataPinStartIndex
  *Start index of the data pin, the FlexIO pin dataPinStartIndex to (dataPinStartIndex + FLEXIO_MCULCD_DATA_BUS_WIDTH -1) will be used for data transfer.*

- uint8_t ENWRPinIndex
  *Pin select for WR(8080 mode), EN(6800 mode).*
- uint8_t RDPinIndex
  *Pin select for RD(8080 mode), not used in 6800 mode.*
- uint8_t txShifterStartIndex
  *Start index of shifters used for data write, it must be 0 or 4.*
- uint8_t txShifterEndIndex
  *End index of shifters used for data write.*
- uint8_t rxShifterStartIndex
  *Start index of shifters used for data read.*
- uint8_t rxShifterEndIndex
  *End index of shifters used for data read, it must be 3 or 7.*
- uint8_t timerIndex
  *Timer index used in FlexIO MCULCD.*
- flexio_mculcd_pin_func_t setCSPin
  *Function to set or clear the CS pin.*
- flexio_mculcd_pin_func_t setRSPin
  *Function to set or clear the RS pin.*
- flexio_mculcd_pin_func_t setRDWRPin
  *Function to set or clear the RD/WR pin, only used in 6800 mode.*

#### 23.6.3.1.0.1 Field Documentation

##### 23.6.3.1.0.1.1 FLEXIO_Type* FLEXIO_MCULCD_Type::flexioBase

##### 23.6.3.1.0.1.2 flexio_mculcd_bus_t FLEXIO_MCULCD_Type::busType

##### 23.6.3.1.0.1.3 uint8_t FLEXIO_MCULCD_Type::dataPinStartIndex

Only support data bus width 8 and 16.

#### 23.6.3.1.0.1.4 uint8_t FLEXIO_MCULCD_Type::ENWRPinIndex

#### 23.6.3.1.0.1.5 uint8_t FLEXIO_MCULCD_Type::RDPinIndex

#### 23.6.3.1.0.1.6 uint8_t FLEXIO_MCULCD_Type::txShifterStartIndex

#### 23.6.3.1.0.1.7 uint8_t FLEXIO_MCULCD_Type::txShifterEndIndex

#### 23.6.3.1.0.1.8 uint8_t FLEXIO_MCULCD_Type::rxShifterStartIndex

#### 23.6.3.1.0.1.9 uint8_t FLEXIO_MCULCD_Type::rxShifterEndIndex

#### 23.6.3.1.0.1.10 uint8_t FLEXIO_MCULCD_Type::timerIndex

#### 23.6.3.1.0.1.11 flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setCSPin

#### 23.6.3.1.0.1.12 flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRSPin

#### 23.6.3.1.0.1.13 flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRDWRPin

### 23.6.3.2 struct flexio_mculcd_config_t

### Data Fields

- bool enable
  *Enable/disable FlexIO MCULCD after configuration.*
- bool enableInDoze
  *Enable/disable FlexIO operation in doze mode.*
- bool enableInDebug
  *Enable/disable FlexIO operation in debug mode.*
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  *fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32_t baudRate_Bps
  *Baud rate in Bps.*

#### 23.6.3.2.0.2 Field Documentation

##### 23.6.3.2.0.2.1 bool flexio_mculcd_config_t::enable

##### 23.6.3.2.0.2.2 bool flexio_mculcd_config_t::enableInDoze

##### 23.6.3.2.0.2.3 bool flexio_mculcd_config_t::enableInDebug

##### 23.6.3.2.0.2.4 bool flexio_mculcd_config_t::enableFastAccess

##### 23.6.3.2.0.2.5 uint32_t flexio_mculcd_config_t::baudRate_Bps

### 23.6.3.3 struct flexio_mculcd_transfer_t

#### Data Fields

- uint32_t command
  *Command to send.*
- flexio_mculcd_transfer_mode_t mode
  *Transfer mode.*
- uint32_t dataAddrOrSameValue
  *When sending the same value for many times, this is the value to send.*
- size_t dataSize
  *How many bytes to transfer.*

#### 23.6.3.3.0.3 Field Documentation

##### 23.6.3.3.0.3.1 uint32_t flexio_mculcd_transfer_t::command

##### 23.6.3.3.0.3.2 flexio_mculcd_transfer_mode_t flexio_mculcd_transfer_t::mode

##### 23.6.3.3.0.3.3 uint32_t flexio_mculcd_transfer_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

##### 23.6.3.3.0.3.4 size_t flexio_mculcd_transfer_t::dataSize

### 23.6.3.4 struct _flexio_mculcd_handle

typedef for flexio_mculcd_handle_t in advance.

#### Data Fields

- uint32_t dataAddrOrSameValue
  *When sending the same value for many times, this is the value to send.*
- size_t dataCount
  *Total count to be transferred.*
- volatile size_t remainingCount
  *Remaining count to transfer.*
- volatile uint32_t state

*FlexIO MCULCD internal state.*
- flexio_mculcd_transfer_callback_t completionCallback
  *FlexIO MCULCD transfer completed callback.*
- void * userData
  *Callback parameter.*

#### 23.6.3.4.0.4 Field Documentation

##### 23.6.3.4.0.4.1 uint32_t flexio_mculcd_handle_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

##### 23.6.3.4.0.4.2 size_t flexio_mculcd_handle_t::dataCount

##### 23.6.3.4.0.4.3 volatile size_t flexio_mculcd_handle_t::remainingCount

##### 23.6.3.4.0.4.4 volatile uint32_t flexio_mculcd_handle_t::state

##### 23.6.3.4.0.4.5 flexio_mculcd_transfer_callback_t flexio_mculcd_handle_t::completionCallback

##### 23.6.3.4.0.4.6 void* flexio_mculcd_handle_t::userData

### 23.6.4 Macro Definition Documentation

#### 23.6.4.1 #define FSL_FLEXIO_MCULCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

#### 23.6.4.2 #define FLEXIO_MCULCD_WAIT_COMPLETE_TIME 512

Currently there is no method to detect whether the data has been sent out from the shifter, so the driver use a software delay for this. When the data is written to shifter buffer, the driver call the delay function to wait for the data shift out. If this value is too small, then the last few bytes might be lost when writing data using interrupt method or DMA method.

### 23.6.5 Typedef Documentation

#### 23.6.5.1 typedef void(* flexio_mculcd_pin_func_t)(bool set)

#### 23.6.5.2 typedef void(* flexio_mculcd_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, status_t status, void *userData)

When transfer finished, the callback function is called and returns the `status` as kStatus_FLEXIO_MCULCD_Idle.

### 23.6.6 Enumeration Type Documentation

#### 23.6.6.1 enum _flexio_mculcd_status

Enumerator

***kStatus_FLEXIO_MCULCD_Idle*** FlexIO LCD is idle.
***kStatus_FLEXIO_MCULCD_Busy*** FlexIO LCD is busy.
***kStatus_FLEXIO_MCULCD_Error*** FlexIO LCD error occurred.

#### 23.6.6.2 enum flexio_mculcd_pixel_format_t

Enumerator

***kFLEXIO_MCULCD_RGB565*** RGB565, 16-bit.
***kFLEXIO_MCULCD_BGR565*** BGR565, 16-bit.
***kFLEXIO_MCULCD_RGB888*** RGB888, 24-bit.
***kFLEXIO_MCULCD_BGR888*** BGR888, 24-bit.

#### 23.6.6.3 enum flexio_mculcd_bus_t

Enumerator

***kFLEXIO_MCULCD_8080*** Using Intel 8080 bus.
***kFLEXIO_MCULCD_6800*** Using Motorola 6800 bus.

#### 23.6.6.4 enum _flexio_mculcd_interrupt_enable

Enumerator

***kFLEXIO_MCULCD_TxEmptyInterruptEnable*** Transmit buffer empty interrupt enable.
***kFLEXIO_MCULCD_RxFullInterruptEnable*** Receive buffer full interrupt enable.

#### 23.6.6.5 enum _flexio_mculcd_status_flags

Enumerator

***kFLEXIO_MCULCD_TxEmptyFlag*** Transmit buffer empty flag.
***kFLEXIO_MCULCD_RxFullFlag*** Receive buffer full flag.

#### 23.6.6.6 enum _flexio_mculcd_dma_enable

Enumerator

***kFLEXIO_MCULCD_TxDmaEnable*** Tx DMA request source.
***kFLEXIO_MCULCD_RxDmaEnable*** Rx DMA request source.

#### 23.6.6.7 enum flexio_mculcd_transfer_mode_t

Enumerator

***kFLEXIO_MCULCD_ReadArray*** Read data into an array.
***kFLEXIO_MCULCD_WriteArray*** Write data from an array.
***kFLEXIO_MCULCD_WriteSameValue*** Write the same value many times.

### 23.6.7 Function Documentation

#### 23.6.7.1 status_t FLEXIO_MCULCD_Init ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_config_t * *config*, uint32_t *srcClock_Hz* )

The configuration structure can be filled by the user, or be set with default values by the FLEXIO_MCULCD_GetDefaultConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *config* | Pointer to the flexio_mculcd_config_t structure. |
| *srcClock_Hz* | FlexIO source clock in Hz. |

Return values

| | |
|---|---|
| *kStatus_Success* | Initialization success. |
| *kStatus_InvalidArgument* | Initialization failed because of invalid argument. |

#### 23.6.7.2 void FLEXIO_MCULCD_Deinit ( FLEXIO_MCULCD_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

#### 23.6.7.3 void FLEXIO_MCULCD_GetDefaultConfig ( flexio_mculcd_config_t * *config* )

The default configuration value is:

```
*  config->enable = true;
*  config->enableInDoze = false;
*  config->enableInDebug = true;
*  config->enableFastAccess = true;
*  config->baudRate_Bps = 96000000U;
*
```

Parameters

| | |
|---|---|
| *Config* | Pointer to the flexio_mculcd_config_t structure. |

#### 23.6.7.4 uint32_t FLEXIO_MCULCD_GetStatusFlags ( FLEXIO_MCULCD_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |

Returns

status flag; OR'ed value or the _flexio_mculcd_status_flags.

Note

Don't use this function with DMA APIs.

#### 23.6.7.5 void FLEXIO_MCULCD_ClearStatusFlags ( FLEXIO_MCULCD_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *mask* | Status to clear, it is the OR'ed value of _flexio_mculcd_status_flags. |

Note

Don't use this function with DMA APIs.

#### 23.6.7.6 void FLEXIO_MCULCD_EnableInterrupts ( FLEXIO_MCULCD_Type * *base*, uint32_t *mask* )

This function enables the FlexIO MCULCD interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *mask* | Interrupts to enable, it is the OR'ed value of _flexio_mculcd_interrupt_enable. |

#### 23.6.7.7 void FLEXIO_MCULCD_DisableInterrupts ( FLEXIO_MCULCD_Type * *base*, uint32_t *mask* )

This function disables the FlexIO MCULCD interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *mask* | Interrupts to disable, it is the OR'ed value of _flexio_mculcd_interrupt_enable. |

#### 23.6.7.8 static void FLEXIO_MCULCD_EnableTxDMA ( FLEXIO_MCULCD_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *mask* | MCULCD DMA source. |

| | |
|---|---|
| *enable* | True means enable DMA, false means disable DMA. |

#### 23.6.7.9 static void FLEXIO_MCULCD_EnableRxDMA ( FLEXIO_MCULCD_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *mask* | MCULCD DMA source. |
| *enable* | True means enable DMA, false means disable DMA. |

#### 23.6.7.10 static uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress ( FLEXIO_MCULCD_Type * *base* ) [inline], [static]

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |

Returns

FlexIO MCULCD transmit data register address.

#### 23.6.7.11 static uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress ( FLEXIO_MCULCD_Type * *base* ) [inline], [static]

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |

Returns

FlexIO MCULCD receive data register address.

#### 23.6.7.12 status_t FLEXIO_MCULCD_SetBaudRate ( FLEXIO_MCULCD_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *baudRate_Bps* | Desired baud rate. |
| *srcClock_Hz* | FLEXIO clock frequency in Hz. |

Return values

| | |
|---|---|
| *kStatus_Success* | Set successfully. |
| *kStatus_InvalidArgument* | Could not set the baud rate. |

#### 23.6.7.13 void FLEXIO_MCULCD_SetSingleBeatWriteConfig ( FLEXIO_MCULCD_Type * *base* )

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by FLEXIO_MCULCD_ClearSingleBeatWriteConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.14 void FLEXIO_MCULCD_ClearSingleBeatWriteConfig ( FLEXIO_MCULCD_Type * *base* )

Clear the write configuration set by FLEXIO_MCULCD_SetSingleBeatWriteConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.15 void FLEXIO_MCULCD_SetSingleBeatReadConfig ( FLEXIO_MCULCD_Type * *base* )

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD_ClearSingleBeatReadConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.16 void FLEXIO_MCULCD_ClearSingleBeatReadConfig ( FLEXIO_MCULCD_Type * *base* )

Clear the read configuration set by FLEXIO_MCULCD_SetSingleBeatReadConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.17 void FLEXIO_MCULCD_SetMultiBeatsWriteConfig ( FLEXIO_MCULCD_Type * *base* )

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by FLEXIO_MCULCD_ClearMultBeatsWriteConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.18 void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig ( FLEXIO_MCULCD_Type * *base* )

Clear the write configuration set by FLEXIO_MCULCD_SetMultBeatsWriteConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.19 void FLEXIO_MCULCD_SetMultiBeatsReadConfig ( FLEXIO_MCULCD_Type * *base* )

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD-_ClearMultBeatsReadConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.20 void FLEXIO_MCULCD_ClearMultiBeatsReadConfig ( FLEXIO_MCULCD_-Type * *base* )

Clear the read configuration set by FLEXIO_MCULCD_SetMultBeatsReadConfig.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |

Note

This is an internal used function, upper layer should not use.

#### 23.6.7.21 static void FLEXIO_MCULCD_Enable ( FLEXIO_MCULCD_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type. |
| *enable* | True to enable, false does not have any effect. |

### 23.6.7.22 uint32_t FLEXIO_MCULCD_ReadData ( FLEXIO_MCULCD_Type * *base* )

Read data from the RX shift buffer directly, it does no check whether the buffer is empty or not.

If the data bus width is 8-bit:

```
* uint8_t value;
* value = (uint8_t)FLEXIO_MCULCD_ReadData(base);
*
```

If the data bus width is 16-bit:

```
* uint16_t value;
* value = (uint16_t)FLEXIO_MCULCD_ReadData(base);
*
```

Note

This function returns the RX shifter buffer value (32-bit) directly. The return value should be converted according to data bus width.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |

Returns

The data read out.

Note

Don't use this function with DMA APIs.

### 23.6.7.23 static void FLEXIO_MCULCD_WriteData ( FLEXIO_MCULCD_Type * *base*, uint32_t *data* ) [inline], [static]

Write data into the TX shift buffer directly, it does no check whether the buffer is full or not.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *data* | The data to write. |

Note

Don't use this function with DMA APIs.

#### 23.6.7.24 static void FLEXIO_MCULCD_StartTransfer ( FLEXIO_MCULCD_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |

#### 23.6.7.25 static void FLEXIO_MCULCD_StopTransfer ( FLEXIO_MCULCD_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |

#### 23.6.7.26 void FLEXIO_MCULCD_WaitTransmitComplete ( void )

Currently there is no effective method to wait for the data send out from the shiter, so here use a while loop to wait.

Note

This is an internal used function.

#### 23.6.7.27 void FLEXIO_MCULCD_WriteCommandBlocking ( FLEXIO_MCULCD_Type * *base*, uint32_t *command* )

This function sends the command and returns when the command has been sent out.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *command* | The command to send. |

#### 23.6.7.28 void FLEXIO_MCULCD_WriteDataArrayBlocking ( FLEXIO_MCULCD_Type * *base*, void * *data*, size_t *size* )

This function sends the data array and returns when the data sent out.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *data* | The data array to send. |
| *size* | How many bytes to write. |

#### 23.6.7.29 void FLEXIO_MCULCD_ReadDataArrayBlocking ( FLEXIO_MCULCD_Type * *base*, void * *data*, size_t *size* )

This function reads the data into array and returns when the data read finished.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *data* | The array to save the data. |
| *size* | How many bytes to read. |

#### 23.6.7.30 void FLEXIO_MCULCD_WriteSameValueBlocking ( FLEXIO_MCULCD_Type * *base*, uint32_t *sameValue*, size_t *size* )

This function sends the same value many times. It could be used to clear the LCD screen. If the data bus width is 8, this function will send LSB 8 bits of `sameValue` for `size` times. If the data bus is 16, this function will send LSB 16 bits of `sameValue` for `size` / 2 times.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *sameValue* | The same value to send. |
| *size* | How many bytes to send. |

### 23.6.7.31 void FLEXIO_MCULCD_TransferBlocking ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_transfer_t * *xfer* )

Note

The API does not return until the transfer finished.

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *xfer* | pointer to flexio_mculcd_transfer_t structure. |

### 23.6.7.32 status_t FLEXIO_MCULCD_TransferCreateHandle ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, flexio_mculcd_transfer_callback_t *callback*, void * *userData* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to the flexio_mculcd_handle_t structure to store the transfer state. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO type/handle/ISR table out of range. |

### 23.6.7.33 status_t FLEXIO_MCULCD_TransferNonBlocking ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer* )

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to the flexio_mculcd_handle_t structure to store the transfer state. |
| *xfer* | FlexIO MCULCD transfer structure. See flexio_mculcd_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_MCULCD_Busy* | MCULCD is busy with another transfer. |

#### 23.6.7.34 void FLEXIO_MCULCD_TransferAbort ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to the flexio_mculcd_handle_t structure to store the transfer state. |

#### 23.6.7.35 status_t FLEXIO_MCULCD_TransferGetCount ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to the flexio_mculcd_handle_t structure to store the transfer state. |
| *count* | How many bytes transferred so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_Success* | Get the transferred count Successfully. |
| *kStatus_NoTransferInProgress* | No transfer in process. |

#### 23.6.7.36 void FLEXIO_MCULCD_TransferHandleIRQ ( void * *base*, void * *handle* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to the flexio_mculcd_handle_t structure to store the transfer state. |

### 23.6.8 FlexIO eDMA MCU Interface LCD Driver

SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.

#### 23.6.8.1 Overview

Note

eDMA transactional functions use multiple beats method for better performance, in contrast, the blocking functions and interupt functions use single beat method. The function FLEXIO_MCULCD_ReadData, FLEXIO_MCULCD_WriteData, FLEXIO_MCULCD_GetStatusFlags, and FLEXIO_MCULCD_ClearStatusFlags are only used for single beat case, so don't use these functions to work together with eDMA functions.

```
flexio_MCULCD_edma_handle_t handle;
volatile bool completeFlag = false;
edma_handle_t rxEdmaHandle;
edma_handle_t txEdmaHandle;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_edma_handle_t *handle,
      status_t status, void *userData)
{
    if (kStatus_FLEXIO_MCULCD_Idle == status)
    {
        completeFlag = true;
    }
}

void main(void)
{
    // Create the edma Handle.
    EDMA_CreateHandle(&rxEdmaHandle, DMA0, channel);
    EDMA_CreateHandle(&txEdmaHandle, DMA0, channel);

    // Configure the DMAMUX.
    // ...
    // rxEdmaHandle should use the last FlexIO RX shifters as DMA requst source.
    // txEdmaHandle should use the first FlexIO TX shifters as DMA requst source.

    // Init the FlexIO LCD driver.
    FLEXIO_MCULCD_Init(...);

    // Create the transactional handle.
    FLEXIO_MCULCD_TransferCreateHandleEDMA(&flexioLcdDev, &handle,
      flexioLcdCallback, NULL, &txEdmaHandle, &rxEdmaHandle);

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
    xfer.dataCount = sizeof(dataToSend);
    completeFlag = false;
    FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
```

```
    xfer.dataAddrOrSameValue = value;
    xfer.dataCount = 1000;
    completeFlag = false;
    FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command3;
    xfer.mode = kFLEXIO_MCULCD_ReadArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
    xfer.dataCount = sizeof(dataToReceive);
    completeFlag = false;
    FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }
}
```

## Data Structures

- struct flexio_mculcd_edma_handle_t
  *FlexIO MCULCD eDMA transfer handle, users should not touch the content of the handle. More...*

## Macros

- #define FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *FlexIO MCULCD EDMA driver version 2.0.2.*

## Typedefs

- typedef void(∗ flexio_mculcd_edma_transfer_callback_t )(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_edma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO MCULCD master callback for transfer complete.*

## eDMA Transactional

- status_t FLEXIO_MCULCD_TransferCreateHandleEDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_edma_handle_t ∗handle, flexio_mculcd_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗txDmaHandle, edma_handle_t ∗rxDmaHandle)
  *Initializes the FLEXO MCULCD master eDMA handle.*
- status_t FLEXIO_MCULCD_TransferEDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_edma_handle_t ∗handle, flexio_mculcd_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO MCULCD transfer using eDMA.*
- void FLEXIO_MCULCD_TransferAbortEDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_edma_handle_t ∗handle)
  *Aborts a FlexIO MCULCD transfer using eDMA.*
- status_t FLEXIO_MCULCD_TransferGetCountEDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_edma_handle_t ∗handle, size_t ∗count)

*Gets the remaining bytes for FlexIO MCULCD eDMA transfer.*

### 23.6.8.2 Data Structure Documentation

#### 23.6.8.2.1 struct _flexio_mculcd_edma_handle

typedef for flexio_mculcd_edma_handle_t in advance.

**Data Fields**

- FLEXIO_MCULCD_Type * base
  *Pointer to the FLEXIO_MCULCD_Type.*
- uint8_t txShifterNum
  *Number of shifters used for TX.*
- uint8_t rxShifterNum
  *Number of shifters used for RX.*
- uint32_t minorLoopBytes
  *eDMA transfer minor loop bytes.*
- edma_modulo_t txEdmaModulo
  *Modulo value for the FlexIO shifter buffer access.*
- edma_modulo_t rxEdmaModulo
  *Modulo value for the FlexIO shifter buffer access.*
- uint32_t dataAddrOrSameValue
  *When sending the same value for many times, this is the value to send.*
- size_t dataCount
  *Total count to be transferred.*
- volatile size_t remainingCount
  *Remaining count still not transfered.*
- volatile uint32_t state
  *FlexIO MCULCD driver internal state.*
- edma_handle_t * txDmaHandle
  *DMA handle for MCULCD TX.*
- edma_handle_t * rxDmaHandle
  *DMA handle for MCULCD RX.*
- flexio_mculcd_edma_transfer_callback_t completionCallback
  *Callback for MCULCD DMA transfer.*
- void * userData
  *User Data for MCULCD DMA callback.*

#### 23.6.8.2.1.1 Field Documentation

##### 23.6.8.2.1.1.1 FLEXIO_MCULCD_Type* flexio_mculcd_edma_handle_t::base

##### 23.6.8.2.1.1.2 uint8_t flexio_mculcd_edma_handle_t::txShifterNum

##### 23.6.8.2.1.1.3 uint8_t flexio_mculcd_edma_handle_t::rxShifterNum

##### 23.6.8.2.1.1.4 uint32_t flexio_mculcd_edma_handle_t::minorLoopBytes

##### 23.6.8.2.1.1.5 edma_modulo_t flexio_mculcd_edma_handle_t::txEdmaModulo

##### 23.6.8.2.1.1.6 edma_modulo_t flexio_mculcd_edma_handle_t::rxEdmaModulo

##### 23.6.8.2.1.1.7 uint32_t flexio_mculcd_edma_handle_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

##### 23.6.8.2.1.1.8 size_t flexio_mculcd_edma_handle_t::dataCount

##### 23.6.8.2.1.1.9 volatile size_t flexio_mculcd_edma_handle_t::remainingCount

##### 23.6.8.2.1.1.10 volatile uint32_t flexio_mculcd_edma_handle_t::state

### 23.6.8.3 Macro Definition Documentation

#### 23.6.8.3.1 #define FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

### 23.6.8.4 Typedef Documentation

#### 23.6.8.4.1 typedef void(* flexio_mculcd_edma_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculcd_edma_handle_t *handle, status_t status, void *userData)

When transfer finished, the callback function is called and returns the `status` as kStatus_FLEXIO_MCULCD_Idle.

### 23.6.8.5 Function Documentation

#### 23.6.8.5.1 status_t FLEXIO_MCULCD_TransferCreateHandleEDMA ( FLEXIO_MCULCD_Type * *base,* flexio_mculcd_edma_handle_t * *handle,* flexio_mculcd_edma_transfer_callback_t *callback,* void * *userData,* edma_handle_t * *txDmaHandle,* edma_handle_t * *rxDmaHandle* )

This function initializes the FLEXO MCULCD master eDMA handle which can be used for other FLEXO MCULCD transactional APIs. For a specified FLEXO MCULCD instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to flexio_mculcd_edma_handle_t structure to store the transfer state. |
| *callback* | MCULCD transfer complete callback, NULL means no callback. |
| *userData* | callback function parameter. |
| *txDmaHandle* | User requested eDMA handle for FlexIO MCULCD eDMA TX, the DMA request source of this handle should be the first of TX shifters. |
| *rxDmaHandle* | User requested eDMA handle for FlexIO MCULCD eDMA RX, the DMA request source of this handle should be the last of RX shifters. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |

#### 23.6.8.5.2 status_t FLEXIO_MCULCD_TransferEDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_edma_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer* )

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

1. Use the transfer completed callback;
2. Polling function FLEXIO_MCULCD_GetTransferCountEDMA

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | pointer to flexio_mculcd_edma_handle_t structure to store the transfer state. |
| *xfer* | Pointer to FlexIO MCULCD transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_MCULCD_Busy* | FlexIO MCULCD is not idle, it is running another transfer. |

#### 23.6.8.5.3 void FLEXIO_MCULCD_TransferAbortEDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_edma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | FlexIO MCULCD eDMA handle pointer. |

#### 23.6.8.5.4 status_t FLEXIO_MCULCD_TransferGetCountEDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | FlexIO MCULCD eDMA handle pointer. |
| *count* | Number of count transferred so far by the eDMA transaction. |

Return values

| | |
|---|---|
| *kStatus_Success* | Get the transferred count Successfully. |
| *kStatus_NoTransferInProgress* | No transfer in process. |

## 23.6.9 FlexIO DMA MCU Interface LCD Driver

### 23.6.9.1 Overview

#### Data Structures

- struct flexio_mculcd_dma_handle_t
  *FlexIO MCULCD DMA transfer handle, users should not touch the content of the handle. More...*

#### Macros

- #define FSL_FLEXIO_MCULCD_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *FlexIO MCULCD DMA driver version 2.0.0.*

#### Typedefs

- typedef void(∗ flexio_mculcd_dma_transfer_callback_t )(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO MCULCD master callback for transfer complete.*

#### DMA Transactional

- status_t FLEXIO_MCULCD_TransferCreateHandleDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, flexio_mculcd_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  *Initializes the FLEXO MCULCD master DMA handle.*
- status_t FLEXIO_MCULCD_TransferDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, flexio_mculcd_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO MCULCD transfer using DMA.*
- void FLEXIO_MCULCD_TransferAbortDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle)
  *Aborts a FlexIO MCULCD transfer using DMA.*
- status_t FLEXIO_MCULCD_TransferGetCountDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, size_t ∗count)
  *Gets the remaining bytes for FlexIO MCULCD DMA transfer.*

### 23.6.9.2 Data Structure Documentation

#### 23.6.9.2.1 struct _flexio_mculcd_dma_handle

typedef for flexio_mculcd_dma_handle_t in advance.

#### Data Fields

- FLEXIO_MCULCD_Type ∗ base

*Pointer to the FLEXIO_MCULCD_Type.*

- uint8_t txShifterNum
  *Number of shifters used for TX.*
- uint8_t rxShifterNum
  *Number of shifters used for RX.*
- dma_trigger_burst_t txTriggerBurst
  *Trigger burst setting for TX.*
- dma_trigger_burst_t rxTriggerBurst
  *Trigger burst setting for RX.*
- uint32_t burstBytes
  *DMA transfer bytes per burst.*
- uint32_t dataAddrOrSameValue
  *When sending the same value for many times, this is the value to send.*
- size_t dataCount
  *Total count to be transferred.*
- volatile size_t remainingCount
  *Remaining count still not transfered.*
- volatile uint32_t state
  *FlexIO MCULCD driver internal state.*
- dma_handle_t * txDmaHandle
  *DMA handle for MCULCD TX.*
- dma_handle_t * rxDmaHandle
  *DMA handle for MCULCD RX.*
- flexio_mculcd_dma_transfer_callback_t completionCallback
  *Callback for MCULCD DMA transfer.*
- void * userData
  *User Data for MCULCD DMA callback.*

#### 23.6.9.2.1.1 Field Documentation

##### 23.6.9.2.1.1.1 FLEXIO_MCULCD_Type* flexio_mculcd_dma_handle_t::base

##### 23.6.9.2.1.1.2 uint8_t flexio_mculcd_dma_handle_t::txShifterNum

##### 23.6.9.2.1.1.3 uint8_t flexio_mculcd_dma_handle_t::rxShifterNum

##### 23.6.9.2.1.1.4 dma_trigger_burst_t flexio_mculcd_dma_handle_t::txTriggerBurst

##### 23.6.9.2.1.1.5 dma_trigger_burst_t flexio_mculcd_dma_handle_t::rxTriggerBurst

##### 23.6.9.2.1.1.6 uint32_t flexio_mculcd_dma_handle_t::burstBytes

##### 23.6.9.2.1.1.7 uint32_t flexio_mculcd_dma_handle_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

#### 23.6.9.2.1.1.8 size_t flexio_mculcd_dma_handle_t::dataCount

#### 23.6.9.2.1.1.9 volatile size_t flexio_mculcd_dma_handle_t::remainingCount

#### 23.6.9.2.1.1.10 volatile uint32_t flexio_mculcd_dma_handle_t::state

### 23.6.9.3 Macro Definition Documentation

#### 23.6.9.3.1 #define FSL_FLEXIO_MCULCD_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

### 23.6.9.4 Typedef Documentation

#### 23.6.9.4.1 typedef void(∗ flexio_mculcd_dma_transfer_callback_t)(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, status_t status, void ∗userData)

When transfer finished, the callback function is called and returns the `status` as kStatus_FLEXIO_MCULCD_Idle.

### 23.6.9.5 Function Documentation

#### 23.6.9.5.1 status_t FLEXIO_MCULCD_TransferCreateHandleDMA ( FLEXIO_MCULCD_Type ∗ *base,* flexio_mculcd_dma_handle_t ∗ *handle,* flexio_mculcd_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *txDmaHandle,* dma_handle_t ∗ *rxDmaHandle* )

This function initializes the FLEXO MCULCD master DMA handle which can be used for other FLEXO MCULCD transactional APIs. For a specified FLEXO MCULCD instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to flexio_mculcd_dma_handle_t structure to store the transfer state. |
| *callback* | MCULCD transfer complete callback, NULL means no callback. |
| *userData* | callback function parameter. |
| *txDmaHandle* | User requested DMA handle for FlexIO MCULCD DMA TX, the DMA request source of this handle should be the first of TX shifters. |

| | |
|---|---|
| *rxDmaHandle* | User requested DMA handle for FlexIO MCULCD DMA RX, the DMA request source of this handle should be the last of RX shifters. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |

#### 23.6.9.5.2 status_t FLEXIO_MCULCD_TransferDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_dma_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer* )

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

1. Use the transfer completed callback;
2. Polling function FLEXIO_MCULCD_GetTransferCountDMA

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | pointer to flexio_mculcd_dma_handle_t structure to store the transfer state. |
| *xfer* | Pointer to FlexIO MCULCD transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_MCULCD_Busy* | FlexIO MCULCD is not idle, it is running another transfer. |

#### 23.6.9.5.3 void FLEXIO_MCULCD_TransferAbortDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_dma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | FlexIO MCULCD DMA handle pointer. |

#### 23.6.9.5.4 status_t FLEXIO_MCULCD_TransferGetCountDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_dma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | FlexIO MCULCD DMA handle pointer. |
| *count* | Number of count transferred so far by the DMA transaction. |

Return values

| | |
|---|---|
| *kStatus_Success* | Get the transferred count Successfully. |
| *kStatus_NoTransferInProgress* | No transfer in process. |

## 23.6.10 FlexIO SMARTDMA MCU Interface LCD Driver

### 23.6.10.1 Overview

#### Data Structures

- struct flexio_mculcd_smartdma_handle_t
  *FlexIO MCULCD SMARTDMA transfer handle, users should not touch the content of the handle. More...*
- struct flexio_mculcd_smartdma_config_t
  *FlexIO MCULCD SMARTDMA configuration. More...*

#### Macros

- #define FSL_FLEXIO_MCULCD_SMARTDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *FlexIO MCULCD SMARTDMA driver version 2.0.0.*
- #define FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN 64U
  *SMARTDMA transfer size should be multiple of 64 bytes.*
- #define FLEXIO_MCULCD_SMARTDMA_TX_ADDR_ALIGN 4U
  *SMARTDMA transfer memory address should be 4 byte aligned.*

#### Typedefs

- typedef void(* flexio_mculcd_smartdma_transfer_callback_t )(FLEXIO_MCULCD_Type *base, flexio_mculcd_smartdma_handle_t *handle, status_t status, void *userData)
  *FlexIO MCULCD master callback for transfer complete.*

#### SMARTDMA Transactional

- status_t FLEXIO_MCULCD_TransferCreateHandleSMARTDMA (FLEXIO_MCULCD_Type *base, flexio_mculcd_smartdma_handle_t *handle, const flexio_mculcd_smartdma_config_t *config, flexio_mculcd_smartdma_transfer_callback_t callback, void *userData)
  *Initializes the FLEXO MCULCD master SMARTDMA handle.*
- status_t FLEXIO_MCULCD_TransferSMARTDMA (FLEXIO_MCULCD_Type *base, flexio_mculcd_smartdma_handle_t *handle, flexio_mculcd_transfer_t *xfer)
  *Performs a non-blocking FlexIO MCULCD transfer using SMARTDMA.*
- void FLEXIO_MCULCD_TransferAbortSMARTDMA (FLEXIO_MCULCD_Type *base, flexio_mculcd_smartdma_handle_t *handle)
  *Aborts a FlexIO MCULCD transfer using SMARTDMA.*
- status_t FLEXIO_MCULCD_TransferGetCountSMARTDMA (FLEXIO_MCULCD_Type *base, flexio_mculcd_smartdma_handle_t *handle, size_t *count)
  *Gets the remaining bytes for FlexIO MCULCD SMARTDMA transfer.*

### 23.6.10.2 Data Structure Documentation

#### 23.6.10.2.1 struct _flexio_mculcd_smartdma_handle

typedef for flexio_mculcd_smartdma_handle_t in advance.

**Data Fields**

- FLEXIO_MCULCD_Type * base
  *Pointer to the FLEXIO_MCULCD_Type.*
- size_t dataCount
  *Total count to be transferred.*
- uint32_t dataAddrOrSameValue
  *When sending the same value for many times, this is the value to send.*
- size_t dataCountUsingEzh
  *Data transfered using SMARTDMA.*
- volatile size_t remainingCount
  *Remaining count to transfer.*
- volatile uint32_t state
  *FlexIO MCULCD driver internal state.*
- uint8_t smartdmaApi
  *The SMARTDMA API used during transfer.*
- bool needColorConvert
  *Need color convert or not.*
- uint8_t blockingXferBuffer [FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN *3/2]
  *Used for blocking method color space convet.*
- flexio_mculcd_smartdma_transfer_callback_t completionCallback
  *Callback for MCULCD SMARTDMA transfer.*
- void * userData
  *User Data for MCULCD SMARTDMA callback.*

##### 23.6.10.2.1.1 Field Documentation

###### 23.6.10.2.1.1.1 FLEXIO_MCULCD_Type* flexio_mculcd_smartdma_handle_t::base

###### 23.6.10.2.1.1.2 size_t flexio_mculcd_smartdma_handle_t::dataCount

###### 23.6.10.2.1.1.3 uint32_t flexio_mculcd_smartdma_handle_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

#### 23.6.10.2.1.1.4 size_t flexio_mculcd_smartdma_handle_t::dataCountUsingEzh

#### 23.6.10.2.1.1.5 volatile size_t flexio_mculcd_smartdma_handle_t::remainingCount

#### 23.6.10.2.1.1.6 volatile uint32_t flexio_mculcd_smartdma_handle_t::state

#### 23.6.10.2.1.1.7 uint8_t flexio_mculcd_smartdma_handle_t::smartdmaApi

#### 23.6.10.2.1.1.8 bool flexio_mculcd_smartdma_handle_t::needColorConvert

#### 23.6.10.2.1.1.9 uint8_t flexio_mculcd_smartdma_handle_t::blockingXferBuffer[FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN *3/2]

### 23.6.10.2.2 struct flexio_mculcd_smartdma_config_t

**Data Fields**

- flexio_mculcd_pixel_format_t inputPixelFormat
  *The pixel format in the frame buffer.*
- flexio_mculcd_pixel_format_t outputPixelFormat
  *The pixel format on the 8080/68k bus.*

#### 23.6.10.2.2.1 Field Documentation

#### 23.6.10.2.2.1.1 flexio_mculcd_pixel_format_t flexio_mculcd_smartdma_config_t::inputPixelFormat

#### 23.6.10.2.2.1.2 flexio_mculcd_pixel_format_t flexio_mculcd_smartdma_config_t::outputPixelFormat

## 23.6.10.3 Macro Definition Documentation

#### 23.6.10.3.1 #define FSL_FLEXIO_MCULCD_SMARTDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

#### 23.6.10.3.2 #define FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN 64U

#### 23.6.10.3.3 #define FLEXIO_MCULCD_SMARTDMA_TX_ADDR_ALIGN 4U

## 23.6.10.4 Typedef Documentation

#### 23.6.10.4.1 typedef void(* flexio_mculcd_smartdma_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculcd_smartdma_handle_t *handle, status_t status, void *userData)

When transfer finished, the callback function is called and returns the `status` as kStatus_FLEXIO_MCULCD_Idle.

### 23.6.10.5 Function Documentation

#### 23.6.10.5.1 status_t FLEXIO_MCULCD_TransferCreateHandleSMARTDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle*, const flexio_mculcd_smartdma_config_t * *config*, flexio_mculcd_smartdma_transfer_callback_t *callback*, void * *userData* )

This function initializes the FLEXO MCULCD master SMARTDMA handle which can be used for other FLEXO MCULCD transactional APIs. For a specified FLEXO MCULCD instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | Pointer to flexio_mculcd_smartdma_handle_t structure to store the transfer state. |
| *config* | Pointer to the configuration. |
| *callback* | MCULCD transfer complete callback, NULL means no callback. |
| *userData* | callback function parameter. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |

#### 23.6.10.5.2 status_t FLEXIO_MCULCD_TransferSMARTDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer* )

This function returns immediately after transfer initiates. Use the callback function to check whether the transfer is completed.

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | pointer to flexio_mculcd_smartdma_handle_t structure to store the transfer state. |
| *xfer* | Pointer to FlexIO MCULCD transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_MCULCD_Busy* | FlexIO MCULCD is not idle, it is running another transfer. |

#### 23.6.10.5.3 void FLEXIO_MCULCD_TransferAbortSMARTDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | FlexIO MCULCD SMARTDMA handle pointer. |

#### 23.6.10.5.4 status_t FLEXIO_MCULCD_TransferGetCountSMARTDMA ( FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_MCULCD_Type structure. |
| *handle* | FlexIO MCULCD SMARTDMA handle pointer. |
| *count* | Number of count transferred so far by the SMARTDMA transaction. |

Return values

| | |
|---|---|
| *kStatus_Success* | Get the transferred count Successfully. |
| *kStatus_NoTransferInProgress* | No transfer in process. |

## 23.7 FlexIO SPI Driver

### 23.7.1 Overview

The MCUXpresso SDK provides a peripheral driver for an SPI function using the Flexible I/O module of MCUXpresso SDK devices.

FlexIO SPI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for FlexIO SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the FLEXIO_SPI_Type ∗base as the first parameter. FlexIO SPI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the flexio_spi_master_handle_t/flexio_spi_slave_handle_t as the second parameter. Initialize the handle by calling the FLEXIO_SPI_MasterTransferCreateHandle() or FLEXIO_SPI_SlaveTransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions FLEXIO_SPI_MasterTransferNonBlocking()/FLEXIO_SPI_SlaveTransferNonBlocking() set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the kStatus_FLEXIO_SPI_Idle status.

Note that the FlexIO SPI slave driver only supports discontinuous PCS access, which is a limitation. The FlexIO SPI slave driver can support continuous PCS, but the slave cannot adapt discontinuous and continuous PCS automatically. Users can change the timer disable mode in FLEXIO_SPI_SlaveInit manually, from kFLEXIO_TimerDisableOnTimerCompare to kFLEXIO_TimerDisableNever to enable a discontinuous PCS access. Only CPHA = 0 is supported.

### 23.7.2 Typical use case

#### 23.7.2.1 FlexIO SPI send/receive using an interrupt method

```
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];

void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle
      , status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_SPI_Idle == status)
    {
        txFinished = true;
    }
```

```
}

void main(void)
{
    //...
    flexio_spi_transfer_t xfer = {0};
    flexio_spi_master_config_t userConfig;

    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);
    userConfig.baudRate_Bps = 500000U;

    spiDev.flexioBase = BOARD_FLEXIO_BASE;
    spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
    spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
    spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
    spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
    spiDev.shifterIndex[0] = 0U;
    spiDev.shifterIndex[1] = 1U;
    spiDev.timerIndex[0] = 0U;
    spiDev.timerIndex[1] = 1U;

    FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

    xfer.txData = srcBuff;
    xfer.rxData = destBuff;
    xfer.dataSize = BUFFER_SIZE;
    xfer.flags = kFLEXIO_SPI_8bitMsb;
    FLEXIO_SPI_MasterTransferCreateHandle(&spiDev, &g_spiHandle,
      FLEXIO_SPI_MasterUserCallback, NULL);
    FLEXIO_SPI_MasterTransferNonBlocking(&spiDev, &g_spiHandle, &xfer);

    // Send finished.
    while (!txFinished)
    {
    }

        // ...
}
```

### 23.7.2.2 FlexIO_SPI Send/Receive in DMA way

```
    dma_handle_t g_spiTxDmaHandle;
    dma_handle_t g_spiRxDmaHandle;
    flexio_spi_master_handle_t g_spiHandle;
    FLEXIO_SPI_Type spiDev;
    volatile bool txFinished;
    static uint8_t srcBuff[BUFFER_SIZE];
    static uint8_t destBuff[BUFFER_SIZE];
    void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_dma_handle_t
       *handle, status_t status, void *userData)
    {
        userData = userData;

        if (kStatus_FLEXIO_SPI_Idle == status)
        {
            txFinished = true;
        }
    }

    void main(void)
    {
        flexio_spi_transfer_t xfer = {0};
        flexio_spi_master_config_t userConfig;

        FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);
```

```
        userConfig.baudRate_Bps = 500000U;

        spiDev.flexioBase = BOARD_FLEXIO_BASE;
        spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
        spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
        spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
        spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
        spiDev.shifterIndex[0] = 0U;
        spiDev.shifterIndex[1] = 1U;
        spiDev.timerIndex[0] = 0U;
        spiDev.timerIndex[1] = 1U;

        /* Init DMAMUX. */
        DMAMUX_Init(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR)

        /* Init the DMA/EDMA module */
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
        DMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR);
        DMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL);
        DMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
        edma_config_t edmaConfig;

        EDMA_GetDefaultConfig(&edmaConfig);
        EDMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, &edmaConfig);
        EDMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
      FLEXIO_SPI_TX_DMA_CHANNEL);
        EDMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
      FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

        dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
      shifterIndex[0]);
        dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
      shifterIndex[1]);

        /* Requests  DMA channels for transmit and receive. */
        DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL, (
      dma_request_source_t)dma_request_source_tx);
        DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL, (
      dma_request_source_t)dma_request_source_rx);
        DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
      FLEXIO_SPI_TX_DMA_CHANNEL);
        DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
      FLEXIO_SPI_RX_DMA_CHANNEL);

        FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

        /* Initializes the buffer. */
        for (i = 0; i < BUFFER_SIZE; i++)
        {
            srcBuff[i] = i;
        }

        /* Sends to the slave. */
        xfer.txData = srcBuff;
        xfer.rxData = destBuff;
        xfer.dataSize = BUFFER_SIZE;
        xfer.flags = kFLEXIO_SPI_8bitMsb;
        FLEXIO_SPI_MasterTransferCreateHandleDMA(&spiDev, &
      g_spiHandle, FLEXIO_SPI_MasterUserCallback, NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);
        FLEXIO_SPI_MasterTransferDMA(&spiDev, &g_spiHandle, &xfer);

        // Send finished.
        while (!txFinished)
        {
```

```
    }

    // ...
}
```

## Modules

- FlexIO DMA SPI Driver
- FlexIO eDMA SPI Driver

## Data Structures

- struct FLEXIO_SPI_Type
  *Define FlexIO SPI access structure typedef. More...*
- struct flexio_spi_master_config_t
  *Define FlexIO SPI master configuration structure. More...*
- struct flexio_spi_slave_config_t
  *Define FlexIO SPI slave configuration structure. More...*
- struct flexio_spi_transfer_t
  *Define FlexIO SPI transfer structure. More...*
- struct flexio_spi_master_handle_t
  *Define FlexIO SPI handle structure. More...*

## Macros

- #define FLEXIO_SPI_DUMMYDATA (0xFFFFU)
  *FlexIO SPI dummy transfer data, the data is sent while txData is NULL.*

## Typedefs

- typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t
  *Slave handle is the same with master handle.*
- typedef void(∗ flexio_spi_master_transfer_callback_t )(FLEXIO_SPI_Type ∗base, flexio_spi_master_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO SPI master callback for finished transmit.*
- typedef void(∗ flexio_spi_slave_transfer_callback_t )(FLEXIO_SPI_Type ∗base, flexio_spi_slave_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO SPI slave callback for finished transmit.*

## Enumerations

- enum _flexio_spi_status {
  kStatus_FLEXIO_SPI_Busy = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 1),
  kStatus_FLEXIO_SPI_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 2),
  kStatus_FLEXIO_SPI_Error = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 3) }

*Error codes for the FlexIO SPI driver.*

- enum flexio_spi_clock_phase_t {
  kFLEXIO_SPI_ClockPhaseFirstEdge = 0x0U,
  kFLEXIO_SPI_ClockPhaseSecondEdge = 0x1U }
  *FlexIO SPI clock phase configuration.*
- enum flexio_spi_shift_direction_t {
  kFLEXIO_SPI_MsbFirst = 0,
  kFLEXIO_SPI_LsbFirst = 1 }
  *FlexIO SPI data shifter direction options.*
- enum flexio_spi_data_bitcount_mode_t {
  kFLEXIO_SPI_8BitMode = 0x08U,
  kFLEXIO_SPI_16BitMode = 0x10U }
  *FlexIO SPI data length mode options.*
- enum _flexio_spi_interrupt_enable {
  kFLEXIO_SPI_TxEmptyInterruptEnable = 0x1U,
  kFLEXIO_SPI_RxFullInterruptEnable = 0x2U }
  *Define FlexIO SPI interrupt mask.*
- enum _flexio_spi_status_flags {
  kFLEXIO_SPI_TxBufferEmptyFlag = 0x1U,
  kFLEXIO_SPI_RxBufferFullFlag = 0x2U }
  *Define FlexIO SPI status mask.*
- enum _flexio_spi_dma_enable {
  kFLEXIO_SPI_TxDmaEnable = 0x1U,
  kFLEXIO_SPI_RxDmaEnable = 0x2U,
  kFLEXIO_SPI_DmaAllEnable = 0x3U }
  *Define FlexIO SPI DMA mask.*
- enum _flexio_spi_transfer_flags {
  kFLEXIO_SPI_8bitMsb = 0x1U,
  kFLEXIO_SPI_8bitLsb = 0x2U,
  kFLEXIO_SPI_16bitMsb = 0x9U,
  kFLEXIO_SPI_16bitLsb = 0xaU }
  *Define FlexIO SPI transfer flags.*

## Driver version

- #define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  *FlexIO SPI driver version 2.1.3.*

## FlexIO SPI Configuration

- void FLEXIO_SPI_MasterInit (FLEXIO_SPI_Type *base, flexio_spi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  *Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration.*
- void FLEXIO_SPI_MasterDeinit (FLEXIO_SPI_Type *base)
  *Resets the FlexIO SPI timer and shifter config.*

- void FLEXIO_SPI_MasterGetDefaultConfig (flexio_spi_master_config_t *masterConfig)
  *Gets the default configuration to configure the FlexIO SPI master.*
- void FLEXIO_SPI_SlaveInit (FLEXIO_SPI_Type *base, flexio_spi_slave_config_t *slaveConfig)
  *Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration.*
- void FLEXIO_SPI_SlaveDeinit (FLEXIO_SPI_Type *base)
  *Gates the FlexIO clock.*
- void FLEXIO_SPI_SlaveGetDefaultConfig (flexio_spi_slave_config_t *slaveConfig)
  *Gets the default configuration to configure the FlexIO SPI slave.*

## Status

- uint32_t FLEXIO_SPI_GetStatusFlags (FLEXIO_SPI_Type *base)
  *Gets FlexIO SPI status flags.*
- void FLEXIO_SPI_ClearStatusFlags (FLEXIO_SPI_Type *base, uint32_t mask)
  *Clears FlexIO SPI status flags.*

## Interrupts

- void FLEXIO_SPI_EnableInterrupts (FLEXIO_SPI_Type *base, uint32_t mask)
  *Enables the FlexIO SPI interrupt.*
- void FLEXIO_SPI_DisableInterrupts (FLEXIO_SPI_Type *base, uint32_t mask)
  *Disables the FlexIO SPI interrupt.*

## DMA Control

- void FLEXIO_SPI_EnableDMA (FLEXIO_SPI_Type *base, uint32_t mask, bool enable)
  *Enables/disables the FlexIO SPI transmit DMA.*
- static uint32_t FLEXIO_SPI_GetTxDataRegisterAddress (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction)
  *Gets the FlexIO SPI transmit data register address for MSB first transfer.*
- static uint32_t FLEXIO_SPI_GetRxDataRegisterAddress (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction)
  *Gets the FlexIO SPI receive data register address for the MSB first transfer.*

## Bus Operations

- static void FLEXIO_SPI_Enable (FLEXIO_SPI_Type *base, bool enable)
  *Enables/disables the FlexIO SPI module operation.*
- void FLEXIO_SPI_MasterSetBaudRate (FLEXIO_SPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClockHz)
  *Sets baud rate for the FlexIO SPI transfer, which is only used for the master.*
- static void FLEXIO_SPI_WriteData (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction, uint16_t data)
  *Writes one byte of data, which is sent using the MSB method.*

- static uint16_t FLEXIO_SPI_ReadData (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction)
  *Reads 8 bit/16 bit data.*
- void FLEXIO_SPI_WriteBlocking (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction, const uint8_t *buffer, size_t size)
  *Sends a buffer of data bytes.*
- void FLEXIO_SPI_ReadBlocking (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction, uint8_t *buffer, size_t size)
  *Receives a buffer of bytes.*
- void FLEXIO_SPI_MasterTransferBlocking (FLEXIO_SPI_Type *base, flexio_spi_transfer_t *xfer)
  *Receives a buffer of bytes.*

## Transactional

- status_t FLEXIO_SPI_MasterTransferCreateHandle (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, flexio_spi_master_transfer_callback_t callback, void *userData)
  *Initializes the FlexIO SPI Master handle, which is used in transactional functions.*
- status_t FLEXIO_SPI_MasterTransferNonBlocking (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, flexio_spi_transfer_t *xfer)
  *Master transfer data using IRQ.*
- void FLEXIO_SPI_MasterTransferAbort (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle)
  *Aborts the master data transfer, which used IRQ.*
- status_t FLEXIO_SPI_MasterTransferGetCount (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, size_t *count)
  *Gets the data transfer status which used IRQ.*
- void FLEXIO_SPI_MasterTransferHandleIRQ (void *spiType, void *spiHandle)
  *FlexIO SPI master IRQ handler function.*
- status_t FLEXIO_SPI_SlaveTransferCreateHandle (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, flexio_spi_slave_transfer_callback_t callback, void *userData)
  *Initializes the FlexIO SPI Slave handle, which is used in transactional functions.*
- status_t FLEXIO_SPI_SlaveTransferNonBlocking (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, flexio_spi_transfer_t *xfer)
  *Slave transfer data using IRQ.*
- static void FLEXIO_SPI_SlaveTransferAbort (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle)
  *Aborts the slave data transfer which used IRQ, share same API with master.*
- static status_t FLEXIO_SPI_SlaveTransferGetCount (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, size_t *count)
  *Gets the data transfer status which used IRQ, share same API with master.*
- void FLEXIO_SPI_SlaveTransferHandleIRQ (void *spiType, void *spiHandle)
  *FlexIO SPI slave IRQ handler function.*

## 23.7.3 Data Structure Documentation

### 23.7.3.1 struct FLEXIO_SPI_Type

#### Data Fields

- FLEXIO_Type * flexioBase
  *FlexIO base pointer.*
- uint8_t SDOPinIndex
  *Pin select for data output.*
- uint8_t SDIPinIndex
  *Pin select for data input.*
- uint8_t SCKPinIndex
  *Pin select for clock.*
- uint8_t CSnPinIndex
  *Pin select for enable.*
- uint8_t shifterIndex [2]
  *Shifter index used in FlexIO SPI.*
- uint8_t timerIndex [2]
  *Timer index used in FlexIO SPI.*

#### 23.7.3.1.0.1 Field Documentation

##### 23.7.3.1.0.1.1 FLEXIO_Type* FLEXIO_SPI_Type::flexioBase

##### 23.7.3.1.0.1.2 uint8_t FLEXIO_SPI_Type::SDOPinIndex

##### 23.7.3.1.0.1.3 uint8_t FLEXIO_SPI_Type::SDIPinIndex

##### 23.7.3.1.0.1.4 uint8_t FLEXIO_SPI_Type::SCKPinIndex

##### 23.7.3.1.0.1.5 uint8_t FLEXIO_SPI_Type::CSnPinIndex

##### 23.7.3.1.0.1.6 uint8_t FLEXIO_SPI_Type::shifterIndex[2]

##### 23.7.3.1.0.1.7 uint8_t FLEXIO_SPI_Type::timerIndex[2]

### 23.7.3.2 struct flexio_spi_master_config_t

#### Data Fields

- bool enableMaster
  *Enable/disable FlexIO SPI master after configuration.*
- bool enableInDoze
  *Enable/disable FlexIO operation in doze mode.*
- bool enableInDebug
  *Enable/disable FlexIO operation in debug mode.*
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  *fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

- uint32_t baudRate_Bps
  *Baud rate in Bps.*
- flexio_spi_clock_phase_t phase
  *Clock phase.*
- flexio_spi_data_bitcount_mode_t dataMode
  *8bit or 16bit mode.*

#### 23.7.3.2.0.2 Field Documentation

#### 23.7.3.2.0.2.1 bool flexio_spi_master_config_t::enableMaster

#### 23.7.3.2.0.2.2 bool flexio_spi_master_config_t::enableInDoze

#### 23.7.3.2.0.2.3 bool flexio_spi_master_config_t::enableInDebug

#### 23.7.3.2.0.2.4 bool flexio_spi_master_config_t::enableFastAccess

#### 23.7.3.2.0.2.5 uint32_t flexio_spi_master_config_t::baudRate_Bps

#### 23.7.3.2.0.2.6 flexio_spi_clock_phase_t flexio_spi_master_config_t::phase

#### 23.7.3.2.0.2.7 flexio_spi_data_bitcount_mode_t flexio_spi_master_config_t::dataMode

### 23.7.3.3 struct flexio_spi_slave_config_t

### Data Fields

- bool enableSlave
  *Enable/disable FlexIO SPI slave after configuration.*
- bool enableInDoze
  *Enable/disable FlexIO operation in doze mode.*
- bool enableInDebug
  *Enable/disable FlexIO operation in debug mode.*
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  *fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- flexio_spi_clock_phase_t phase
  *Clock phase.*
- flexio_spi_data_bitcount_mode_t dataMode
  *8bit or 16bit mode.*

#### 23.7.3.3.0.3 Field Documentation

##### 23.7.3.3.0.3.1 bool flexio_spi_slave_config_t::enableSlave

##### 23.7.3.3.0.3.2 bool flexio_spi_slave_config_t::enableInDoze

##### 23.7.3.3.0.3.3 bool flexio_spi_slave_config_t::enableInDebug

##### 23.7.3.3.0.3.4 bool flexio_spi_slave_config_t::enableFastAccess

##### 23.7.3.3.0.3.5 flexio_spi_clock_phase_t flexio_spi_slave_config_t::phase

##### 23.7.3.3.0.3.6 flexio_spi_data_bitcount_mode_t flexio_spi_slave_config_t::dataMode

### 23.7.3.4 struct flexio_spi_transfer_t

#### Data Fields

- uint8_t * txData
  *Send buffer.*
- uint8_t * rxData
  *Receive buffer.*
- size_t dataSize
  *Transfer bytes.*
- uint8_t flags
  *FlexIO SPI control flag, MSB first or LSB first.*

#### 23.7.3.4.0.4 Field Documentation

##### 23.7.3.4.0.4.1 uint8_t* flexio_spi_transfer_t::txData

##### 23.7.3.4.0.4.2 uint8_t* flexio_spi_transfer_t::rxData

##### 23.7.3.4.0.4.3 size_t flexio_spi_transfer_t::dataSize

##### 23.7.3.4.0.4.4 uint8_t flexio_spi_transfer_t::flags

### 23.7.3.5 struct _flexio_spi_master_handle

typedef for flexio_spi_master_handle_t in advance.

#### Data Fields

- uint8_t * txData
  *Transfer buffer.*
- uint8_t * rxData
  *Receive buffer.*
- size_t transferSize
  *Total bytes to be transferred.*
- volatile size_t txRemainingBytes

*Send data remaining in bytes.*

- volatile size_t rxRemainingBytes
  *Receive data remaining in bytes.*
- volatile uint32_t state
  *FlexIO SPI internal state.*
- uint8_t bytePerFrame
  *SPI mode, 2bytes or 1byte in a frame.*
- flexio_spi_shift_direction_t direction
  *Shift direction.*
- flexio_spi_master_transfer_callback_t callback
  *FlexIO SPI callback.*
- void * userData
  *Callback parameter.*

#### 23.7.3.5.0.5 Field Documentation

##### 23.7.3.5.0.5.1 uint8_t* flexio_spi_master_handle_t::txData

##### 23.7.3.5.0.5.2 uint8_t* flexio_spi_master_handle_t::rxData

##### 23.7.3.5.0.5.3 size_t flexio_spi_master_handle_t::transferSize

##### 23.7.3.5.0.5.4 volatile size_t flexio_spi_master_handle_t::txRemainingBytes

##### 23.7.3.5.0.5.5 volatile size_t flexio_spi_master_handle_t::rxRemainingBytes

##### 23.7.3.5.0.5.6 volatile uint32_t flexio_spi_master_handle_t::state

##### 23.7.3.5.0.5.7 flexio_spi_shift_direction_t flexio_spi_master_handle_t::direction

##### 23.7.3.5.0.5.8 flexio_spi_master_transfer_callback_t flexio_spi_master_handle_t::callback

##### 23.7.3.5.0.5.9 void* flexio_spi_master_handle_t::userData

### 23.7.4 Macro Definition Documentation

#### 23.7.4.1 #define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

#### 23.7.4.2 #define FLEXIO_SPI_DUMMYDATA (0xFFFFU)

### 23.7.5 Typedef Documentation

#### 23.7.5.1 typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t

### 23.7.6 Enumeration Type Documentation

#### 23.7.6.1 enum _flexio_spi_status

Enumerator

***kStatus_FLEXIO_SPI_Busy*** FlexIO SPI is busy.
***kStatus_FLEXIO_SPI_Idle*** SPI is idle.
***kStatus_FLEXIO_SPI_Error*** FlexIO SPI error.

#### 23.7.6.2 enum flexio_spi_clock_phase_t

Enumerator

***kFLEXIO_SPI_ClockPhaseFirstEdge*** First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

*kFLEXIO_SPI_ClockPhaseSecondEdge* First edge on SPSCK occurs at the start of the first cycle of a data transfer.

#### 23.7.6.3 enum flexio_spi_shift_direction_t

Enumerator

*kFLEXIO_SPI_MsbFirst* Data transfers start with most significant bit.
*kFLEXIO_SPI_LsbFirst* Data transfers start with least significant bit.

#### 23.7.6.4 enum flexio_spi_data_bitcount_mode_t

Enumerator

*kFLEXIO_SPI_8BitMode* 8-bit data transmission mode.
*kFLEXIO_SPI_16BitMode* 16-bit data transmission mode.

#### 23.7.6.5 enum _flexio_spi_interrupt_enable

Enumerator

*kFLEXIO_SPI_TxEmptyInterruptEnable* Transmit buffer empty interrupt enable.
*kFLEXIO_SPI_RxFullInterruptEnable* Receive buffer full interrupt enable.

#### 23.7.6.6 enum _flexio_spi_status_flags

Enumerator

*kFLEXIO_SPI_TxBufferEmptyFlag* Transmit buffer empty flag.
*kFLEXIO_SPI_RxBufferFullFlag* Receive buffer full flag.

#### 23.7.6.7 enum _flexio_spi_dma_enable

Enumerator

*kFLEXIO_SPI_TxDmaEnable* Tx DMA request source.
*kFLEXIO_SPI_RxDmaEnable* Rx DMA request source.
*kFLEXIO_SPI_DmaAllEnable* All DMA request source.

#### 23.7.6.8 enum _flexio_spi_transfer_flags

Enumerator

***kFLEXIO_SPI_8bitMsb*** FlexIO SPI 8-bit MSB first.
***kFLEXIO_SPI_8bitLsb*** FlexIO SPI 8-bit LSB first.
***kFLEXIO_SPI_16bitMsb*** FlexIO SPI 16-bit MSB first.
***kFLEXIO_SPI_16bitLsb*** FlexIO SPI 16-bit LSB first.

### 23.7.7 Function Documentation

#### 23.7.7.1 void FLEXIO_SPI_MasterInit ( FLEXIO_SPI_Type * *base*, flexio_spi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz* )

The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Note

FlexIO SPI master only support CPOL = 0, which means clock inactive low.

Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *masterConfig* | Pointer to the flexio_spi_master_config_t structure. |
| *srcClock_Hz* | FlexIO source clock in Hz. |

#### 23.7.7.2 void FLEXIO_SPI_MasterDeinit ( FLEXIO_SPI_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type. |

#### 23.7.7.3 void FLEXIO_SPI_MasterGetDefaultConfig ( flexio_spi_master_config_t * *masterConfig* )

The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

| | |
|---|---|
| *masterConfig* | Pointer to the flexio_spi_master_config_t structure. |

#### 23.7.7.4 void FLEXIO_SPI_SlaveInit ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_config_t * *slaveConfig* )

The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_-SlaveGetDefaultConfig().

Note

Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
```

```
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *slaveConfig* | Pointer to the flexio_spi_slave_config_t structure. |

#### 23.7.7.5 void FLEXIO_SPI_SlaveDeinit ( FLEXIO_SPI_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type. |

#### 23.7.7.6 void FLEXIO_SPI_SlaveGetDefaultConfig ( flexio_spi_slave_config_t * *slaveConfig* )

The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

| | |
|---|---|
| *slaveConfig* | Pointer to the flexio_spi_slave_config_t structure. |

#### 23.7.7.7 uint32_t FLEXIO_SPI_GetStatusFlags ( FLEXIO_SPI_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |

Returns

status flag; Use the status flag to AND the following flag mask and get the status.
- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

#### 23.7.7.8 void FLEXIO_SPI_ClearStatusFlags ( FLEXIO_SPI_Type * *base,* uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *mask* | status flag The parameter can be any combination of the following values:<br>• kFLEXIO_SPI_TxEmptyFlag<br>• kFLEXIO_SPI_RxEmptyFlag |

#### 23.7.7.9 void FLEXIO_SPI_EnableInterrupts ( FLEXIO_SPI_Type * *base*, uint32_t *mask* )

This function enables the FlexIO SPI interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *mask* | interrupt source. The parameter can be any combination of the following values:<br>• kFLEXIO_SPI_RxFullInterruptEnable<br>• kFLEXIO_SPI_TxEmptyInterruptEnable |

#### 23.7.7.10 void FLEXIO_SPI_DisableInterrupts ( FLEXIO_SPI_Type * *base*, uint32_t *mask* )

This function disables the FlexIO SPI interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *mask* | interrupt source The parameter can be any combination of the following values:<br>• kFLEXIO_SPI_RxFullInterruptEnable<br>• kFLEXIO_SPI_TxEmptyInterruptEnable |

#### 23.7.7.11 void FLEXIO_SPI_EnableDMA ( FLEXIO_SPI_Type * *base*, uint32_t *mask*, bool *enable* )

This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_-TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *mask* | SPI DMA source. |
| *enable* | True means enable DMA, false means disable DMA. |

#### 23.7.7.12 static uint32_t FLEXIO_SPI_GetTxDataRegisterAddress ( FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction* ) [inline], [static]

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *direction* | Shift direction of MSB first or LSB first. |

Returns

FlexIO SPI transmit data register address.

#### 23.7.7.13 static uint32_t FLEXIO_SPI_GetRxDataRegisterAddress ( FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction* ) [inline], [static]

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *direction* | Shift direction of MSB first or LSB first. |

Returns

FlexIO SPI receive data register address.

#### 23.7.7.14 static void FLEXIO_SPI_Enable ( FLEXIO_SPI_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type. |
| *enable* | True to enable, false does not have any effect. |

#### 23.7.7.15 void FLEXIO_SPI_MasterSetBaudRate ( FLEXIO_SPI_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClockHz* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *baudRate_Bps* | Baud Rate needed in Hz. |
| *srcClockHz* | SPI source clock frequency in Hz. |

#### 23.7.7.16 static void FLEXIO_SPI_WriteData ( FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*, uint16_t *data* ) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *direction* | Shift direction of MSB first or LSB first. |
| *data* | 8 bit/16 bit data. |

#### 23.7.7.17 static uint16_t FLEXIO_SPI_ReadData ( FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction* ) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *direction* | Shift direction of MSB first or LSB first. |

Returns

8 bit/16 bit data received.

### 23.7.7.18 void FLEXIO_SPI_WriteBlocking ( FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*, const uint8_t * *buffer*, size_t *size* )

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *direction* | Shift direction of MSB first or LSB first. |
| *buffer* | The data bytes to send. |
| *size* | The number of data bytes to send. |

### 23.7.7.19 void FLEXIO_SPI_ReadBlocking ( FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*, uint8_t * *buffer*, size_t *size* )

Note

This function blocks using the polling method until all bytes have been received.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *direction* | Shift direction of MSB first or LSB first. |
| *buffer* | The buffer to store the received bytes. |

| | |
|---|---|
| *size* | The number of data bytes to be received. |
| *direction* | Shift direction of MSB first or LSB first. |

#### 23.7.7.20 void FLEXIO_SPI_MasterTransferBlocking ( FLEXIO_SPI_Type * *base*, flexio_spi_transfer_t * *xfer* )

Note

This function blocks via polling until all bytes have been received.

Parameters

| | |
|---|---|
| *base* | pointer to FLEXIO_SPI_Type structure |
| *xfer* | FlexIO SPI transfer structure, see flexio_spi_transfer_t. |

#### 23.7.7.21 status_t FLEXIO_SPI_MasterTransferCreateHandle ( FLEXIO_SPI_Type * *base*, flexio_spi_master_handle_t * *handle*, flexio_spi_master_transfer_callback_t *callback*, void * *userData* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *handle* | Pointer to the flexio_spi_master_handle_t structure to store the transfer state. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO type/handle/ISR table out of range. |

#### 23.7.7.22 status_t FLEXIO_SPI_MasterTransferNonBlocking ( FLEXIO_SPI_Type * *base*, flexio_spi_master_handle_t * *handle*, flexio_spi_transfer_t * *xfer* )

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *handle* | Pointer to the flexio_spi_master_handle_t structure to store the transfer state. |
| *xfer* | FlexIO SPI transfer structure. See flexio_spi_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_SPI_Busy* | SPI is not idle, is running another transfer. |

#### 23.7.7.23 void FLEXIO_SPI_MasterTransferAbort ( FLEXIO_SPI_Type * *base*, flexio_spi_master_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *handle* | Pointer to the flexio_spi_master_handle_t structure to store the transfer state. |

#### 23.7.7.24 status_t FLEXIO_SPI_MasterTransferGetCount ( FLEXIO_SPI_Type * *base*, flexio_spi_master_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *handle* | Pointer to the flexio_spi_master_handle_t structure to store the transfer state. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_InvalidArgument* | count is Invalid. |
| *kStatus_Success* | Successfully return the count. |

#### 23.7.7.25 void FLEXIO_SPI_MasterTransferHandleIRQ ( void * *spiType*, void * *spiHandle* )

Parameters

| | |
|---|---|
| *spiType* | Pointer to the FLEXIO_SPI_Type structure. |
| *spiHandle* | Pointer to the flexio_spi_master_handle_t structure to store the transfer state. |

### 23.7.7.26 status_t FLEXIO_SPI_SlaveTransferCreateHandle ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*, flexio_spi_slave_transfer_callback_t *callback*, void * *userData* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *handle* | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO type/handle/ISR table out of range. |

### 23.7.7.27 status_t FLEXIO_SPI_SlaveTransferNonBlocking ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*, flexio_spi_transfer_t * *xfer* )

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

| | |
|---|---|
| *handle* | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *xfer* | FlexIO SPI transfer structure. See flexio_spi_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_SPI_Busy* | SPI is not idle; it is running another transfer. |

#### 23.7.7.28 static void FLEXIO_SPI_SlaveTransferAbort ( FLEXIO_SPI_Type * *base,* flexio_spi_slave_handle_t * *handle* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *handle* | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |

#### 23.7.7.29 static status_t FLEXIO_SPI_SlaveTransferGetCount ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*, size_t * *count* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_SPI_Type structure. |
| *handle* | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_InvalidArgument* | count is Invalid. |
| *kStatus_Success* | Successfully return the count. |

#### 23.7.7.30 void FLEXIO_SPI_SlaveTransferHandleIRQ ( void * *spiType*, void * *spiHandle* )

Parameters

| | |
|---|---|
| *spiType* | Pointer to the FLEXIO_SPI_Type structure. |
| *spiHandle* | Pointer to the flexio_spi_slave_handle_t structure to store the transfer state. |

## 23.7.8 FlexIO eDMA SPI Driver

### 23.7.8.1 Overview

#### Data Structures

- struct flexio_spi_master_edma_handle_t
  *FlexIO SPI eDMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef
  flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t
  *Slave handle is the same with master handle.*
- typedef void(∗ flexio_spi_master_edma_transfer_callback_t )(FLEXIO_SPI_Type ∗base, flexio_spi_master_edma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO SPI master callback for finished transmit.*
- typedef void(∗ flexio_spi_slave_edma_transfer_callback_t )(FLEXIO_SPI_Type ∗base, flexio_spi_slave_edma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO SPI slave callback for finished transmit.*

#### Driver version

- #define FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  *FlexIO SPI EDMA driver version 2.1.3.*

#### eDMA Transactional

- status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_edma_handle_t ∗handle, flexio_spi_master_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗txHandle, edma_handle_t ∗rxHandle)
  *Initializes the FlexIO SPI master eDMA handle.*
- status_t FLEXIO_SPI_MasterTransferEDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_edma_handle_t ∗handle, flexio_spi_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO SPI transfer using eDMA.*
- void FLEXIO_SPI_MasterTransferAbortEDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_edma_handle_t ∗handle)
  *Aborts a FlexIO SPI transfer using eDMA.*
- status_t FLEXIO_SPI_MasterTransferGetCountEDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_edma_handle_t ∗handle, size_t ∗count)
  *Gets the remaining bytes for FlexIO SPI eDMA transfer.*
- static void FLEXIO_SPI_SlaveTransferCreateHandleEDMA (FLEXIO_SPI_Type ∗base, flexio_spi_slave_edma_handle_t ∗handle, flexio_spi_slave_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗txHandle, edma_handle_t ∗rxHandle)
  *Initializes the FlexIO SPI slave eDMA handle.*

- status_t FLEXIO_SPI_SlaveTransferEDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_edma_handle_t *handle, flexio_spi_transfer_t *xfer)
  *Performs a non-blocking FlexIO SPI transfer using eDMA.*
- static void FLEXIO_SPI_SlaveTransferAbortEDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_edma_handle_t *handle)
  *Aborts a FlexIO SPI transfer using eDMA.*
- static status_t FLEXIO_SPI_SlaveTransferGetCountEDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_edma_handle_t *handle, size_t *count)
  *Gets the remaining bytes to be transferred for FlexIO SPI eDMA.*

### 23.7.8.2 Data Structure Documentation

#### 23.7.8.2.1 struct _flexio_spi_master_edma_handle

typedef for flexio_spi_master_edma_handle_t in advance.

#### Data Fields

- size_t transferSize
  *Total bytes to be transferred.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- bool txInProgress
  *Send transfer in progress.*
- bool rxInProgress
  *Receive transfer in progress.*
- edma_handle_t * txHandle
  *DMA handler for SPI send.*
- edma_handle_t * rxHandle
  *DMA handler for SPI receive.*
- flexio_spi_master_edma_transfer_callback_t callback
  *Callback for SPI DMA transfer.*
- void * userData
  *User Data for SPI DMA callback.*

#### 23.7.8.2.1.1 Field Documentation

##### 23.7.8.2.1.1.1 size_t flexio_spi_master_edma_handle_t::transferSize

##### 23.7.8.2.1.1.2 uint8_t flexio_spi_master_edma_handle_t::nbytes

### 23.7.8.3 Macro Definition Documentation

#### 23.7.8.3.1 #define FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

### 23.7.8.4 Typedef Documentation

#### 23.7.8.4.1 typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t

### 23.7.8.5 Function Documentation

#### 23.7.8.5.1 status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*, flexio_spi_master_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *txHandle*, edma_handle_t * *rxHandle* )

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state. |
| *callback* | SPI callback, NULL means no callback. |
| *userData* | callback function parameter. |
| *txHandle* | User requested eDMA handle for FlexIO SPI RX eDMA transfer. |
| *rxHandle* | User requested eDMA handle for FlexIO SPI TX eDMA transfer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO SPI eDMA type/handle table out of range. |

#### 23.7.8.5.2 status_t FLEXIO_SPI_MasterTransferEDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*, flexio_spi_transfer_t * *xfer* )

Note

This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state. |
| *xfer* | Pointer to FlexIO SPI transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_SPI_Busy* | FlexIO SPI is not idle, is running another transfer. |

#### 23.7.8.5.3 void FLEXIO_SPI_MasterTransferAbortEDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | FlexIO SPI eDMA handle pointer. |

#### 23.7.8.5.4 status_t FLEXIO_SPI_MasterTransferGetCountEDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | FlexIO SPI eDMA handle pointer. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

#### 23.7.8.5.5 static void FLEXIO_SPI_SlaveTransferCreateHandleEDMA ( FLEXIO_SPI_Type * *base,* flexio_spi_slave_edma_handle_t * *handle,* flexio_spi_slave_edma_transfer_callback_t *callback,* void * *userData,* edma_handle_t * *txHandle,* edma_handle_t * *rxHandle* ) [inline], [static]

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state. |
| *callback* | SPI callback, NULL means no callback. |
| *userData* | callback function parameter. |
| *txHandle* | User requested eDMA handle for FlexIO SPI TX eDMA transfer. |
| *rxHandle* | User requested eDMA handle for FlexIO SPI RX eDMA transfer. |

#### 23.7.8.5.6 status_t FLEXIO_SPI_SlaveTransferEDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_edma_handle_t * *handle*, flexio_spi_transfer_t * *xfer* )

Note

This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state. |
| *xfer* | Pointer to FlexIO SPI transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_SPI_Busy* | FlexIO SPI is not idle, is running another transfer. |

#### 23.7.8.5.7 static void FLEXIO_SPI_SlaveTransferAbortEDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_edma_handle_t * *handle* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state. |

#### 23.7.8.5.8 static status_t FLEXIO_SPI_SlaveTransferGetCountEDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_edma_handle_t * *handle*, size_t * *count* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | FlexIO SPI eDMA handle pointer. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

## 23.7.9 FlexIO DMA SPI Driver

### 23.7.9.1 Overview

#### Data Structures

- struct flexio_spi_master_dma_handle_t
  *FlexIO SPI DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef
  flexio_spi_master_dma_handle_t flexio_spi_slave_dma_handle_t
  *Slave handle is the same with master handle.*
- typedef void(∗ flexio_spi_master_dma_transfer_callback_t )(FLEXIO_SPI_Type ∗base, flexio_spi_master_dma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO SPI master callback for finished transmit.*
- typedef void(∗ flexio_spi_slave_dma_transfer_callback_t )(FLEXIO_SPI_Type ∗base, flexio_spi_slave_dma_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO SPI slave callback for finished transmit.*

#### Driver version

- #define FSL_FLEXIO_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  *FlexIO SPI DMA driver version 2.1.3.*

#### DMA Transactional

- status_t FLEXIO_SPI_MasterTransferCreateHandleDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_dma_handle_t ∗handle, flexio_spi_master_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  *Initializes the FLEXO SPI master DMA handle.*
- status_t FLEXIO_SPI_MasterTransferDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_dma_handle_t ∗handle, flexio_spi_transfer_t ∗xfer)
  *Performs a non-blocking FlexIO SPI transfer using DMA.*
- void FLEXIO_SPI_MasterTransferAbortDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_dma_handle_t ∗handle)
  *Aborts a FlexIO SPI transfer using DMA.*
- status_t FLEXIO_SPI_MasterTransferGetCountDMA (FLEXIO_SPI_Type ∗base, flexio_spi_master_dma_handle_t ∗handle, size_t ∗count)
  *Gets the remaining bytes for FlexIO SPI DMA transfer.*
- static void FLEXIO_SPI_SlaveTransferCreateHandleDMA (FLEXIO_SPI_Type ∗base, flexio_spi_slave_dma_handle_t ∗handle, flexio_spi_slave_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  *Initializes the FlexIO SPI slave DMA handle.*

- status_t FLEXIO_SPI_SlaveTransferDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_dma_handle_t *handle, flexio_spi_transfer_t *xfer)
  *Performs a non-blocking FlexIO SPI transfer using DMA.*
- static void FLEXIO_SPI_SlaveTransferAbortDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_dma_handle_t *handle)
  *Aborts a FlexIO SPI transfer using DMA.*
- static status_t FLEXIO_SPI_SlaveTransferGetCountDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_dma_handle_t *handle, size_t *count)
  *Gets the remaining bytes to be transferred for FlexIO SPI DMA.*

### 23.7.9.2 Data Structure Documentation

#### 23.7.9.2.1 struct _flexio_spi_master_dma_handle

typedef for flexio_spi_master_dma_handle_t in advance.

#### Data Fields

- size_t transferSize
  *Total bytes to be transferred.*
- bool txInProgress
  *Send transfer in progress.*
- bool rxInProgress
  *Receive transfer in progress.*
- dma_handle_t * txHandle
  *DMA handler for SPI send.*
- dma_handle_t * rxHandle
  *DMA handler for SPI receive.*
- flexio_spi_master_dma_transfer_callback_t callback
  *Callback for SPI DMA transfer.*
- void * userData
  *User Data for SPI DMA callback.*

#### 23.7.9.2.1.1 Field Documentation

##### 23.7.9.2.1.1.1 size_t flexio_spi_master_dma_handle_t::transferSize

### 23.7.9.3 Macro Definition Documentation

#### 23.7.9.3.1 #define FSL_FLEXIO_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

### 23.7.9.4 Typedef Documentation

#### 23.7.9.4.1 typedef flexio_spi_master_dma_handle_t flexio_spi_slave_dma_handle_t

### 23.7.9.5 Function Documentation

#### 23.7.9.5.1 status_t FLEXIO_SPI_MasterTransferCreateHandleDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_dma_handle_t * *handle*, flexio_spi_master_dma_transfer_callback_t *callback*, void * *userData*, dma_handle_t * *txHandle*, dma_handle_t * *rxHandle* )

This function initializes the FLEXO SPI master DMA handle which can be used for other FLEXO SPI master transactional APIs. Usually, for a specified FLEXO SPI instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_master_dma_handle_t structure to store the transfer state. |
| *callback* | SPI callback, NULL means no callback. |
| *userData* | callback function parameter. |
| *txHandle* | User requested DMA handle for FlexIO SPI RX DMA transfer. |
| *rxHandle* | User requested DMA handle for FlexIO SPI TX DMA transfer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO SPI DMA type/handle table out of range. |

#### 23.7.9.5.2 status_t FLEXIO_SPI_MasterTransferDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_dma_handle_t * *handle*, flexio_spi_transfer_t * *xfer* )

Note

This interface returned immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountDMA to poll the transfer status to check whether the FlexIO SPI transfer is finished.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_master_dma_handle_t structure to store the transfer state. |
| *xfer* | Pointer to FlexIO SPI transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_SPI_Busy* | FlexIO SPI is not idle, is running another transfer. |

#### 23.7.9.5.3 void FLEXIO_SPI_MasterTransferAbortDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_dma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | FlexIO SPI DMA handle pointer. |

#### 23.7.9.5.4 status_t FLEXIO_SPI_MasterTransferGetCountDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_master_dma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | FlexIO SPI DMA handle pointer. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

#### 23.7.9.5.5 static void FLEXIO_SPI_SlaveTransferCreateHandleDMA ( FLEXIO_SPI_Type * *base,* flexio_spi_slave_dma_handle_t * *handle,* flexio_spi_slave_dma_transfer_callback_t *callback,* void * *userData,* dma_handle_t * *txHandle,* dma_handle_t * *rxHandle* ) [inline], [static]

This function initializes the FlexIO SPI slave DMA handle.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_slave_dma_handle_t structure to store the transfer state. |
| *callback* | SPI callback, NULL means no callback. |
| *userData* | callback function parameter. |
| *txHandle* | User requested DMA handle for FlexIO SPI TX DMA transfer. |
| *rxHandle* | User requested DMA handle for FlexIO SPI RX DMA transfer. |

#### 23.7.9.5.6 status_t FLEXIO_SPI_SlaveTransferDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_dma_handle_t * *handle*, flexio_spi_transfer_t * *xfer* )

Note

This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_slave_dma_handle_t structure to store the transfer state. |
| *xfer* | Pointer to FlexIO SPI transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start a transfer. |
| *kStatus_InvalidArgument* | Input argument is invalid. |
| *kStatus_FLEXIO_SPI_Busy* | FlexIO SPI is not idle, is running another transfer. |

#### 23.7.9.5.7 static void FLEXIO_SPI_SlaveTransferAbortDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_dma_handle_t * *handle* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | Pointer to flexio_spi_slave_dma_handle_t structure to store the transfer state. |

#### 23.7.9.5.8 static status_t FLEXIO_SPI_SlaveTransferGetCountDMA ( FLEXIO_SPI_Type * *base*, flexio_spi_slave_dma_handle_t * *handle*, size_t * *count* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_SPI_Type structure. |
| *handle* | FlexIO SPI DMA handle pointer. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

## 23.8 FlexIO UART Driver

### 23.8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) function using the Flexible I/O.

FlexIO UART driver includes functional APIs and transactional APIs. Functional APIs target low-level APIs. Functional APIs can be used for the FlexIO UART initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO UART peripheral and how to organize functional APIs to meet the application requirements. All functional API use the FLEXIO_UART_Type * as the first parameter. FlexIO UART functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the flexio_uart_handle_t as the second parameter. Initialize the handle by calling the FLEXIO_UART_TransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions FLEXIO_UART_SendNonBlocking() and FLEXIO_UART_ReceiveNonBlocking() set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the kStatus_FLEXIO_UART_TxIdle and kStatus_FLEXIO_UART_RxIdle status.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size through calling the FLEXIO_UART_InstallRingBuffer(). When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The function FLEXIO_UART_ReceiveNonBlocking() first gets data the from the ring buffer. If ring buffer does not have enough data, the function returns the data to the ring buffer and saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the statuskStatus_FLEXIO_UART_RxIdle status.

If the receive ring buffer is full, the upper layer is informed through a callback with status kStatus_FLEXIO_UART_RxRingBufferOverrun. In the callback function, the upper layer reads data from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when calling the FLEXIO_UART_InstallRingBuffer. Note that one byte is reserved for the ring buffer maintenance. Create a handle as follows.

```
FLEXIO_UART_InstallRingBuffer(&uartDev, &handle, &ringBuffer, 32);
```

In this example, the buffer size is 32. However, only 31 bytes are used for saving data.

### 23.8.2 Typical use case

#### 23.8.2.1 FlexIO UART send/receive using a polling method

```
uint8_t ch;
```

```
FLEXIO_UART_Type uartDev;
status_t result = kStatus_Success;
flexio_uart_user_config user_config;
FLEXIO_UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableUart = true;

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
    return;
}
FLEXIO_UART_WriteBlocking(&uartDev, txbuff, sizeof(txbuff));

while(1)
{
    FLEXIO_UART_ReadBlocking(&uartDev, &ch, 1);
    FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
}
```

#### 23.8.2.2 FlexIO UART send/receive using an interrupt method

```
FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
      status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;
```

```
    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;

    result = FLEXIO_UART_Init(&uartDev, &user_config, 120000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
      FLEXIO_UART_UserCallback, NULL);

    // Prepares to send.
    sendXfer.data = sendData;
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_UART_SendNonBlocking(&uartDev, &g_uartHandle, &sendXfer);

    // Send finished.
    while (!txFinished)
    {
    }

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, NULL);

    // Receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

#### 23.8.2.3 FlexIO UART receive using the ringbuffer feature

```
#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE     32

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
      status_t status, void *userData)
{
```

```
    userData = userData;

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;

    result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
      FLEXIO_UART_UserCallback, NULL);
    FLEXIO_UART_InstallRingBuffer(&uartDev, &g_uartHandle, ringBuffer, RING_BUFFER_SIZE);

    // Receive is working in the background to the ring buffer.

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = RX_DATA_SIZE;
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, &bytesRead);

    if (bytesRead = RX_DATA_SIZE) /* Have read enough data. */
    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}
```

#### 23.8.2.4 FlexIO UART send/receive using a DMA method

```
    FLEXIO_UART_Type uartDev;
    flexio_uart_handle_t g_uartHandle;
    dma_handle_t g_uartTxDmaHandle;
    dma_handle_t g_uartRxDmaHandle;
    flexio_uart_config_t user_config;
    flexio_uart_transfer_t sendXfer;
    flexio_uart_transfer_t receiveXfer;
    volatile bool txFinished;
    volatile bool rxFinished;
    uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
    uint8_t receiveData[32];

    void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
      status_t status, void *userData)
    {
        userData = userData;

        if (kStatus_FLEXIO_UART_TxIdle == status)
        {
            txFinished = true;
        }

        if (kStatus_FLEXIO_UART_RxIdle == status)
        {
            rxFinished = true;
        }
    }

    void main(void)
    {
        //...

        FLEXIO_UART_GetDefaultConfig(&user_config);
        user_config.baudRate_Bps = 115200U;
        user_config.enableUart = true;

        uartDev.flexioBase = BOARD_FLEXIO_BASE;
        uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
        uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
        uartDev.shifterIndex[0] = 0U;
        uartDev.shifterIndex[1] = 1U;
        uartDev.timerIndex[0] = 0U;
        uartDev.timerIndex[1] = 1U;
        result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
        //Check if configuration is correct.
        if(result != kStatus_Success)
        {
            return;
        }

        /* Init DMAMUX. */
        DMAMUX_Init(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR)

        /* Init the DMA/EDMA module */
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
        DMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR);
        DMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL);
        DMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
        edma_config_t edmaConfig;

        EDMA_GetDefaultConfig(&edmaConfig);
        EDMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR, &edmaConfig);
```

```
    EDMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
  FLEXIO_UART_TX_DMA_CHANNEL);
    EDMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
  FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

    dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
  shifterIndex[0]);
    dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
  shifterIndex[1]);

    /* Requests  DMA channels for transmit and receive. */
    DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL, (
  dma_request_source_t)dma_request_source_tx);
    DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL, (
  dma_request_source_t)dma_request_source_rx);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
  FLEXIO_UART_TX_DMA_CHANNEL);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
  FLEXIO_UART_RX_DMA_CHANNEL);

    FLEXIO_UART_TransferCreateHandleDMA(&uartDev, &g_uartHandle,
  FLEXIO_UART_UserCallback, NULL, &g_uartTxDmaHandle, &g_uartRxDmaHandle);

    // Prepares to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_UART_SendDMA(&uartDev, &g_uartHandle, &sendXfer);

    // Send finished.
    while (!txFinished)
    {
    }

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveDMA(&uartDev, &g_uartHandle, &receiveXfer, NULL);

    // Receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

## Modules

- FlexIO DMA UART Driver
- FlexIO eDMA UART Driver

## Data Structures

- struct FLEXIO_UART_Type

*Define FlexIO UART access structure typedef. More...*
- struct flexio_uart_config_t
  *Define FlexIO UART user configuration structure. More...*
- struct flexio_uart_transfer_t
  *Define FlexIO UART transfer structure. More...*
- struct flexio_uart_handle_t
  *Define FLEXIO UART handle structure. More...*

## Typedefs

- typedef void(∗ flexio_uart_transfer_callback_t )(FLEXIO_UART_Type ∗base, flexio_uart_handle_t ∗handle, status_t status, void ∗userData)
  *FlexIO UART transfer callback function.*

## Enumerations

- enum _flexio_uart_status {
  kStatus_FLEXIO_UART_TxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 0),
  kStatus_FLEXIO_UART_RxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 1),
  kStatus_FLEXIO_UART_TxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 2),
  kStatus_FLEXIO_UART_RxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 3),
  kStatus_FLEXIO_UART_ERROR = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 4),
  kStatus_FLEXIO_UART_RxRingBufferOverrun,
  kStatus_FLEXIO_UART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 6) }
  *Error codes for the UART driver.*
- enum flexio_uart_bit_count_per_char_t {
  kFLEXIO_UART_7BitsPerChar = 7U,
  kFLEXIO_UART_8BitsPerChar = 8U,
  kFLEXIO_UART_9BitsPerChar = 9U }
  *FlexIO UART bit count per char.*
- enum _flexio_uart_interrupt_enable {
  kFLEXIO_UART_TxDataRegEmptyInterruptEnable = 0x1U,
  kFLEXIO_UART_RxDataRegFullInterruptEnable = 0x2U }
  *Define FlexIO UART interrupt mask.*
- enum _flexio_uart_status_flags {
  kFLEXIO_UART_TxDataRegEmptyFlag = 0x1U,
  kFLEXIO_UART_RxDataRegFullFlag = 0x2U,
  kFLEXIO_UART_RxOverRunFlag = 0x4U }
  *Define FlexIO UART status mask.*

## Driver version

- #define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))
  *FlexIO UART driver version 2.1.5.*

### Initialization and deinitialization

- status_t FLEXIO_UART_Init (FLEXIO_UART_Type *base, const flexio_uart_config_t *userConfig, uint32_t srcClock_Hz)
  *Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration.*
- void FLEXIO_UART_Deinit (FLEXIO_UART_Type *base)
  *Resets the FlexIO UART shifter and timer config.*
- void FLEXIO_UART_GetDefaultConfig (flexio_uart_config_t *userConfig)
  *Gets the default configuration to configure the FlexIO UART.*

### Status

- uint32_t FLEXIO_UART_GetStatusFlags (FLEXIO_UART_Type *base)
  *Gets the FlexIO UART status flags.*
- void FLEXIO_UART_ClearStatusFlags (FLEXIO_UART_Type *base, uint32_t mask)
  *Gets the FlexIO UART status flags.*

### Interrupts

- void FLEXIO_UART_EnableInterrupts (FLEXIO_UART_Type *base, uint32_t mask)
  *Enables the FlexIO UART interrupt.*
- void FLEXIO_UART_DisableInterrupts (FLEXIO_UART_Type *base, uint32_t mask)
  *Disables the FlexIO UART interrupt.*

### DMA Control

- static uint32_t FLEXIO_UART_GetTxDataRegisterAddress (FLEXIO_UART_Type *base)
  *Gets the FlexIO UARt transmit data register address.*
- static uint32_t FLEXIO_UART_GetRxDataRegisterAddress (FLEXIO_UART_Type *base)
  *Gets the FlexIO UART receive data register address.*
- static void FLEXIO_UART_EnableTxDMA (FLEXIO_UART_Type *base, bool enable)
  *Enables/disables the FlexIO UART transmit DMA.*
- static void FLEXIO_UART_EnableRxDMA (FLEXIO_UART_Type *base, bool enable)
  *Enables/disables the FlexIO UART receive DMA.*

### Bus Operations

- static void FLEXIO_UART_Enable (FLEXIO_UART_Type *base, bool enable)
  *Enables/disables the FlexIO UART module operation.*
- static void FLEXIO_UART_WriteByte (FLEXIO_UART_Type *base, const uint8_t *buffer)
  *Writes one byte of data.*
- static void FLEXIO_UART_ReadByte (FLEXIO_UART_Type *base, uint8_t *buffer)
  *Reads one byte of data.*

- void FLEXIO_UART_WriteBlocking (FLEXIO_UART_Type *base, const uint8_t *txData, size_t txSize)
  *Sends a buffer of data bytes.*
- void FLEXIO_UART_ReadBlocking (FLEXIO_UART_Type *base, uint8_t *rxData, size_t rxSize)
  *Receives a buffer of bytes.*

### Transactional

- status_t FLEXIO_UART_TransferCreateHandle (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_callback_t callback, void *userData)
  *Initializes the UART handle.*
- void FLEXIO_UART_TransferStartRingBuffer (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  *Sets up the RX ring buffer.*
- void FLEXIO_UART_TransferStopRingBuffer (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
  *Aborts the background transfer and uninstalls the ring buffer.*
- status_t FLEXIO_UART_TransferSendNonBlocking (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer)
  *Transmits a buffer of data using the interrupt method.*
- void FLEXIO_UART_TransferAbortSend (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
  *Aborts the interrupt-driven data transmit.*
- status_t FLEXIO_UART_TransferGetSendCount (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
  *Gets the number of bytes sent.*
- status_t FLEXIO_UART_TransferReceiveNonBlocking (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer, size_t *receivedBytes)
  *Receives a buffer of data using the interrupt method.*
- void FLEXIO_UART_TransferAbortReceive (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
  *Aborts the receive data which was using IRQ.*
- status_t FLEXIO_UART_TransferGetReceiveCount (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
  *Gets the number of bytes received.*
- void FLEXIO_UART_TransferHandleIRQ (void *uartType, void *uartHandle)
  *FlexIO UART IRQ handler function.*

## 23.8.3 Data Structure Documentation

### 23.8.3.1 struct FLEXIO_UART_Type

### Data Fields

- FLEXIO_Type * flexioBase

*FlexIO base pointer.*
- uint8_t TxPinIndex
  *Pin select for UART_Tx.*
- uint8_t RxPinIndex
  *Pin select for UART_Rx.*
- uint8_t shifterIndex [2]
  *Shifter index used in FlexIO UART.*
- uint8_t timerIndex [2]
  *Timer index used in FlexIO UART.*

#### 23.8.3.1.0.1 Field Documentation

##### 23.8.3.1.0.1.1 FLEXIO_Type* FLEXIO_UART_Type::flexioBase

##### 23.8.3.1.0.1.2 uint8_t FLEXIO_UART_Type::TxPinIndex

##### 23.8.3.1.0.1.3 uint8_t FLEXIO_UART_Type::RxPinIndex

##### 23.8.3.1.0.1.4 uint8_t FLEXIO_UART_Type::shifterIndex[2]

##### 23.8.3.1.0.1.5 uint8_t FLEXIO_UART_Type::timerIndex[2]

### 23.8.3.2 struct flexio_uart_config_t

### Data Fields

- bool enableUart
  *Enable/disable FlexIO UART TX & RX.*
- bool enableInDoze
  *Enable/disable FlexIO operation in doze mode.*
- bool enableInDebug
  *Enable/disable FlexIO operation in debug mode.*
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  *fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32_t baudRate_Bps
  *Baud rate in Bps.*
- flexio_uart_bit_count_per_char_t bitCountPerChar
  *number of bits, 7/8/9 -bit*

#### 23.8.3.2.0.2 Field Documentation

##### 23.8.3.2.0.2.1 bool flexio_uart_config_t::enableUart

##### 23.8.3.2.0.2.2 bool flexio_uart_config_t::enableFastAccess

##### 23.8.3.2.0.2.3 uint32_t flexio_uart_config_t::baudRate_Bps

### 23.8.3.3 struct flexio_uart_transfer_t

#### Data Fields

- uint8_t * data
  *Transfer buffer.*
- size_t dataSize
  *Transfer size.*

### 23.8.3.4 struct _flexio_uart_handle

#### Data Fields

- uint8_t *volatile txData
  *Address of remaining data to send.*
- volatile size_t txDataSize
  *Size of the remaining data to send.*
- uint8_t *volatile rxData
  *Address of remaining data to receive.*
- volatile size_t rxDataSize
  *Size of the remaining data to receive.*
- size_t txDataSizeAll
  *Total bytes to be sent.*
- size_t rxDataSizeAll
  *Total bytes to be received.*
- uint8_t * rxRingBuffer
  *Start address of the receiver ring buffer.*
- size_t rxRingBufferSize
  *Size of the ring buffer.*
- volatile uint16_t rxRingBufferHead
  *Index for the driver to store received data into ring buffer.*
- volatile uint16_t rxRingBufferTail
  *Index for the user to get data from the ring buffer.*
- flexio_uart_transfer_callback_t callback
  *Callback function.*
- void * userData
  *UART callback function parameter.*
- volatile uint8_t txState
  *TX transfer state.*
- volatile uint8_t rxState
  *RX transfer state.*

#### 23.8.3.4.0.3 Field Documentation

##### 23.8.3.4.0.3.1 uint8_t* volatile flexio_uart_handle_t::txData

##### 23.8.3.4.0.3.2 volatile size_t flexio_uart_handle_t::txDataSize

##### 23.8.3.4.0.3.3 uint8_t* volatile flexio_uart_handle_t::rxData

##### 23.8.3.4.0.3.4 volatile size_t flexio_uart_handle_t::rxDataSize

##### 23.8.3.4.0.3.5 size_t flexio_uart_handle_t::txDataSizeAll

##### 23.8.3.4.0.3.6 size_t flexio_uart_handle_t::rxDataSizeAll

##### 23.8.3.4.0.3.7 uint8_t* flexio_uart_handle_t::rxRingBuffer

##### 23.8.3.4.0.3.8 size_t flexio_uart_handle_t::rxRingBufferSize

##### 23.8.3.4.0.3.9 volatile uint16_t flexio_uart_handle_t::rxRingBufferHead

##### 23.8.3.4.0.3.10 volatile uint16_t flexio_uart_handle_t::rxRingBufferTail

##### 23.8.3.4.0.3.11 flexio_uart_transfer_callback_t flexio_uart_handle_t::callback

##### 23.8.3.4.0.3.12 void* flexio_uart_handle_t::userData

##### 23.8.3.4.0.3.13 volatile uint8_t flexio_uart_handle_t::txState

### 23.8.4 Macro Definition Documentation

#### 23.8.4.1 #define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))

### 23.8.5 Typedef Documentation

#### 23.8.5.1 typedef void(* flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, status_t status, void *userData)

### 23.8.6 Enumeration Type Documentation

#### 23.8.6.1 enum _flexio_uart_status

Enumerator

***kStatus_FLEXIO_UART_TxBusy*** Transmitter is busy.
***kStatus_FLEXIO_UART_RxBusy*** Receiver is busy.
***kStatus_FLEXIO_UART_TxIdle*** UART transmitter is idle.
***kStatus_FLEXIO_UART_RxIdle*** UART receiver is idle.
***kStatus_FLEXIO_UART_ERROR*** ERROR happens on UART.

*kStatus_FLEXIO_UART_RxRingBufferOverrun* UART RX software ring buffer overrun.
*kStatus_FLEXIO_UART_RxHardwareOverrun* UART RX receiver overrun.

#### 23.8.6.2 enum flexio_uart_bit_count_per_char_t

Enumerator

*kFLEXIO_UART_7BitsPerChar* 7-bit data characters
*kFLEXIO_UART_8BitsPerChar* 8-bit data characters
*kFLEXIO_UART_9BitsPerChar* 9-bit data characters

#### 23.8.6.3 enum _flexio_uart_interrupt_enable

Enumerator

*kFLEXIO_UART_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.
*kFLEXIO_UART_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

#### 23.8.6.4 enum _flexio_uart_status_flags

Enumerator

*kFLEXIO_UART_TxDataRegEmptyFlag* Transmit buffer empty flag.
*kFLEXIO_UART_RxDataRegFullFlag* Receive buffer full flag.
*kFLEXIO_UART_RxOverRunFlag* Receive buffer over run flag.

### 23.8.7 Function Documentation

#### 23.8.7.1 status_t FLEXIO_UART_Init ( FLEXIO_UART_Type * *base*, const flexio_uart_config_t * *userConfig*, uint32_t *srcClock_Hz* )

The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_GetDefaultConfig().

Example

```
FLEXIO_UART_Type base = {
.flexioBase = FLEXIO,
.TxPinIndex = 0,
.RxPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_uart_config_t config = {
.enableInDoze = false,
```

```
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *userConfig* | Pointer to the flexio_uart_config_t structure. |
| *srcClock_Hz* | FlexIO source clock in Hz. |

Return values

| | |
|---|---|
| *kStatus_Success* | Configuration success |
| *kStatus_InvalidArgument* | Buadrate configuration out of range |

#### 23.8.7.2 void FLEXIO_UART_Deinit ( FLEXIO_UART_Type * *base* )

Note

After calling this API, call the FLEXO_UART_Init to use the FlexIO UART module.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure |

#### 23.8.7.3 void FLEXIO_UART_GetDefaultConfig ( flexio_uart_config_t * *userConfig* )

The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

| | |
|---|---|
| *userConfig* | Pointer to the flexio_uart_config_t structure. |

#### 23.8.7.4 uint32_t FLEXIO_UART_GetStatusFlags ( FLEXIO_UART_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |

Returns

FlexIO UART status flags.

#### 23.8.7.5 void FLEXIO_UART_ClearStatusFlags ( FLEXIO_UART_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *mask* | Status flag. The parameter can be any combination of the following values:<br>• kFLEXIO_UART_TxDataRegEmptyFlag<br>• kFLEXIO_UART_RxEmptyFlag<br>• kFLEXIO_UART_RxOverRunFlag |

#### 23.8.7.6 void FLEXIO_UART_EnableInterrupts ( FLEXIO_UART_Type * *base*, uint32_t *mask* )

This function enables the FlexIO UART interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *mask* | Interrupt source. |

#### 23.8.7.7 void FLEXIO_UART_DisableInterrupts ( FLEXIO_UART_Type * *base*, uint32_t *mask* )

This function disables the FlexIO UART interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *mask* | Interrupt source. |

#### 23.8.7.8 static uint32_t FLEXIO_UART_GetTxDataRegisterAddress ( FLEXIO_UART_Type * *base* ) [inline], [static]

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |

Returns

FlexIO UART transmit data register address.

#### 23.8.7.9 static uint32_t FLEXIO_UART_GetRxDataRegisterAddress ( FLEXIO_UART_Type * *base* ) [inline], [static]

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |

Returns

FlexIO UART receive data register address.

#### 23.8.7.10 static void FLEXIO_UART_EnableTxDMA ( FLEXIO_UART_Type * *base*, bool *enable* ) [inline], [static]

This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *enable* | True to enable, false to disable. |

#### 23.8.7.11 static void FLEXIO_UART_EnableRxDMA ( FLEXIO_UART_Type * *base,* bool *enable* ) [inline], [static]

This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_-RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *enable* | True to enable, false to disable. |

### 23.8.7.12 static void FLEXIO_UART_Enable ( FLEXIO_UART_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type. |
| *enable* | True to enable, false does not have any effect. |

### 23.8.7.13 static void FLEXIO_UART_WriteByte ( FLEXIO_UART_Type * *base*, const uint8_t * *buffer* ) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *buffer* | The data bytes to send. |

### 23.8.7.14 static void FLEXIO_UART_ReadByte ( FLEXIO_UART_Type * *base*, uint8_t * *buffer* ) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *buffer* | The buffer to store the received bytes. |

#### 23.8.7.15 void FLEXIO_UART_WriteBlocking ( FLEXIO_UART_Type * *base*, const uint8_t * *txData*, size_t *txSize* )

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *txData* | The data bytes to send. |
| *txSize* | The number of data bytes to send. |

#### 23.8.7.16 void FLEXIO_UART_ReadBlocking ( FLEXIO_UART_Type * *base*, uint8_t * *rxData*, size_t *rxSize* )

Note

This function blocks using the polling method until all bytes have been received.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *rxData* | The buffer to store the received bytes. |
| *rxSize* | The number of data bytes to be received. |

#### 23.8.7.17 status_t FLEXIO_UART_TransferCreateHandle ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_callback_t *callback*, void * *userData* )

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

| | |
|---|---|
| *base* | to FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO type/handle/ISR table out of range. |

### 23.8.7.18 void FLEXIO_UART_TransferStartRingBuffer ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, uint8_t * *ringBuffer*, size_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |
| *ringBuffer* | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| *ringBufferSize* | Size of the ring buffer. |

### 23.8.7.19 void FLEXIO_UART_TransferStopRingBuffer ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |

#### 23.8.7.20 status_t FLEXIO_UART_TransferSendNonBlocking ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_t * *xfer* )

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note

The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |
| *xfer* | FlexIO UART transfer structure. See flexio_uart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully starts the data transmission. |
| *kStatus_UART_TxBusy* | Previous transmission still not finished, data not written to the TX register. |

#### 23.8.7.21 void FLEXIO_UART_TransferAbortSend ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle* )

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |

#### 23.8.7.22 status_t FLEXIO_UART_TransferGetSendCount ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, size_t * *count* )

This function gets the number of bytes sent driven by interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |
| *count* | Number of bytes sent so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | transfer has finished or no transfer in progress. |
| *kStatus_Success* | Successfully return the count. |

#### 23.8.7.23 status_t FLEXIO_UART_TransferReceiveNonBlocking ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_t * *xfer*, size_t * *receivedBytes* )

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to xfer->data. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |
| *xfer* | UART transfer structure. See flexio_uart_transfer_t. |
| *receivedBytes* | Bytes received from the ring buffer directly. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully queue the transfer into the transmit queue. |
| *kStatus_FLEXIO_UART-_RxBusy* | Previous receive request is not finished. |

#### 23.8.7.24 void FLEXIO_UART_TransferAbortReceive ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle* )

This function aborts the receive data which was using IRQ.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |

#### 23.8.7.25 status_t FLEXIO_UART_TransferGetReceiveCount ( FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, size_t * *count* )

This function gets the number of bytes received driven by interrupt.

Parameters

| | |
|---|---|
| *base* | Pointer to the FLEXIO_UART_Type structure. |
| *handle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |
| *count* | Number of bytes received so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_NoTransferIn-Progress* | transfer has finished or no transfer in progress. |
| *kStatus_Success* | Successfully return the count. |

#### 23.8.7.26 void FLEXIO_UART_TransferHandleIRQ ( void * *uartType,* void * *uartHandle* )

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

| | |
|---|---|
| *uartType* | Pointer to the FLEXIO_UART_Type structure. |
| *uartHandle* | Pointer to the flexio_uart_handle_t structure to store the transfer state. |

## 23.8.8 FlexIO eDMA UART Driver

### 23.8.8.1 Overview

#### Data Structures

- struct flexio_uart_edma_handle_t
  *UART eDMA handle. More...*

#### Typedefs

- typedef void(∗ flexio_uart_edma_transfer_callback_t )(FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle, status_t status, void ∗userData)
  *UART transfer callback function.*

#### Driver version

- #define FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))
  *FlexIO UART EDMA driver version 2.1.4.*

#### eDMA transactional

- status_t FLEXIO_UART_TransferCreateHandleEDMA (FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle, flexio_uart_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗txEdmaHandle, edma_handle_t ∗rxEdmaHandle)
  *Initializes the UART handle which is used in transactional functions.*
- status_t FLEXIO_UART_TransferSendEDMA (FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle, flexio_uart_transfer_t ∗xfer)
  *Sends data using eDMA.*
- status_t FLEXIO_UART_TransferReceiveEDMA (FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle, flexio_uart_transfer_t ∗xfer)
  *Receives data using eDMA.*
- void FLEXIO_UART_TransferAbortSendEDMA (FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle)
  *Aborts the sent data which using eDMA.*
- void FLEXIO_UART_TransferAbortReceiveEDMA (FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle)
  *Aborts the receive data which using eDMA.*
- status_t FLEXIO_UART_TransferGetSendCountEDMA (FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle, size_t ∗count)
  *Gets the number of bytes sent out.*
- status_t FLEXIO_UART_TransferGetReceiveCountEDMA (FLEXIO_UART_Type ∗base, flexio_uart_edma_handle_t ∗handle, size_t ∗count)
  *Gets the number of bytes received.*

### 23.8.8.2 Data Structure Documentation

#### 23.8.8.2.1 struct _flexio_uart_edma_handle

**Data Fields**

- flexio_uart_edma_transfer_callback_t callback
  *Callback function.*
- void * userData
  *UART callback function parameter.*
- size_t txDataSizeAll
  *Total bytes to be sent.*
- size_t rxDataSizeAll
  *Total bytes to be received.*
- edma_handle_t * txEdmaHandle
  *The eDMA TX channel used.*
- edma_handle_t * rxEdmaHandle
  *The eDMA RX channel used.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- volatile uint8_t txState
  *TX transfer state.*
- volatile uint8_t rxState
  *RX transfer state.*

#### 23.8.8.2.1.1 Field Documentation

##### 23.8.8.2.1.1.1 flexio_uart_edma_transfer_callback_t flexio_uart_edma_handle_t::callback

##### 23.8.8.2.1.1.2 void* flexio_uart_edma_handle_t::userData

##### 23.8.8.2.1.1.3 size_t flexio_uart_edma_handle_t::txDataSizeAll

##### 23.8.8.2.1.1.4 size_t flexio_uart_edma_handle_t::rxDataSizeAll

##### 23.8.8.2.1.1.5 edma_handle_t* flexio_uart_edma_handle_t::txEdmaHandle

##### 23.8.8.2.1.1.6 edma_handle_t* flexio_uart_edma_handle_t::rxEdmaHandle

##### 23.8.8.2.1.1.7 uint8_t flexio_uart_edma_handle_t::nbytes

##### 23.8.8.2.1.1.8 volatile uint8_t flexio_uart_edma_handle_t::txState

### 23.8.8.3 Macro Definition Documentation

#### 23.8.8.3.1 #define FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))

### 23.8.8.4 Typedef Documentation

#### 23.8.8.4.1 typedef void(* flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_edma_handle_t *handle, status_t status, void *userData)

### 23.8.8.5 Function Documentation

#### 23.8.8.5.1 status_t FLEXIO_UART_TransferCreateHandleEDMA ( FLEXIO_UART_Type * *base,* flexio_uart_edma_handle_t * *handle,* flexio_uart_edma_transfer_callback_t *callback,* void * *userData,* edma_handle_t * *txEdmaHandle,* edma_handle_t * *rxEdmaHandle* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type. |
| *handle* | Pointer to flexio_uart_edma_handle_t structure. |
| *callback* | The callback function. |
| *userData* | The parameter of the callback function. |
| *rxEdmaHandle* | User requested DMA handle for RX DMA transfer. |
| *txEdmaHandle* | User requested DMA handle for TX DMA transfer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO SPI eDMA type/handle table out of range. |

#### 23.8.8.5.2 status_t FLEXIO_UART_TransferSendEDMA ( FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*, flexio_uart_transfer_t * *xfer* )

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type |
| *handle* | UART handle pointer. |
| *xfer* | UART eDMA transfer structure, see flexio_uart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_FLEXIO_UART-_TxBusy* | Previous transfer on going. |

#### 23.8.8.5.3 status_t FLEXIO_UART_TransferReceiveEDMA ( FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*, flexio_uart_transfer_t * *xfer* )

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type |
| *handle* | Pointer to flexio_uart_edma_handle_t structure |
| *xfer* | UART eDMA transfer structure, see flexio_uart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_UART_RxBusy* | Previous transfer on going. |

#### 23.8.8.5.4 void FLEXIO_UART_TransferAbortSendEDMA ( FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle* )

This function aborts sent data which using eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type |
| *handle* | Pointer to flexio_uart_edma_handle_t structure |

#### 23.8.8.5.5 void FLEXIO_UART_TransferAbortReceiveEDMA ( FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle* )

This function aborts the receive data which using eDMA.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type |
| *handle* | Pointer to flexio_uart_edma_handle_t structure |

#### 23.8.8.5.6 status_t FLEXIO_UART_TransferGetSendCountEDMA ( FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*, size_t * *count* )

This function gets the number of bytes sent out.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type |
| *handle* | Pointer to flexio_uart_edma_handle_t structure |
| *count* | Number of bytes sent so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | transfer has finished or no transfer in progress. |
| *kStatus_Success* | Successfully return the count. |

#### 23.8.8.5.7 status_t FLEXIO_UART_TransferGetReceiveCountEDMA ( FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*, size_t * *count* )

This function gets the number of bytes received.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type |
| *handle* | Pointer to flexio_uart_edma_handle_t structure |
| *count* | Number of bytes received so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | transfer has finished or no transfer in progress. |
| *kStatus_Success* | Successfully return the count. |

## 23.8.9 FlexIO DMA UART Driver

### 23.8.9.1 Overview

#### Data Structures

- struct flexio_uart_dma_handle_t
  *UART DMA handle. More...*

#### Typedefs

- typedef void(∗ flexio_uart_dma_transfer_callback_t )(FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle, status_t status, void ∗userData)
  *UART transfer callback function.*

#### Driver version

- #define FSL_FLEXIO_UART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))
  *FlexIO UART DMA driver version 2.1.4.*

#### eDMA transactional

- status_t FLEXIO_UART_TransferCreateHandleDMA (FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle, flexio_uart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  *Initializes the FLEXIO_UART handle which is used in transactional functions.*
- status_t FLEXIO_UART_TransferSendDMA (FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle, flexio_uart_transfer_t ∗xfer)
  *Sends data using DMA.*
- status_t FLEXIO_UART_TransferReceiveDMA (FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle, flexio_uart_transfer_t ∗xfer)
  *Receives data using DMA.*
- void FLEXIO_UART_TransferAbortSendDMA (FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle)
  *Aborts the sent data which using DMA.*
- void FLEXIO_UART_TransferAbortReceiveDMA (FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle)
  *Aborts the receive data which using DMA.*
- status_t FLEXIO_UART_TransferGetSendCountDMA (FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle, size_t ∗count)
  *Gets the number of bytes sent out.*
- status_t FLEXIO_UART_TransferGetReceiveCountDMA (FLEXIO_UART_Type ∗base, flexio_uart_dma_handle_t ∗handle, size_t ∗count)
  *Gets the number of bytes received.*

### 23.8.9.2 Data Structure Documentation

#### 23.8.9.2.1 struct _flexio_uart_dma_handle

**Data Fields**

- flexio_uart_dma_transfer_callback_t callback
  *Callback function.*
- void * userData
  *UART callback function parameter.*
- size_t txDataSizeAll
  *Total bytes to be sent.*
- size_t rxDataSizeAll
  *Total bytes to be received.*
- dma_handle_t * txDmaHandle
  *The DMA TX channel used.*
- dma_handle_t * rxDmaHandle
  *The DMA RX channel used.*
- volatile uint8_t txState
  *TX transfer state.*
- volatile uint8_t rxState
  *RX transfer state.*

#### 23.8.9.2.1.1 Field Documentation

##### 23.8.9.2.1.1.1 flexio_uart_dma_transfer_callback_t flexio_uart_dma_handle_t::callback

##### 23.8.9.2.1.1.2 void* flexio_uart_dma_handle_t::userData

##### 23.8.9.2.1.1.3 size_t flexio_uart_dma_handle_t::txDataSizeAll

##### 23.8.9.2.1.1.4 size_t flexio_uart_dma_handle_t::rxDataSizeAll

##### 23.8.9.2.1.1.5 dma_handle_t* flexio_uart_dma_handle_t::txDmaHandle

##### 23.8.9.2.1.1.6 dma_handle_t* flexio_uart_dma_handle_t::rxDmaHandle

##### 23.8.9.2.1.1.7 volatile uint8_t flexio_uart_dma_handle_t::txState

### 23.8.9.3 Macro Definition Documentation

#### 23.8.9.3.1 #define FSL_FLEXIO_UART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))

### 23.8.9.4 Typedef Documentation

#### 23.8.9.4.1 typedef void(* flexio_uart_dma_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_dma_handle_t *handle, status_t status, void *userData)

### 23.8.9.5 Function Documentation

#### 23.8.9.5.1 status_t FLEXIO_UART_TransferCreateHandleDMA ( FLEXIO_UART_Type * *base,* flexio_uart_dma_handle_t * *handle,* flexio_uart_dma_transfer_callback_t *callback,* void * *userData,* dma_handle_t * *txDmaHandle,* dma_handle_t * *rxDmaHandle* )

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure. |
| *handle* | Pointer to flexio_uart_dma_handle_t structure. |
| *callback* | FlexIO UART callback, NULL means no callback. |
| *userData* | User callback function data. |
| *txDmaHandle* | User requested DMA handle for TX DMA transfer. |
| *rxDmaHandle* | User requested DMA handle for RX DMA transfer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully create the handle. |
| *kStatus_OutOfRange* | The FlexIO UART DMA type/handle table out of range. |

#### 23.8.9.5.2 status_t FLEXIO_UART_TransferSendDMA ( FLEXIO_UART_Type * *base*, flexio_uart_dma_handle_t * *handle*, flexio_uart_transfer_t * *xfer* )

This function send data using DMA. This is non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure |
| *handle* | Pointer to flexio_uart_dma_handle_t structure |
| *xfer* | FLEXIO_UART DMA transfer structure, see flexio_uart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_FLEXIO_UART-_TxBusy* | Previous transfer on going. |

#### 23.8.9.5.3 status_t FLEXIO_UART_TransferReceiveDMA ( FLEXIO_UART_Type * *base*, flexio_uart_dma_handle_t * *handle*, flexio_uart_transfer_t * *xfer* )

This function receives data using DMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure |
| *handle* | Pointer to flexio_uart_dma_handle_t structure |
| *xfer* | FLEXIO_UART DMA transfer structure, see flexio_uart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_FLEXIO_UART-_RxBusy* | Previous transfer on going. |

#### 23.8.9.5.4 void FLEXIO_UART_TransferAbortSendDMA ( FLEXIO_UART_Type * *base*, flexio_uart_dma_handle_t * *handle* )

This function aborts the sent data which using DMA.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure |
| *handle* | Pointer to flexio_uart_dma_handle_t structure |

#### 23.8.9.5.5 void FLEXIO_UART_TransferAbortReceiveDMA ( FLEXIO_UART_Type * *base*, flexio_uart_dma_handle_t * *handle* )

This function aborts the receive data which using DMA.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure |
| *handle* | Pointer to flexio_uart_dma_handle_t structure |

#### 23.8.9.5.6 status_t FLEXIO_UART_TransferGetSendCountDMA ( FLEXIO_UART_Type * *base*, flexio_uart_dma_handle_t * *handle*, size_t * *count* )

This function gets the number of bytes sent out.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure |
| *handle* | Pointer to flexio_uart_dma_handle_t structure |
| *count* | Number of bytes sent so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | transfer has finished or no transfer in progress. |
| *kStatus_Success* | Successfully return the count. |

#### 23.8.9.5.7 status_t FLEXIO_UART_TransferGetReceiveCountDMA ( FLEXIO_UART_Type * *base*, flexio_uart_dma_handle_t * *handle*, size_t * *count* )

This function gets the number of bytes received.

Parameters

| | |
|---|---|
| *base* | Pointer to FLEXIO_UART_Type structure |
| *handle* | Pointer to flexio_uart_dma_handle_t structure |
| *count* | Number of bytes received so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | transfer has finished or no transfer in progress. |
| *kStatus_Success* | Successfully return the count. |

# Chapter 24
# FLEXSPI: Flexible Serial Peripheral Interface Driver

## 24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flexible Serial Peripheral Interface (FLEXSPI) module of MCUXpresso SDK/i.MX devices.

FLEXSPI driver includes functional APIs and interrupt/EDMA non-blocking transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FLEXSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FLEXSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FLEXSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the flexspi_handle_t/flexspi_edma_handle_t as the second parameter. Initialize the handle for interrupt non-blocking transfer by calling the FLEXSPI_TransferCreateHandle API. Initialize the handle for interrupt non-blocking transfer by calling the FLEXSPI_TransferCreateHandleEDMA API.

Transactional APIs support asynchronous transfer. This means that the functions FLEXSPI_TransferNonBlocking() and FLEXSPI_TransferEDMA() set up data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_FLEXSPI_Idle status.

## Modules

- FLEXSPI eDMA Driver

## Data Structures

- struct flexspi_config_t
  *FLEXSPI configuration structure. More...*
- struct flexspi_device_config_t
  *External device configuration items. More...*
- struct flexspi_transfer_t
  *Transfer structure for FLEXSPI. More...*
- struct flexspi_handle_t
  *Transfer handle structure for FLEXSPI. More...*

## Macros

- #define FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
  *Formula to form FLEXSPI instructions in LUT table.*

## Typedefs

- typedef void(∗ flexspi_transfer_callback_t )(FLEXSPI_Type ∗base, flexspi_handle_t ∗handle, status_t status, void ∗userData)
  *FLEXSPI transfer callback function.*

## Enumerations

- enum _flexspi_status {
  kStatus_FLEXSPI_Busy = MAKE_STATUS(kStatusGroup_FLEXSPI, 0),
  kStatus_FLEXSPI_SequenceExecutionTimeout = MAKE_STATUS(kStatusGroup_FLEXSPI, 1),
  kStatus_FLEXSPI_IpCommandSequenceError = MAKE_STATUS(kStatusGroup_FLEXSPI, 2),
  kStatus_FLEXSPI_IpCommandGrantTimeout = MAKE_STATUS(kStatusGroup_FLEXSPI, 3) }
  *Status structure of FLEXSPI.*
- enum _flexspi_command {
  kFLEXSPI_Command_STOP = 0x00U,
  kFLEXSPI_Command_SDR = 0x01U,
  kFLEXSPI_Command_RADDR_SDR = 0x02U,
  kFLEXSPI_Command_CADDR_SDR = 0x03U,
  kFLEXSPI_Command_MODE1_SDR = 0x04U,
  kFLEXSPI_Command_MODE2_SDR = 0x05U,
  kFLEXSPI_Command_MODE4_SDR = 0x06U,
  kFLEXSPI_Command_MODE8_SDR = 0x07U,
  kFLEXSPI_Command_WRITE_SDR = 0x08U,
  kFLEXSPI_Command_READ_SDR = 0x09U,
  kFLEXSPI_Command_LEARN_SDR = 0x0AU,
  kFLEXSPI_Command_DATSZ_SDR = 0x0BU,
  kFLEXSPI_Command_DUMMY_SDR = 0x0CU,
  kFLEXSPI_Command_DUMMY_RWDS_SDR = 0x0DU,
  kFLEXSPI_Command_DDR = 0x21U,
  kFLEXSPI_Command_RADDR_DDR = 0x22U,
  kFLEXSPI_Command_CADDR_DDR = 0x23U,
  kFLEXSPI_Command_MODE1_DDR = 0x24U,
  kFLEXSPI_Command_MODE2_DDR = 0x25U,
  kFLEXSPI_Command_MODE4_DDR = 0x26U,
  kFLEXSPI_Command_MODE8_DDR = 0x27U,
  kFLEXSPI_Command_WRITE_DDR = 0x28U,
  kFLEXSPI_Command_READ_DDR = 0x29U,
  kFLEXSPI_Command_LEARN_DDR = 0x2AU,
  kFLEXSPI_Command_DATSZ_DDR = 0x2BU,
  kFLEXSPI_Command_DUMMY_DDR = 0x2CU,
  kFLEXSPI_Command_DUMMY_RWDS_DDR = 0x2DU,
  kFLEXSPI_Command_JUMP_ON_CS = 0x1FU }
  *CMD definition of FLEXSPI, use to form LUT instruction.*
- enum flexspi_pad_t {

kFLEXSPI_1PAD = 0x00U,
kFLEXSPI_2PAD = 0x01U,
kFLEXSPI_4PAD = 0x02U,
kFLEXSPI_8PAD = 0x03U }
*pad definition of FLEXSPI, use to form LUT instruction.*

- enum flexspi_flags_t {
kFLEXSPI_SequenceExecutionTimeoutFlag = FLEXSPI_INTEN_SEQTIMEOUTEN_MASK,
kFLEXSPI_AhbBusTimeoutFlag = FLEXSPI_INTEN_AHBBUSTIMEOUTEN_MASK,
kFLEXSPI_SckStoppedBecauseTxEmptyFlag,
kFLEXSPI_SckStoppedBecauseRxFullFlag,
kFLEXSPI_IpTxFifoWatermarkEmpltyFlag = FLEXSPI_INTEN_IPTXWEEN_MASK,
kFLEXSPI_IpRxFifoWatermarkAvailableFlag = FLEXSPI_INTEN_IPRXWAEN_MASK,
kFLEXSPI_AhbCommandSequenceErrorFlag,
kFLEXSPI_IpCommandSequenceErrorFlag = FLEXSPI_INTEN_IPCMDERREN_MASK,
kFLEXSPI_AhbCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandExcutionDoneFlag,
kFLEXSPI_AllInterruptFlags = 0xFFFU }
*FLEXSPI interrupt status flags.*
- enum flexspi_read_sample_clock_t {
kFLEXSPI_ReadSampleClkLoopbackInternally = 0x0U,
kFLEXSPI_ReadSampleClkLoopbackFromDqsPad = 0x1U,
kFLEXSPI_ReadSampleClkLoopbackFromSckPad = 0x2U,
kFLEXSPI_ReadSampleClkExternalInputFromDqsPad = 0x3U }
*FLEXSPI sample clock source selection for Flash Reading.*
- enum flexspi_cs_interval_cycle_unit_t {
kFLEXSPI_CsIntervalUnit1SckCycle = 0x0U,
kFLEXSPI_CsIntervalUnit256SckCycle = 0x1U }
*FLEXSPI interval unit for flash device select.*
- enum flexspi_ahb_write_wait_unit_t {
kFLEXSPI_AhbWriteWaitUnit2AhbCycle = 0x0U,
kFLEXSPI_AhbWriteWaitUnit8AhbCycle = 0x1U,
kFLEXSPI_AhbWriteWaitUnit32AhbCycle = 0x2U,
kFLEXSPI_AhbWriteWaitUnit128AhbCycle = 0x3U,
kFLEXSPI_AhbWriteWaitUnit512AhbCycle = 0x4U,
kFLEXSPI_AhbWriteWaitUnit2048AhbCycle = 0x5U,
kFLEXSPI_AhbWriteWaitUnit8192AhbCycle = 0x6U,
kFLEXSPI_AhbWriteWaitUnit32768AhbCycle = 0x7U }
*FLEXSPI AHB wait interval unit for writting.*
- enum flexspi_ip_error_code_t {

kFLEXSPI_IpCmdErrorNoError = 0x0U,
kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd = 0x2U,
kFLEXSPI_IpCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_IpCmdErrorInvalidAddress = 0x6U,
kFLEXSPI_IpCmdErrorSequenceExecutionTimeout = 0xEU,
kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss = 0xFU }
*Error Code when IP command Error detected.*

- enum flexspi_ahb_error_code_t {
kFLEXSPI_AhbCmdErrorNoError = 0x0U,
kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd = 0x2U,
kFLEXSPI_AhbCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_AhbCmdSequenceExecutionTimeout = 0x6U }
*Error Code when AHB command Error detected.*
- enum flexspi_port_t {
kFLEXSPI_PortA1 = 0x0U,
kFLEXSPI_PortA2,
kFLEXSPI_PortB1,
kFLEXSPI_PortB2 }
*FLEXSPI operation port select.*
- enum flexspi_arb_command_source_t
*Trigger source of current command sequence granted by arbitrator.*
- enum flexspi_command_type_t {
kFLEXSPI_Command,
kFLEXSPI_Config }

## Driver version

- #define FSL_FLEXSPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
*FLEXSPI driver version 2.1.1.*

## Initialization and deinitialization

- void FLEXSPI_Init (FLEXSPI_Type *base, const flexspi_config_t *config)
*Initializes the FLEXSPI module and internal state.*
- void FLEXSPI_GetDefaultConfig (flexspi_config_t *config)
*Gets default settings for FLEXSPI.*
- void FLEXSPI_Deinit (FLEXSPI_Type *base)
*Deinitializes the FLEXSPI module.*
- void FLEXSPI_SetFlashConfig (FLEXSPI_Type *base, flexspi_device_config_t *config, flexspi_port_t port)
*Configures the connected device parameter.*
- static void FLEXSPI_SoftwareReset (FLEXSPI_Type *base)
*Software reset for the FLEXSPI logic.*
- static void FLEXSPI_Enable (FLEXSPI_Type *base, bool enable)

*Enables or disables the FLEXSPI module.*

## Interrupts

- static void FLEXSPI_EnableInterrupts (FLEXSPI_Type *base, uint32_t mask)
  *Enables the FLEXSPI interrupts.*
- static void FLEXSPI_DisableInterrupts (FLEXSPI_Type *base, uint32_t mask)
  *Disable the FLEXSPI interrupts.*

## DMA control

- static void FLEXSPI_EnableTxDMA (FLEXSPI_Type *base, bool enable)
  *Enables or disables FLEXSPI IP Tx FIFO DMA requests.*
- static void FLEXSPI_EnableRxDMA (FLEXSPI_Type *base, bool enable)
  *Enables or disables FLEXSPI IP Rx FIFO DMA requests.*
- static uint32_t FLEXSPI_GetTxFifoAddress (FLEXSPI_Type *base)
  *Gets FLEXSPI IP tx fifo address for DMA transfer.*
- static uint32_t FLEXSPI_GetRxFifoAddress (FLEXSPI_Type *base)
  *Gets FLEXSPI IP rx fifo address for DMA transfer.*

## FIFO control

- static void FLEXSPI_ResetFifos (FLEXSPI_Type *base, bool txFifo, bool rxFifo)
  *Clears the FLEXSPI IP FIFO logic.*
- static void FLEXSPI_GetFifoCounts (FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)
  *Gets the valid data entries in the FLEXSPI FIFOs.*

## Status

- static uint32_t FLEXSPI_GetInterruptStatusFlags (FLEXSPI_Type *base)
  *Get the FLEXSPI interrupt status flags.*
- static void FLEXSPI_ClearInterruptStatusFlags (FLEXSPI_Type *base, uint32_t mask)
  *Get the FLEXSPI interrupt status flags.*
- static flexspi_arb_command_source_t FLEXSPI_GetArbitratorCommandSource (FLEXSPI_Type *base)
  *Gets the trigger source of current command sequence granted by arbitrator.*
- static flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode (FLEXSPI_Type *base, uint8_t *index)
  *Gets the error code when IP command error detected.*
- static flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode (FLEXSPI_Type *base, uint8_t *index)
  *Gets the error code when AHB command error detected.*
- static bool FLEXSPI_GetBusIdleStatus (FLEXSPI_Type *base)
  *Returns whether the bus is idle.*

## Bus Operations

- void FLEXSPI_UpdateRxSampleClock (FLEXSPI_Type *base, flexspi_read_sample_clock_t clockSource)
  *Update read sample clock source.*

- static void FLEXSPI_EnableIPParallelMode (FLEXSPI_Type *base, bool enable)
  *Enables/disables the FLEXSPI IP command parallel mode.*
- static void FLEXSPI_EnableAHBParallelMode (FLEXSPI_Type *base, bool enable)
  *Enables/disables the FLEXSPI AHB command parallel mode.*
- void FLEXSPI_UpdateLUT (FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t count)
  *Updates the LUT table.*
- static void FLEXSPI_WriteData (FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)
  *Writes data into FIFO.*
- static uint32_t FLEXSPI_ReadData (FLEXSPI_Type *base, uint8_t fifoIndex)
  *Receives data from data FIFO.*
- status_t FLEXSPI_WriteBlocking (FLEXSPI_Type *base, uint32_t *buffer, size_t size)
  *Sends a buffer of data bytes using blocking method.*
- status_t FLEXSPI_ReadBlocking (FLEXSPI_Type *base, uint32_t *buffer, size_t size)
  *Receives a buffer of data bytes using a blocking method.*
- status_t FLEXSPI_TransferBlocking (FLEXSPI_Type *base, flexspi_transfer_t *xfer)
  *Execute command to transfer a buffer data bytes using a blocking method.*

## Transactional

- void FLEXSPI_TransferCreateHandle (FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_callback_t callback, void *userData)
  *Initializes the FLEXSPI handle which is used in transactional functions.*
- status_t FLEXSPI_TransferNonBlocking (FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_t *xfer)
  *Performs a interrupt non-blocking transfer on the FLEXSPI bus.*
- status_t FLEXSPI_TransferGetCount (FLEXSPI_Type *base, flexspi_handle_t *handle, size_t *count)
  *Gets the master transfer status during a interrupt non-blocking transfer.*
- void FLEXSPI_TransferAbort (FLEXSPI_Type *base, flexspi_handle_t *handle)
  *Aborts an interrupt non-blocking transfer early.*
- void FLEXSPI_TransferHandleIRQ (FLEXSPI_Type *base, flexspi_handle_t *handle)
  *Master interrupt handler.*

# 24.2 Data Structure Documentation

## 24.2.1 struct flexspi_config_t

### Data Fields

- flexspi_read_sample_clock_t rxSampleClock
  *Sample Clock source selection for Flash Reading.*
- bool enableSckFreeRunning
  *Enable/disable SCK output free-running.*
- bool enableCombination
  `Enable/disable combining PORT A and B Data Pins`
  *(SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.*
- bool enableDoze
  *Enable/disable doze mode support.*

- bool enableHalfSpeedAccess
  `Enable/disable divide by 2 of the clock for half`
  *speed commands.*
- bool enableSckBDiffOpt
  `Enable/disable SCKB pad use as SCKA differential clock`
  *output, when enable, Port B flash access is not available.*
- bool enableSameConfigForAll
  `Enable/disable same configuration for all connected devices`
  *when enabled, same configuration in FLASHA1CRx is applied to all.*
- uint16_t seqTimeoutCycle
  `Timeout wait cycle for command sequence execution,`
  *timeout after ahbGrantTimeoutCyle∗1024 serial root clock cycles.*
- uint8_t ipGrantTimeoutCycle
  `Timeout wait cycle for IP command grant, timeout after`
  *ipGrantTimeoutCycle∗1024 AHB clock cycles.*
- uint8_t txWatermark
  *FLEXSPI IP transmit watermark value.*
- uint8_t rxWatermark
  *FLEXSPI receive watermark value.*
- bool enableAHBWriteIpTxFifo
  *Enable AHB bus write access to IP TX FIFO.*
- bool enableAHBWriteIpRxFifo
  *Enable AHB bus write access to IP RX FIFO.*
- uint8_t ahbGrantTimeoutCycle
  `Timeout wait cycle for AHB command grant,`
  *timeout after ahbGrantTimeoutCyle∗1024 AHB clock cycles.*
- uint16_t ahbBusTimeoutCycle
  `Timeout wait cycle for AHB read/write access,`
  *timeout after ahbBusTimeoutCycle∗1024 AHB clock cycles.*
- uint8_t resumeWaitCycle
  `Wait cycle for idle state before suspended command sequence`
  *resume, timeout after ahbBusTimeoutCycle AHB clock cycles.*
- flexspi_ahbBuffer_config_t buffer [FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT]
  *AHB buffer size.*
- bool enableClearAHBBufferOpt
  `Enable/disable automatically clean AHB RX Buffer and TX Buffer`
  *when FLEXSPI returns STOP mode ACK.*
- bool enableReadAddressOpt
  *Enable/disable remove AHB read burst start address alignment limitation.*
- bool enableAHBPrefetch
  `Enable/disable AHB read prefetch feature, when enabled, FLEXSPI`
  *will fetch more data than current AHB burst.*
- bool enableAHBBufferable
  `Enable/disable AHB bufferable write access support, when enabled,`
  *FLEXSPI return before waiting for command excution finished.*
- bool enableAHBCachable
  *Enable AHB bus cachable read access support.*

#### 24.2.1.0.7.1 Field Documentation

##### 24.2.1.0.7.1.1 flexspi_read_sample_clock_t flexspi_config_t::rxSampleClock

##### 24.2.1.0.7.1.2 bool flexspi_config_t::enableSckFreeRunning

##### 24.2.1.0.7.1.3 bool flexspi_config_t::enableCombination

##### 24.2.1.0.7.1.4 bool flexspi_config_t::enableDoze

##### 24.2.1.0.7.1.5 bool flexspi_config_t::enableHalfSpeedAccess

##### 24.2.1.0.7.1.6 bool flexspi_config_t::enableSckBDiffOpt

##### 24.2.1.0.7.1.7 bool flexspi_config_t::enableSameConfigForAll

##### 24.2.1.0.7.1.8 uint16_t flexspi_config_t::seqTimeoutCycle

##### 24.2.1.0.7.1.9 uint8_t flexspi_config_t::ipGrantTimeoutCycle

##### 24.2.1.0.7.1.10 uint8_t flexspi_config_t::txWatermark

##### 24.2.1.0.7.1.11 uint8_t flexspi_config_t::rxWatermark

##### 24.2.1.0.7.1.12 bool flexspi_config_t::enableAHBWriteIpTxFifo

##### 24.2.1.0.7.1.13 bool flexspi_config_t::enableAHBWriteIpRxFifo

##### 24.2.1.0.7.1.14 uint8_t flexspi_config_t::ahbGrantTimeoutCycle

##### 24.2.1.0.7.1.15 uint16_t flexspi_config_t::ahbBusTimeoutCycle

##### 24.2.1.0.7.1.16 uint8_t flexspi_config_t::resumeWaitCycle

##### 24.2.1.0.7.1.17 flexspi_ahbBuffer_config_t flexspi_config_t::buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT]

##### 24.2.1.0.7.1.18 bool flexspi_config_t::enableClearAHBBufferOpt

##### 24.2.1.0.7.1.19 bool flexspi_config_t::enableReadAddressOpt

when eanble, there is no AHB read burst start address alignment limitation.

#### 24.2.1.0.7.1.20 bool flexspi_config_t::enableAHBPrefetch

#### 24.2.1.0.7.1.21 bool flexspi_config_t::enableAHBBufferable

#### 24.2.1.0.7.1.22 bool flexspi_config_t::enableAHBCachable

### 24.2.2 struct flexspi_device_config_t

## Data Fields

- uint32_t flexspiRootClk
  *FLEXSPI serial root clock.*
- bool isSck2Enabled
  *FLEXSPI use SCK2.*
- uint32_t flashSize
  *Flash size in KByte.*
- flexspi_cs_interval_cycle_unit_t CSIntervalUnit
  *CS interval unit, 1 or 256 cycle.*
- uint16_t CSInterval
  `CS line assert interval, mutiply CS interval unit to`
  *get the CS line assert interval cycles.*
- uint8_t CSHoldTime
  *CS line hold time.*
- uint8_t CSSetupTime
  *CS line setup time.*
- uint8_t dataValidTime
  *Data valid time for external device.*
- uint8_t columnspace
  *Column space size.*
- bool enableWordAddress
  *If enable word address.*
- uint8_t AWRSeqIndex
  *Sequence ID for AHB write command.*
- uint8_t AWRSeqNumber
  *Sequence number for AHB write command.*
- uint8_t ARDSeqIndex
  *Sequence ID for AHB read command.*
- uint8_t ARDSeqNumber
  *Sequence number for AHB read command.*
- flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit
  *AHB write wait unit.*
- uint16_t AHBWriteWaitInterval
  `AHB write wait interval, mutiply AHB write interval`
  *unit to get the AHB write wait cycles.*
- bool enableWriteMask
  `Enable/Disable FLEXSPI drive DQS pin as write mask`
  *when writing to external device.*

#### 24.2.2.0.7.2 Field Documentation

##### 24.2.2.0.7.2.1 uint32_t flexspi_device_config_t::flexspiRootClk

##### 24.2.2.0.7.2.2 bool flexspi_device_config_t::isSck2Enabled

##### 24.2.2.0.7.2.3 uint32_t flexspi_device_config_t::flashSize

##### 24.2.2.0.7.2.4 flexspi_cs_interval_cycle_unit_t flexspi_device_config_t::CSIntervalUnit

##### 24.2.2.0.7.2.5 uint16_t flexspi_device_config_t::CSInterval

##### 24.2.2.0.7.2.6 uint8_t flexspi_device_config_t::CSHoldTime

##### 24.2.2.0.7.2.7 uint8_t flexspi_device_config_t::CSSetupTime

##### 24.2.2.0.7.2.8 uint8_t flexspi_device_config_t::dataValidTime

##### 24.2.2.0.7.2.9 uint8_t flexspi_device_config_t::columnspace

##### 24.2.2.0.7.2.10 bool flexspi_device_config_t::enableWordAddress

##### 24.2.2.0.7.2.11 uint8_t flexspi_device_config_t::AWRSeqIndex

##### 24.2.2.0.7.2.12 uint8_t flexspi_device_config_t::AWRSeqNumber

##### 24.2.2.0.7.2.13 uint8_t flexspi_device_config_t::ARDSeqIndex

##### 24.2.2.0.7.2.14 uint8_t flexspi_device_config_t::ARDSeqNumber

##### 24.2.2.0.7.2.15 flexspi_ahb_write_wait_unit_t flexspi_device_config_t::AHBWriteWaitUnit

##### 24.2.2.0.7.2.16 uint16_t flexspi_device_config_t::AHBWriteWaitInterval

##### 24.2.2.0.7.2.17 bool flexspi_device_config_t::enableWriteMask

### 24.2.3 struct flexspi_transfer_t

## Data Fields

- uint32_t deviceAddress
  *Operation device address.*
- flexspi_port_t port
  *Operation port.*
- flexspi_command_type_t cmdType
  *Execution command type.*
- uint8_t seqIndex
  *Sequence ID for command.*
- uint8_t SeqNumber
  *Sequence number for command.*

- uint32_t * data
  *Data buffer.*
- size_t dataSize
  *Data size in bytes.*

#### 24.2.3.0.7.3 Field Documentation

##### 24.2.3.0.7.3.1 uint32_t flexspi_transfer_t::deviceAddress

##### 24.2.3.0.7.3.2 flexspi_port_t flexspi_transfer_t::port

##### 24.2.3.0.7.3.3 flexspi_command_type_t flexspi_transfer_t::cmdType

##### 24.2.3.0.7.3.4 uint8_t flexspi_transfer_t::seqIndex

##### 24.2.3.0.7.3.5 uint8_t flexspi_transfer_t::SeqNumber

##### 24.2.3.0.7.3.6 uint32_t* flexspi_transfer_t::data

##### 24.2.3.0.7.3.7 size_t flexspi_transfer_t::dataSize

### 24.2.4 struct _flexspi_handle

## Data Fields

- uint32_t state
  *Internal state for FLEXSPI transfer.*
- uint32_t * data
  *Data buffer.*
- size_t dataSize
  *Remaining Data size in bytes.*
- size_t transferTotalSize
  *Total Data size in bytes.*
- flexspi_transfer_callback_t completionCallback
  *Callback for users while transfer finish or error occurred.*
- void * userData
  *FLEXSPI callback function parameter.*

#### 24.2.4.0.7.4 Field Documentation

##### 24.2.4.0.7.4.1 uint32_t* flexspi_handle_t::data

##### 24.2.4.0.7.4.2 size_t flexspi_handle_t::dataSize

##### 24.2.4.0.7.4.3 size_t flexspi_handle_t::transferTotalSize

##### 24.2.4.0.7.4.4 void* flexspi_handle_t::userData

## 24.3 Macro Definition Documentation

### 24.3.1 #define FSL_FLEXSPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

### 24.3.2 #define FLEXSPI_LUT_SEQ( *cmd0, pad0, op0, cmd1, pad1, op1* )

**Value:**

```
(FLEXSPI_LUT_OPERAND0(op0) | FLEXSPI_LUT_NUM_PADS0(pad0) | FLEXSPI_LUT_OPCODE0(cmd0) | FLEXSPI_LUT_OPERAND1
      (op1) | \
     FLEXSPI_LUT_NUM_PADS1(pad1) | FLEXSPI_LUT_OPCODE1(cmd1))
```

## 24.4 Typedef Documentation

### 24.4.1 typedef void(∗ flexspi_transfer_callback_t)(FLEXSPI_Type ∗base, flexspi_handle_t ∗handle, status_t status, void ∗userData)

## 24.5 Enumeration Type Documentation

### 24.5.1 enum _flexspi_status

Enumerator

***kStatus_FLEXSPI_Busy*** FLEXSPI is busy.
***kStatus_FLEXSPI_SequenceExecutionTimeout*** Sequence execution timeout error occurred during FLEXSPI transfer.
***kStatus_FLEXSPI_IpCommandSequenceError*** IP command Sequence execution timeout error occurred during FLEXSPI transfer.
***kStatus_FLEXSPI_IpCommandGrantTimeout*** IP command grant timeout error occurred during FLEXSPI transfer.

### 24.5.2 enum _flexspi_command

Enumerator

***kFLEXSPI_Command_STOP*** Stop execution, deassert CS.
***kFLEXSPI_Command_SDR*** Transmit Command code to Flash, using SDR mode.
***kFLEXSPI_Command_RADDR_SDR*** Transmit Row Address to Flash, using SDR mode.
***kFLEXSPI_Command_CADDR_SDR*** Transmit Column Address to Flash, using SDR mode.
***kFLEXSPI_Command_MODE1_SDR*** Transmit 1-bit Mode bits to Flash, using SDR mode.
***kFLEXSPI_Command_MODE2_SDR*** Transmit 2-bit Mode bits to Flash, using SDR mode.
***kFLEXSPI_Command_MODE4_SDR*** Transmit 4-bit Mode bits to Flash, using SDR mode.
***kFLEXSPI_Command_MODE8_SDR*** Transmit 8-bit Mode bits to Flash, using SDR mode.
***kFLEXSPI_Command_WRITE_SDR*** Transmit Programming Data to Flash, using SDR mode.
***kFLEXSPI_Command_READ_SDR*** Receive Read Data from Flash, using SDR mode.

*kFLEXSPI_Command_LEARN_SDR* Receive Read Data or Preamble bit from Flash, SDR mode.

*kFLEXSPI_Command_DATSZ_SDR* Transmit Read/Program Data size (byte) to Flash, SDR mode.

*kFLEXSPI_Command_DUMMY_SDR* Leave data lines undriven by FlexSPI controller.

*kFLEXSPI_Command_DUMMY_RWDS_SDR* Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

*kFLEXSPI_Command_DDR* Transmit Command code to Flash, using DDR mode.

*kFLEXSPI_Command_RADDR_DDR* Transmit Row Address to Flash, using DDR mode.

*kFLEXSPI_Command_CADDR_DDR* Transmit Column Address to Flash, using DDR mode.

*kFLEXSPI_Command_MODE1_DDR* Transmit 1-bit Mode bits to Flash, using DDR mode.

*kFLEXSPI_Command_MODE2_DDR* Transmit 2-bit Mode bits to Flash, using DDR mode.

*kFLEXSPI_Command_MODE4_DDR* Transmit 4-bit Mode bits to Flash, using DDR mode.

*kFLEXSPI_Command_MODE8_DDR* Transmit 8-bit Mode bits to Flash, using DDR mode.

*kFLEXSPI_Command_WRITE_DDR* Transmit Programming Data to Flash, using DDR mode.

*kFLEXSPI_Command_READ_DDR* Receive Read Data from Flash, using DDR mode.

*kFLEXSPI_Command_LEARN_DDR* Receive Read Data or Preamble bit from Flash, DDR mode.

*kFLEXSPI_Command_DATSZ_DDR* Transmit Read/Program Data size (byte) to Flash, DDR mode.

*kFLEXSPI_Command_DUMMY_DDR* Leave data lines undriven by FlexSPI controller.

*kFLEXSPI_Command_DUMMY_RWDS_DDR* Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

*kFLEXSPI_Command_JUMP_ON_CS* Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence.

### 24.5.3 enum flexspi_pad_t

Enumerator

*kFLEXSPI_1PAD* Transmit command/address and transmit/receive data only through DATA0/DATA1.

*kFLEXSPI_2PAD* Transmit command/address and transmit/receive data only through DATA[1:0].

*kFLEXSPI_4PAD* Transmit command/address and transmit/receive data only through DATA[3:0].

*kFLEXSPI_8PAD* Transmit command/address and transmit/receive data only through DATA[7:0].

### 24.5.4 enum flexspi_flags_t

Enumerator

*kFLEXSPI_SequenceExecutionTimeoutFlag* Sequence execution timeout.

***kFLEXSPI_AhbBusTimeoutFlag*** AHB Bus timeout.
***kFLEXSPI_SckStoppedBecauseTxEmptyFlag*** SCK is stopped during command sequence because Async TX FIFO empty.
***kFLEXSPI_SckStoppedBecauseRxFullFlag*** SCK is stopped during command sequence because Async RX FIFO full.
***kFLEXSPI_IpTxFifoWatermarkEmpltyFlag*** IP TX FIFO WaterMark empty.
***kFLEXSPI_IpRxFifoWatermarkAvailableFlag*** IP RX FIFO WaterMark available.
***kFLEXSPI_AhbCommandSequenceErrorFlag*** AHB triggered Command Sequences Error.
***kFLEXSPI_IpCommandSequenceErrorFlag*** IP triggered Command Sequences Error.
***kFLEXSPI_AhbCommandGrantTimeoutFlag*** AHB triggered Command Sequences Grant Timeout.
***kFLEXSPI_IpCommandGrantTimeoutFlag*** IP triggered Command Sequences Grant Timeout.
***kFLEXSPI_IpCommandExcutionDoneFlag*** IP triggered Command Sequences Execution finished.

***kFLEXSPI_AllInterruptFlags*** All flags.

### 24.5.5 enum flexspi_read_sample_clock_t

Enumerator

***kFLEXSPI_ReadSampleClkLoopbackInternally*** Dummy Read strobe generated by FlexSPI Controller and loopback internally.
***kFLEXSPI_ReadSampleClkLoopbackFromDqsPad*** Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.
***kFLEXSPI_ReadSampleClkLoopbackFromSckPad*** SCK output clock and loopback from SCK pad.
***kFLEXSPI_ReadSampleClkExternalInputFromDqsPad*** Flash provided Read strobe and input from DQS pad.

### 24.5.6 enum flexspi_cs_interval_cycle_unit_t

Enumerator

***kFLEXSPI_CsIntervalUnit1SckCycle*** Chip selection interval: CSINTERVAL $*$ 1 serial clock cycle.
***kFLEXSPI_CsIntervalUnit256SckCycle*** Chip selection interval: CSINTERVAL $*$ 256 serial clock cycle.

### 24.5.7 enum flexspi_ahb_write_wait_unit_t

Enumerator

***kFLEXSPI_AhbWriteWaitUnit2AhbCycle*** AWRWAIT unit is 2 ahb clock cycle.

*kFLEXSPI_AhbWriteWaitUnit8AhbCycle* AWRWAIT unit is 8 ahb clock cycle.
*kFLEXSPI_AhbWriteWaitUnit32AhbCycle* AWRWAIT unit is 32 ahb clock cycle.
*kFLEXSPI_AhbWriteWaitUnit128AhbCycle* AWRWAIT unit is 128 ahb clock cycle.
*kFLEXSPI_AhbWriteWaitUnit512AhbCycle* AWRWAIT unit is 512 ahb clock cycle.
*kFLEXSPI_AhbWriteWaitUnit2048AhbCycle* AWRWAIT unit is 2048 ahb clock cycle.
*kFLEXSPI_AhbWriteWaitUnit8192AhbCycle* AWRWAIT unit is 8192 ahb clock cycle.
*kFLEXSPI_AhbWriteWaitUnit32768AhbCycle* AWRWAIT unit is 32768 ahb clock cycle.

### 24.5.8 enum flexspi_ip_error_code_t

Enumerator

*kFLEXSPI_IpCmdErrorNoError* No error.
*kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd* IP command with JMP_ON_CS instruction used.
*kFLEXSPI_IpCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.
*kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence* Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.
*kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence* Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.
*kFLEXSPI_IpCmdErrorInvalidAddress* Flash access start address exceed the whole flash address range (A1/A2/B1/B2).
*kFLEXSPI_IpCmdErrorSequenceExecutionTimeout* Sequence execution timeout.
*kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss* Flash boundary crossed.

### 24.5.9 enum flexspi_ahb_error_code_t

Enumerator

*kFLEXSPI_AhbCmdErrorNoError* No error.
*kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd* AHB Write command with JMP_ON_CS instruction used in the sequence.
*kFLEXSPI_AhbCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.
*kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence* Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.
*kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence* Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.
*kFLEXSPI_AhbCmdSequenceExecutionTimeout* Sequence execution timeout.

### 24.5.10 enum flexspi_port_t

Enumerator

*kFLEXSPI_PortA1* Access flash on A1 port.

***kFLEXSPI_PortA2*** Access flash on A2 port.
***kFLEXSPI_PortB1*** Access flash on B1 port.
***kFLEXSPI_PortB2*** Access flash on B2 port.

### 24.5.11 enum flexspi_arb_command_source_t

### 24.5.12 enum flexspi_command_type_t

Enumerator

***kFLEXSPI_Command*** FlexSPI operation: Only command, both TX and Rx buffer are ignored.
***kFLEXSPI_Config*** FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

## 24.6 Function Documentation

### 24.6.1 void FLEXSPI_Init ( FLEXSPI_Type * *base*, const flexspi_config_t * *config* )

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *config* | FLEXSPI configure structure. |

### 24.6.2 void FLEXSPI_GetDefaultConfig ( flexspi_config_t * *config* )

Parameters

| | |
|---|---|
| *config* | FLEXSPI configuration structure. |

### 24.6.3 void FLEXSPI_Deinit ( FLEXSPI_Type * *base* )

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |

### 24.6.4 void FLEXSPI_SetFlashConfig ( FLEXSPI_Type * *base*, flexspi_device_config_t * *config*, flexspi_port_t *port* )

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *config* | Flash configuration parameters. |
| *port* | FLEXSPI Operation port. |

### 24.6.5 static void FLEXSPI_SoftwareReset ( FLEXSPI_Type * *base* ) [inline], [static]

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |

### 24.6.6 static void FLEXSPI_Enable ( FLEXSPI_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *enable* | True means enable FLEXSPI, false means disable. |

### 24.6.7 static void FLEXSPI_EnableInterrupts ( FLEXSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *mask* | FLEXSPI interrupt source. |

### 24.6.8 static void FLEXSPI_DisableInterrupts ( FLEXSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *mask* | FLEXSPI interrupt source. |

### 24.6.9 static void FLEXSPI_EnableTxDMA ( FLEXSPI_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *enable* | Enable flag for transmit DMA request. Pass true for enable, false for disable. |

### 24.6.10 static void FLEXSPI_EnableRxDMA ( FLEXSPI_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *enable* | Enable flag for receive DMA request. Pass true for enable, false for disable. |

### 24.6.11 static uint32_t FLEXSPI_GetTxFifoAddress ( FLEXSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |

Return values

| | |
|---|---|
| *The* | tx fifo address. |

### 24.6.12 static uint32_t FLEXSPI_GetRxFifoAddress ( FLEXSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |

Return values

| | |
|---|---|
| *The* | rx fifo address. |

### 24.6.13 static void FLEXSPI_ResetFifos ( FLEXSPI_Type * *base,* bool *txFifo,* bool *rxFifo* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *txFifo* | Pass true to reset TX FIFO. |
| *rxFifo* | Pass true to reset RX FIFO. |

### 24.6.14 static void FLEXSPI_GetFifoCounts ( FLEXSPI_Type * *base,* size_t * *txCount,* size_t * *rxCount* ) [inline], [static]

Parameters

| | | |
|---|---|---|
| | *base* | FLEXSPI peripheral base address. |
| out | *txCount* | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | *rxCount* | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required. |

### 24.6.15 static uint32_t FLEXSPI_GetInterruptStatusFlags ( FLEXSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |

Return values

| | |
|---|---|
| *interrupt* | status flag, use status flag to AND flexspi_flags_t could get the related status. |

### 24.6.16 static void FLEXSPI_ClearInterruptStatusFlags ( FLEXSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *interrupt* | status flag. |

### 24.6.17 static flexspi_arb_command_source_t FLEXSPI_GetArbitratorCommandSource ( FLEXSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |

Return values

| | |
|---|---|
| *trigger* | source of current command sequence. |

### 24.6.18 static flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode ( FLEXSPI_Type * *base*, uint8_t * *index* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *index* | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

| | |
|---|---|
| *error* | code when IP command error detected. |

### 24.6.19 static flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode ( FLEXSPI_Type * *base*, uint8_t * *index* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *index* | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

| | |
|---|---|
| *error* | code when AHB command error detected. |

### 24.6.20 static bool FLEXSPI_GetBusIdleStatus ( FLEXSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |

Return values

| | |
|---|---|
| *true* | Bus is idle. |
| *false* | Bus is busy. |

### 24.6.21 void FLEXSPI_UpdateRxSampleClock ( FLEXSPI_Type * *base*, flexspi_read_sample_clock_t *clockSource* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *clockSource* | clockSource of type flexspi_read_sample_clock_t |

### 24.6.22 static void FLEXSPI_EnableIPParallelMode ( FLEXSPI_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *enable* | True means enable parallel mode, false means disable parallel mode. |

### 24.6.23 static void FLEXSPI_EnableAHBParallelMode ( FLEXSPI_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *enable* | True means enable parallel mode, false means disable parallel mode. |

### 24.6.24 void FLEXSPI_UpdateLUT ( FLEXSPI_Type * *base*, uint32_t *index*, const uint32_t * *cmd*, uint32_t *count* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *index* | From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4∗32-bit memory. |
| *cmd* | Command sequence array. |
| *count* | Number of sequences. |

### 24.6.25 static void FLEXSPI_WriteData ( FLEXSPI_Type ∗ *base,* uint32_t *data,* uint8_t *fifoIndex* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address |
| *data* | The data bytes to send |
| *fifoIndex* | Destination fifo index. |

### 24.6.26 static uint32_t FLEXSPI_ReadData ( FLEXSPI_Type ∗ *base,* uint8_t *fifoIndex* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address |
| *fifoIndex* | Source fifo index. |

Returns

The data in the FIFO.

### 24.6.27 status_t FLEXSPI_WriteBlocking ( FLEXSPI_Type ∗ *base,* uint32_t ∗ *buffer,* size_t *size* )

Note

This function blocks via polling until all bytes have been sent.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address |
| *buffer* | The data bytes to send |
| *size* | The number of data bytes to send |

Return values

| | |
|---|---|
| *kStatus_Success* | write success without error |
| *kStatus_FLEXSPI_SequenceExecutionTimeout* | sequence execution timeout |
| *kStatus_FLEXSPI_IpCommandSequenceError* | IP command sequencen error detected |
| *kStatus_FLEXSPI_IpCommandGrantTimeout* | IP command grant timeout detected |

### 24.6.28 status_t FLEXSPI_ReadBlocking ( FLEXSPI_Type * *base*, uint32_t * *buffer*, size_t *size* )

Note

This function blocks via polling until all bytes have been sent.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address |
| *buffer* | The data bytes to send |
| *size* | The number of data bytes to receive |

Return values

| | |
|---|---|
| *kStatus_Success* | read success without error |
| *kStatus_FLEXSPI_SequenceExecutionTimeout* | sequence execution timeout |

| | |
|---|---|
| *kStatus_FLEXSPI_IpCommandSequenceError* | IP command sequencen error detected |
| *kStatus_FLEXSPI_IpCommandGrantTimeout* | IP command grant timeout detected |

### 24.6.29 status_t FLEXSPI_TransferBlocking ( FLEXSPI_Type * *base*, flexspi_transfer_t * *xfer* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address |
| *xfer* | pointer to the transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | command transfer success without error |
| *kStatus_FLEXSPI_SequenceExecutionTimeout* | sequence execution timeout |
| *kStatus_FLEXSPI_IpCommandSequenceError* | IP command sequencen error detected |
| *kStatus_FLEXSPI_IpCommandGrantTimeout* | IP command grant timeout detected |

### 24.6.30 void FLEXSPI_TransferCreateHandle ( FLEXSPI_Type * *base*, flexspi_handle_t * *handle*, flexspi_transfer_callback_t *callback*, void * *userData* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | pointer to flexspi_handle_t structure to store the transfer state. |
| *callback* | pointer to user callback function. |
| *userData* | user parameter passed to the callback function. |

### 24.6.31 status_t FLEXSPI_TransferNonBlocking ( FLEXSPI_Type * *base*, flexspi_handle_t * *handle*, flexspi_transfer_t * *xfer* )

Note

Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXSPI_Busy, the transfer is finished. For FLEXSPI_Read, the dataSize should be multiple of rx watermark levle, or FLEXSPI could not read data properly.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | pointer to flexspi_handle_t structure which stores the transfer state. |
| *xfer* | pointer to flexspi_transfer_t structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start the data transmission. |
| *kStatus_FLEXSPI_Busy* | Previous transmission still not finished. |

### 24.6.32 status_t FLEXSPI_TransferGetCount ( FLEXSPI_Type * *base*, flexspi_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | pointer to flexspi_handle_t structure which stores the transfer state. |
| *count* | Number of bytes transferred so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *kStatus_InvalidArgument* | count is Invalid. |
| *kStatus_Success* | Successfully return the count. |

### 24.6.33 void FLEXSPI_TransferAbort ( FLEXSPI_Type * *base*, flexspi_handle_t * *handle* )

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | pointer to flexspi_handle_t structure which stores the transfer state |

### 24.6.34 void FLEXSPI_TransferHandleIRQ ( FLEXSPI_Type * *base*, flexspi_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | pointer to flexspi_handle_t structure. |

## 24.7 FLEXSPI eDMA Driver

### 24.7.1 Overview

#### Data Structures

- struct flexspi_edma_handle_t
  *FLEXSPI DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef void(∗ flexspi_edma_callback_t )(FLEXSPI_Type ∗base, flexspi_edma_handle_t ∗handle, status_t status, void ∗userData)
  *FLEXSPI eDMA transfer callback function for finish and error.*

#### Enumerations

- enum flexspi_edma_transfer_nsize_t {
  kFLEXPSI_EDMAnSize1Bytes = 0x1U,
  kFLEXPSI_EDMAnSize2Bytes = 0x2U,
  kFLEXPSI_EDMAnSize4Bytes = 0x4U,
  kFLEXPSI_EDMAnSize8Bytes = 0x8U,
  kFLEXPSI_EDMAnSize32Bytes = 0x20U }
  *eDMA transfer configuration*

#### Driver version

- #define FSL_FLEXSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *FLEXSPI EDMA driver version 2.0.2.*

#### FLEXSPI eDMA Transactional

- void FLEXSPI_TransferCreateHandleEDMA (FLEXSPI_Type ∗base, flexspi_edma_handle_t ∗handle, flexspi_edma_callback_t callback, void ∗userData, edma_handle_t ∗txDmaHandle, edma_handle_t ∗rxDmaHandle)
  *Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.*
- void FLEXSPI_TransferUpdateSizeEDMA (FLEXSPI_Type ∗base, flexspi_edma_handle_t ∗handle, flexspi_edma_transfer_nsize_t nsize)
  *Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).*
- status_t FLEXSPI_TransferEDMA (FLEXSPI_Type ∗base, flexspi_edma_handle_t ∗handle, flexspi_transfer_t ∗xfer)
  *Transfers FLEXSPI data using an eDMA non-blocking method.*

- void FLEXSPI_TransferAbortEDMA (FLEXSPI_Type *base, flexspi_edma_handle_t *handle)
  *Aborts the transfer data using eDMA.*
- status_t FLEXSPI_TransferGetCountEDMA (FLEXSPI_Type *base, flexspi_edma_handle_t *handle, size_t *count)
  *Gets the transferred counts of transfer.*

## 24.7.2 Data Structure Documentation

### 24.7.2.1 struct _flexspi_edma_handle

#### Data Fields

- edma_handle_t * txDmaHandle
  *eDMA handler for FLEXSPI Tx.*
- edma_handle_t * rxDmaHandle
  *eDMA handler for FLEXSPI Rx.*
- size_t transferSize
  *Bytes need to transfer.*
- flexspi_edma_transfer_nsize_t nsize
  *eDMA SSIZE/DSIZE in each transfer.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- uint8_t count
  *The transfer data count in a DMA request.*
- uint32_t state
  *Internal state for FLEXSPI eDMA transfer.*
- flexspi_edma_callback_t completionCallback
  *A callback function called after the eDMA transfer is finished.*
- void * userData
  *User callback parameter.*

#### 24.7.2.1.0.5 Field Documentation

##### 24.7.2.1.0.5.1 edma_handle_t* flexspi_edma_handle_t::txDmaHandle

##### 24.7.2.1.0.5.2 edma_handle_t* flexspi_edma_handle_t::rxDmaHandle

##### 24.7.2.1.0.5.3 size_t flexspi_edma_handle_t::transferSize

##### 24.7.2.1.0.5.4 flexspi_edma_transfer_nsize_t flexspi_edma_handle_t::nsize

##### 24.7.2.1.0.5.5 uint8_t flexspi_edma_handle_t::nbytes

##### 24.7.2.1.0.5.6 uint8_t flexspi_edma_handle_t::count

##### 24.7.2.1.0.5.7 uint32_t flexspi_edma_handle_t::state

##### 24.7.2.1.0.5.8 flexspi_edma_callback_t flexspi_edma_handle_t::completionCallback

### 24.7.3 Macro Definition Documentation

#### 24.7.3.1 #define FSL_FLEXSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

### 24.7.4 Enumeration Type Documentation

#### 24.7.4.1 enum flexspi_edma_transfer_nsize_t

Enumerator

| | |
|---|---|
| ***kFLEXPSI_EDMAnSize1Bytes*** | Source/Destination data transfer size is 1 byte every time. |
| ***kFLEXPSI_EDMAnSize2Bytes*** | Source/Destination data transfer size is 2 bytes every time. |
| ***kFLEXPSI_EDMAnSize4Bytes*** | Source/Destination data transfer size is 4 bytes every time. |
| ***kFLEXPSI_EDMAnSize8Bytes*** | Source/Destination data transfer size is 8 bytes every time. |
| ***kFLEXPSI_EDMAnSize32Bytes*** | Source/Destination data transfer size is 32 bytes every time. |

### 24.7.5 Function Documentation

#### 24.7.5.1 void FLEXSPI_TransferCreateHandleEDMA ( FLEXSPI_Type * *base,* flexspi_edma_handle_t * *handle,* flexspi_edma_callback_t *callback,* void * *userData,* edma_handle_t * *txDmaHandle,* edma_handle_t * *rxDmaHandle* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address |
| *handle* | Pointer to flexspi_edma_handle_t structure |
| *callback* | FLEXSPI callback, NULL means no callback. |
| *userData* | User callback function data. |
| *txDmaHandle* | User requested DMA handle for TX DMA transfer. |
| *rxDmaHandle* | User requested DMA handle for RX DMA transfer. |

#### 24.7.5.2 void FLEXSPI_TransferUpdateSizeEDMA ( FLEXSPI_Type * *base*, flexspi_edma_handle_t * *handle*, flexspi_edma_transfer_nsize_t *nsize* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address |
| *handle* | Pointer to flexspi_edma_handle_t structure |
| *nsize* | FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_EDMAnSize1Bytes(one byte). |

See Also

flexspi_edma_transfer_nsize_t .

#### 24.7.5.3 status_t FLEXSPI_TransferEDMA ( FLEXSPI_Type * *base*, flexspi_edma_handle_t * *handle*, flexspi_transfer_t * *xfer* )

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | Pointer to flexspi_edma_handle_t structure |
| *xfer* | FLEXSPI transfer structure. |

Return values

| | |
|---|---|
| *kStatus_FLEXSPI_Busy* | FLEXSPI is busy transfer. |
| *kStatus_InvalidArgument* | The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer. |
| *kStatus_Success* | FLEXSPI successfully start edma transfer. |

#### 24.7.5.4 void FLEXSPI_TransferAbortEDMA ( FLEXSPI_Type * *base*, flexspi_edma_handle_t * *handle* )

This function aborts the transfer data using eDMA.

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | Pointer to flexspi_edma_handle_t structure |

#### 24.7.5.5 status_t FLEXSPI_TransferGetCountEDMA ( FLEXSPI_Type * *base*, flexspi_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | FLEXSPI peripheral base address. |
| *handle* | Pointer to flexspi_edma_handle_t structure. |
| *count* | Bytes transfer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

# Chapter 25
# GPC: General Power Controller Driver

## 25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Power Controller (GPC) module of MCUXpresso SDK devices.

API functions are provided to configure the system about working in dedicated power mode. There are mainly about enabling the power for memory, enabling the wakeup sources for Partial Sleep mode, and power up/down operations.

## 25.2 Typical use case

1. Enable the wakeup source for Partial Sleep (PSLEEP) mode. In PSLEEP mode, HP domain is powered down, while LP domain remains powered on. Therefore, peripherals in LP domain can wakeup the system from PSLEEP mode via interrupts. In PSLEEP mode, system clocks are stopped and peripheral clocks of LP domain can be optionally on. LP domain peripherals can generate interrupt either asynchronously or need its peripheral clock on, depending on what kind of wakeup event is expected. Using the API of "GPC_EnablePartialSleepWakeupSource()" can enable the corresponding module as a wakeup source of PSLEEP mode. When the MCU exits from the PSLEEP mode, the API of "GPC_GetPartialSleepWakeupFlag()" can be used to check if the expected event occurs. However, to clear the wakeup flags, the user has to clear the interrupt status of the corresponding wakeup module.
2. Power up/down sequence. After the power up/down request, the power of HP domain would not be ready immediately, it would wait for a few delay so that the voltage is stable. Then switch the power of HP domain by hardware. To set the delay time, the driver privodes the API of "GPC_ConfigPowerUpSequence()" and "GPC_ConfigPowerDownSequence()".

## Enumerations

- enum _gpc_memory_power_gate {
  kGPC_MemoryPowerGateL2Cache = GPC_MPCTR_L2_PGE_MASK,
  kGPC_MemoryPowerGateITCM = GPC_MPCTR_ITCM_PGE_MASK,
  kGPC_MemoryPowerGateDTCM = GPC_MPCTR_DTCM_PGE_MASK }
  *Enumeration of the memory power gate control.*
- enum _gpc_status_flags { kGPC_PoweredDownFlag = GPC_PGS_PS_MASK }
  *GPC flags.*

## Driver version

- #define FSL_GPC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *GPC driver version 2.0.0.*

### Memory power Control

- static void GPC_EnableMemoryGate (GPC_Type *base, uint32_t modules, bool enable)
  *Control power for memory.*

### Partial sleep source

- void GPC_EnablePartialSleepWakeupSource (GPC_Type *base, gpc_wakeup_source_t source, bool enable)
  *Enable the modules as wakeup sources of PSLEEP (Partial Sleep) mode.*
- bool GPC_GetPartialSleepWakeupFlag (GPC_Type *base, gpc_wakeup_source_t source)
  *Get if indicated wakeup module just caused the wakeup event to exit PSLEEP mode.*
- static void GPC_EnablePartialSleepMode (GPC_Type *base, bool enable)
  *Switch to the Partial Sleep mode.*

### Power gate control

- static void GPC_ConfigPowerUpSequence (GPC_Type *base, uint32_t sw, uint32_t sw2iso)
  *Configure the power up sequence.*
- static void GPC_ConfigPowerDownSequence (GPC_Type *base, uint32_t iso, uint32_t iso2w)
  *Configure the power down sequence.*
- static uint32_t GPC_GetStatusFlags (GPC_Type *base)
  *Get the status flags of GPC.*
- static void GPC_ClearStatusFlags (GPC_Type *base, uint32_t flags)
  *Clear the status flags of GPC.*

## 25.3 Macro Definition Documentation

### 25.3.1 #define FSL_GPC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

## 25.4 Enumeration Type Documentation

### 25.4.1 enum _gpc_memory_power_gate

Once the clock gate is enabled, the responding part would be powered off and contents are not retained in Partial SLEEP mode.

Enumerator

***kGPC_MemoryPowerGateL2Cache*** L2 Cache Power Gate.
***kGPC_MemoryPowerGateITCM*** ITCM Power Gate Enable.
***kGPC_MemoryPowerGateDTCM*** DTCM Power Gate Enable.

### 25.4.2 enum _gpc_status_flags

Enumerator

***kGPC_PoweredDownFlag*** Power status. HP domain was powered down for the previous power down request.

## 25.5 Function Documentation

### 25.5.1 static void GPC_EnableMemoryGate ( GPC_Type * *base,* uint32_t *modules,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |
| *modules* | Mask value for Modules to be operated, see to _gpc_memory_power_gate. |
| *enable* | Enable the power or not. |

### 25.5.2 void GPC_EnablePartialSleepWakeupSource ( GPC_Type * *base*, gpc_wakeup_source_t *source*, bool *enable* )

In PSLEEP mode, HP domain is powered down, while LP domain remains powered on so peripherals in LP domain can wakeup the system from PSLEEP mode via interrupts. In PSLEEP mode, system clocks are stopped and peripheral clocks of LP domain can be optionally on. LP domain peripherals can generate interrupt either asynchronously or need its peripheral clock on, depending on what kind of wakeup event is expected. Refer to the corresponding module description about what kind of interrupts are supported by the module.

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |
| *source* | Wakeup source for responding module, see to #gpc_wakeup_source_t. |
| *enable* | Enable the wakeup source or not. |

### 25.5.3 bool GPC_GetPartialSleepWakeupFlag ( GPC_Type * *base*, gpc_wakeup_source_t *source* )

This function returns if the responding wakeup module just caused the MCU to exit PSLEEP mode. In hardware level, the flags of wakeup source are read only and will be cleared by cleaning the interrupt status of the corresponding wakeup module.

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |
| *source* | Wakeup source for responding module, see to #gpc_wakeup_source_t. |

Return values

| | |
|---|---|
| *true* | - Indicated wakeup flag is asserted. |
| *false* | - Indicated wakeup flag is not asserted. |

### 25.5.4 static void GPC_EnablePartialSleepMode ( GPC_Type * *base,* bool *enable* ) [inline], [static]

This function controls if the system will enter Partial SLEEP mode or remain in STOP mode.

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |
| *enable* | Enable the gate or not. |

### 25.5.5 static void GPC_ConfigPowerUpSequence ( GPC_Type * *base,* uint32_t *sw,* uint32_t *sw2iso* ) [inline], [static]

There will be two steps for power up sequence:

- After a power up request, GPC waits for a number of IP BUS clocks equal to the value of SW before turning on the power of HP domain. SW must not be programmed to zero.
- After GPC turnning on the power of HP domain, it waits for a number of IP BUS clocks equal to the value of SW2ISO before disable the isolation of HP domain. SW2ISO must not be programmed to zero.

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |
| *sw* | Count of IP BUS clocks before disabling the isolation of HP domain. |
| *sw2iso* | Count of IP BUS clocks before turning on the power of HP domain. |

### 25.5.6 static void GPC_ConfigPowerDownSequence ( GPC_Type * *base,* uint32_t *iso,* uint32_t *iso2w* ) [inline], [static]

There will be two steps for power down sequence:

- After a power down request, the GPC waits for a number of IP BUS clocks equal to the value of ISO before it enables the isolation of HP domain. ISO must not be programmed to zero.
- After HP domain is isolated, GPC waits for a number of IPG BUS clocks equal to the value of ISO2SW before it turning off the power of HP domain. ISO2SW must not be programmed to zero.

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |
| *iso* | Count of IP BUS clocks before it enables the isolation of HP domain. |
| *iso2w* | Count of IP BUS clocks before it turning off the power of HP domain. |

### 25.5.7 static uint32_t GPC_GetStatusFlags ( GPC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |

Returns

Mask value of flags, see to _gpc_status_flags.

### 25.5.8 static void GPC_ClearStatusFlags ( GPC_Type * *base,* uint32_t *flags* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPC peripheral base address. |
| *flags* | Mask value of flags to be cleared, see to _gpc_status_flags. |

# Chapter 26
# GPIO: General-Purpose Input/Output Driver

## 26.1 Overview

### Modules

- GPIO Driver

## 26.2 GPIO Driver

### 26.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

### 26.2.2 Typical use case

#### 26.2.2.1 Input Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/gpio

### Data Structures

- struct gpio_pin_config_t
  *GPIO Init structure definition. More...*

### Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  *GPIO direction definition.*
- enum gpio_interrupt_mode_t {
  kGPIO_NoIntmode = 0U,
  kGPIO_IntLowLevel = 1U,
  kGPIO_IntHighLevel = 2U,
  kGPIO_IntRisingEdge = 3U,
  kGPIO_IntFallingEdge = 4U,
  kGPIO_IntRisingOrFallingEdge = 5U }
  *GPIO interrupt mode definition.*

### Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *GPIO driver version 2.0.2.*

### GPIO Initialization and Configuration functions

- void GPIO_PinInit (GPIO_Type *base, uint32_t pin, const gpio_pin_config_t *Config)
  *Initializes the GPIO peripheral according to the specified parameters in the initConfig.*

## GPIO Reads and Write Functions

- void GPIO_PinWrite (GPIO_Type *base, uint32_t pin, uint8_t output)
  *Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void GPIO_WritePinOutput (GPIO_Type *base, uint32_t pin, uint8_t output)
  *Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void GPIO_PortSet (GPIO_Type *base, uint32_t mask)
  *Sets the output level of the multiple GPIO pins to the logic 1.*
- static void GPIO_SetPinsOutput (GPIO_Type *base, uint32_t mask)
  *Sets the output level of the multiple GPIO pins to the logic 1.*
- static void GPIO_PortClear (GPIO_Type *base, uint32_t mask)
  *Sets the output level of the multiple GPIO pins to the logic 0.*
- static void GPIO_ClearPinsOutput (GPIO_Type *base, uint32_t mask)
  *Sets the output level of the multiple GPIO pins to the logic 0.*
- static void GPIO_PortToggle (GPIO_Type *base, uint32_t mask)
  *Reverses the current output logic of the multiple GPIO pins.*
- static uint32_t GPIO_PinRead (GPIO_Type *base, uint32_t pin)
  *Reads the current input value of the GPIO port.*
- static uint32_t GPIO_ReadPinInput (GPIO_Type *base, uint32_t pin)
  *Reads the current input value of the GPIO port.*

## GPIO Reads Pad Status Functions

- static uint8_t GPIO_PinReadPadStatus (GPIO_Type *base, uint32_t pin)
  *Reads the current GPIO pin pad status.*
- static uint8_t GPIO_ReadPadStatus (GPIO_Type *base, uint32_t pin)
  *Reads the current GPIO pin pad status.*

## Interrupts and flags management functions

- void GPIO_PinSetInterruptConfig (GPIO_Type *base, uint32_t pin, gpio_interrupt_mode_t pinInterruptMode)
  *Sets the current pin interrupt mode.*
- static void GPIO_SetPinInterruptConfig (GPIO_Type *base, uint32_t pin, gpio_interrupt_mode_t pinInterruptMode)
  *Sets the current pin interrupt mode.*
- static void GPIO_PortEnableInterrupts (GPIO_Type *base, uint32_t mask)
  *Enables the specific pin interrupt.*
- static void GPIO_EnableInterrupts (GPIO_Type *base, uint32_t mask)
  *Enables the specific pin interrupt.*
- static void GPIO_PortDisableInterrupts (GPIO_Type *base, uint32_t mask)
  *Disables the specific pin interrupt.*
- static void GPIO_DisableInterrupts (GPIO_Type *base, uint32_t mask)
  *Disables the specific pin interrupt.*
- static uint32_t GPIO_PortGetInterruptFlags (GPIO_Type *base)
  *Reads individual pin interrupt status.*
- static uint32_t GPIO_GetPinsInterruptFlags (GPIO_Type *base)
  *Reads individual pin interrupt status.*

- static void GPIO_PortClearInterruptFlags (GPIO_Type ∗base, uint32_t mask)
  *Clears pin interrupt flag.*
- static void GPIO_ClearPinsInterruptFlags (GPIO_Type ∗base, uint32_t mask)
  *Clears pin interrupt flag.*

## 26.2.3 Data Structure Documentation

### 26.2.3.1 struct gpio_pin_config_t

#### Data Fields

- gpio_pin_direction_t direction
  *Specifies the pin direction.*
- uint8_t outputLogic
  *Set a default output logic, which has no use in input.*
- gpio_interrupt_mode_t interruptMode
  *Specifies the pin interrupt mode, a value of gpio_interrupt_mode_t.*

#### 26.2.3.1.0.6 Field Documentation

#### 26.2.3.1.0.6.1 gpio_pin_direction_t gpio_pin_config_t::direction

#### 26.2.3.1.0.6.2 gpio_interrupt_mode_t gpio_pin_config_t::interruptMode

## 26.2.4 Macro Definition Documentation

### 26.2.4.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

## 26.2.5 Enumeration Type Documentation

### 26.2.5.1 enum gpio_pin_direction_t

Enumerator

***kGPIO_DigitalInput*** Set current pin as digital input.
***kGPIO_DigitalOutput*** Set current pin as digital output.

### 26.2.5.2 enum gpio_interrupt_mode_t

Enumerator

***kGPIO_NoIntmode*** Set current pin general IO functionality.
***kGPIO_IntLowLevel*** Set current pin interrupt is low-level sensitive.
***kGPIO_IntHighLevel*** Set current pin interrupt is high-level sensitive.
***kGPIO_IntRisingEdge*** Set current pin interrupt is rising-edge sensitive.

***kGPIO_IntFallingEdge*** Set current pin interrupt is falling-edge sensitive.
***kGPIO_IntRisingOrFallingEdge*** Enable the edge select bit to override the ICR register's configuration.

### 26.2.6 Function Documentation

#### 26.2.6.1 void GPIO_PinInit ( GPIO_Type * *base*, uint32_t *pin*, const gpio_pin_config_t * *Config* )

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *pin* | Specifies the pin number |
| *initConfig* | pointer to a gpio_pin_config_t structure that contains the configuration information. |

#### 26.2.6.2 void GPIO_PinWrite ( GPIO_Type * *base*, uint32_t *pin*, uint8_t *output* )

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *pin* | GPIO port pin number. |
| *output* | GPIOpin output logic level.<br>• 0: corresponding pin output low-logic level.<br>• 1: corresponding pin output high-logic level. |

#### 26.2.6.3 static void GPIO_WritePinOutput ( GPIO_Type * *base*, uint32_t *pin*, uint8_t *output* ) [inline], [static]

#### 26.2.6.4 static void GPIO_PortSet ( GPIO_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| *mask* | GPIO pin number macro |

#### 26.2.6.5 static void GPIO_SetPinsOutput ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

#### 26.2.6.6 static void GPIO_PortClear ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| *mask* | GPIO pin number macro |

#### 26.2.6.7 static void GPIO_ClearPinsOutput ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

#### 26.2.6.8 static void GPIO_PortToggle ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| *mask* | GPIO pin number macro |

#### 26.2.6.9 static uint32_t GPIO_PinRead ( GPIO_Type * *base,* uint32_t *pin* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *pin* | GPIO port pin number. |

Return values

| | |
|---|---|
| *GPIO* | port input value. |

#### 26.2.6.10 static uint32_t GPIO_ReadPinInput ( GPIO_Type * *base*, uint32_t *pin* ) [inline], [static]

#### 26.2.6.11 static uint8_t GPIO_PinReadPadStatus ( GPIO_Type * *base*, uint32_t *pin* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *pin* | GPIO port pin number. |

Return values

| | |
|---|---|
| *GPIO* | pin pad status value. |

#### 26.2.6.12 static uint8_t GPIO_ReadPadStatus ( GPIO_Type * *base*, uint32_t *pin* ) [inline], [static]

#### 26.2.6.13 void GPIO_PinSetInterruptConfig ( GPIO_Type * *base*, uint32_t *pin*, gpio_interrupt_mode_t *pinInterruptMode* )

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *pin* | GPIO port pin number. |
| *pininterrupt-Mode* | pointer to a gpio_interrupt_mode_t structure that contains the interrupt mode information. |

#### 26.2.6.14 static void GPIO_SetPinInterruptConfig ( GPIO_Type * *base*, uint32_t *pin*, gpio_interrupt_mode_t *pinInterruptMode* ) [inline], [static]

#### 26.2.6.15 static void GPIO_PortEnableInterrupts ( GPIO_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *mask* | GPIO pin number macro. |

#### 26.2.6.16 static void GPIO_EnableInterrupts ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *mask* | GPIO pin number macro. |

#### 26.2.6.17 static void GPIO_PortDisableInterrupts ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *mask* | GPIO pin number macro. |

#### 26.2.6.18 static void GPIO_DisableInterrupts ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

#### 26.2.6.19 static uint32_t GPIO_PortGetInterruptFlags ( GPIO_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |

Return values

| | |
|---|---|
| *current* | pin interrupt status flag. |

#### 26.2.6.20 static uint32_t GPIO_GetPinsInterruptFlags ( GPIO_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |

Return values

| | |
|---|---|
| *current* | pin interrupt status flag. |

#### 26.2.6.21 static void GPIO_PortClearInterruptFlags ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *mask* | GPIO pin number macro. |

#### 26.2.6.22 static void GPIO_ClearPinsInterruptFlags ( GPIO_Type * *base,* uint32_t *mask* ) [inline], [static]

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

| | |
|---|---|
| *base* | GPIO base pointer. |
| *mask* | GPIO pin number macro. |

# Chapter 27
# GPT: General Purpose Timer

## 27.1 Overview

The MCUXpresso SDK provides a driver for the General Purpose Timer (GPT) of MCUXpresso SDK devices.

## 27.2 Function groups

The gpt driver supports the generation of PWM signals, input capture, and setting up the timer match conditions.

### 27.2.1 Initialization and deinitialization

The function GPT_Init() initializes the gpt with specified configurations. The function GPT_GetDefault-Config() gets the default configurations. The initialization function configures the restart/free-run mode and input selection when running.

The function GPT_Deinit() stops the timer and turns off the module clock.

## 27.3 Typical use case

### 27.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/gpt

## Data Structures

- struct gpt_config_t
  *Structure to configure the running mode. More...*

## Enumerations

- enum gpt_clock_source_t {
  kGPT_ClockSource_Off = 0U,
  kGPT_ClockSource_Periph = 1U,
  kGPT_ClockSource_HighFreq = 2U,
  kGPT_ClockSource_Ext = 3U,
  kGPT_ClockSource_LowFreq = 4U,
  kGPT_ClockSource_Osc = 5U }
  *List of clock sources.*

- enum gpt_input_capture_channel_t {
  kGPT_InputCapture_Channel1 = 0U,
  kGPT_InputCapture_Channel2 = 1U }
  *List of input capture channel number.*
- enum gpt_input_operation_mode_t {
  kGPT_InputOperation_Disabled = 0U,
  kGPT_InputOperation_RiseEdge = 1U,
  kGPT_InputOperation_FallEdge = 2U,
  kGPT_InputOperation_BothEdge = 3U }
  *List of input capture operation mode.*
- enum gpt_output_compare_channel_t {
  kGPT_OutputCompare_Channel1 = 0U,
  kGPT_OutputCompare_Channel2 = 1U,
  kGPT_OutputCompare_Channel3 = 2U }
  *List of output compare channel number.*
- enum gpt_output_operation_mode_t {
  kGPT_OutputOperation_Disconnected = 0U,
  kGPT_OutputOperation_Toggle = 1U,
  kGPT_OutputOperation_Clear = 2U,
  kGPT_OutputOperation_Set = 3U,
  kGPT_OutputOperation_Activelow = 4U }
  *List of output compare operation mode.*
- enum gpt_interrupt_enable_t {
  kGPT_OutputCompare1InterruptEnable = GPT_IR_OF1IE_MASK,
  kGPT_OutputCompare2InterruptEnable = GPT_IR_OF2IE_MASK,
  kGPT_OutputCompare3InterruptEnable = GPT_IR_OF3IE_MASK,
  kGPT_InputCapture1InterruptEnable = GPT_IR_IF1IE_MASK,
  kGPT_InputCapture2InterruptEnable = GPT_IR_IF2IE_MASK,
  kGPT_RollOverFlagInterruptEnable = GPT_IR_ROVIE_MASK }
  *List of GPT interrupts.*
- enum gpt_status_flag_t {
  kGPT_OutputCompare1Flag = GPT_SR_OF1_MASK,
  kGPT_OutputCompare2Flag = GPT_SR_OF2_MASK,
  kGPT_OutputCompare3Flag = GPT_SR_OF3_MASK,
  kGPT_InputCapture1Flag = GPT_SR_IF1_MASK,
  kGPT_InputCapture2Flag = GPT_SR_IF2_MASK,
  kGPT_RollOverFlag = GPT_SR_ROV_MASK }
  *Status flag.*

## Driver version

- #define FSL_GPT_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *Version 2.0.0.*

## Initialization and deinitialization

- void GPT_Init (GPT_Type *base, const gpt_config_t *initConfig)

*Initialize GPT to reset state and initialize running mode.*
- void GPT_Deinit (GPT_Type *base)
  *Disables the module and gates the GPT clock.*
- void GPT_GetDefaultConfig (gpt_config_t *config)
  *Fills in the GPT configuration structure with default settings.*

## Software Reset

- static void GPT_SoftwareReset (GPT_Type *base)
  *Software reset of GPT module.*

## Clock source and frequency control

- static void GPT_SetClockSource (GPT_Type *base, gpt_clock_source_t source)
  *Set clock source of GPT.*
- static gpt_clock_source_t GPT_GetClockSource (GPT_Type *base)
  *Get clock source of GPT.*
- static void GPT_SetClockDivider (GPT_Type *base, uint32_t divider)
  *Set pre scaler of GPT.*
- static uint32_t GPT_GetClockDivider (GPT_Type *base)
  *Get clock divider in GPT module.*
- static void GPT_SetOscClockDivider (GPT_Type *base, uint32_t divider)
  *OSC 24M pre-scaler before selected by clock source.*
- static uint32_t GPT_GetOscClockDivider (GPT_Type *base)
  *Get OSC 24M clock divider in GPT module.*

## Timer Start and Stop

- static void GPT_StartTimer (GPT_Type *base)
  *Start GPT timer.*
- static void GPT_StopTimer (GPT_Type *base)
  *Stop GPT timer.*

## Read the timer period

- static uint32_t GPT_GetCurrentTimerCount (GPT_Type *base)
  *Reads the current GPT counting value.*

## GPT Input/Output Signal Control

- static void GPT_SetInputOperationMode (GPT_Type *base, gpt_input_capture_channel_t channel, gpt_input_operation_mode_t mode)
  *Set GPT operation mode of input capture channel.*
- static gpt_input_operation_mode_t GPT_GetInputOperationMode (GPT_Type *base, gpt_input_capture_channel_t channel)
  *Get GPT operation mode of input capture channel.*
- static uint32_t GPT_GetInputCaptureValue (GPT_Type *base, gpt_input_capture_channel_t channel)
  *Get GPT input capture value of certain channel.*

- static void GPT_SetOutputOperationMode (GPT_Type *base, gpt_output_compare_channel_t channel, gpt_output_operation_mode_t mode)
  *Set GPT operation mode of output compare channel.*
- static gpt_output_operation_mode_t GPT_GetOutputOperationMode (GPT_Type *base, gpt_output_compare_channel_t channel)
  *Get GPT operation mode of output compare channel.*
- static void GPT_SetOutputCompareValue (GPT_Type *base, gpt_output_compare_channel_t channel, uint32_t value)
  *Set GPT output compare value of output compare channel.*
- static uint32_t GPT_GetOutputCompareValue (GPT_Type *base, gpt_output_compare_channel_t channel)
  *Get GPT output compare value of output compare channel.*
- static void GPT_ForceOutput (GPT_Type *base, gpt_output_compare_channel_t channel)
  *Force GPT output action on output compare channel, ignoring comparator.*

## GPT Interrupt and Status Interface

- static void GPT_EnableInterrupts (GPT_Type *base, uint32_t mask)
  *Enables the selected GPT interrupts.*
- static void GPT_DisableInterrupts (GPT_Type *base, uint32_t mask)
  *Disables the selected GPT interrupts.*
- static uint32_t GPT_GetEnabledInterrupts (GPT_Type *base)
  *Gets the enabled GPT interrupts.*

## Status Interface

- static uint32_t GPT_GetStatusFlags (GPT_Type *base, gpt_status_flag_t flags)
  *Get GPT status flags.*
- static void GPT_ClearStatusFlags (GPT_Type *base, gpt_status_flag_t flags)
  *Clears the GPT status flags.*

# 27.4 Data Structure Documentation

## 27.4.1 struct gpt_config_t

### Data Fields

- gpt_clock_source_t clockSource
  *clock source for GPT module.*
- uint32_t divider
  *clock divider (prescaler+1) from clock source to counter.*
- bool enableFreeRun
  *true: FreeRun mode, false: Restart mode.*
- bool enableRunInWait
  *GPT enabled in wait mode.*
- bool enableRunInStop
  *GPT enabled in stop mode.*
- bool enableRunInDoze
  *GPT enabled in doze mode.*

- bool enableRunInDbg
  *GPT enabled in debug mode.*
- bool enableMode
  `true: counter reset to 0 when enabled;`
  *false: counter retain its value when enabled.*

#### 27.4.1.0.0.7 Field Documentation

#### 27.4.1.0.0.7.1 gpt_clock_source_t gpt_config_t::clockSource

#### 27.4.1.0.0.7.2 uint32_t gpt_config_t::divider

#### 27.4.1.0.0.7.3 bool gpt_config_t::enableFreeRun

#### 27.4.1.0.0.7.4 bool gpt_config_t::enableRunInWait

#### 27.4.1.0.0.7.5 bool gpt_config_t::enableRunInStop

#### 27.4.1.0.0.7.6 bool gpt_config_t::enableRunInDoze

#### 27.4.1.0.0.7.7 bool gpt_config_t::enableRunInDbg

#### 27.4.1.0.0.7.8 bool gpt_config_t::enableMode

## 27.5 Enumeration Type Documentation

### 27.5.1 enum gpt_clock_source_t

Note

Actual number of clock sources is SoC dependent

Enumerator

***kGPT_ClockSource_Off*** GPT Clock Source Off.
***kGPT_ClockSource_Periph*** GPT Clock Source from Peripheral Clock.
***kGPT_ClockSource_HighFreq*** GPT Clock Source from High Frequency Reference Clock.
***kGPT_ClockSource_Ext*** GPT Clock Source from external pin.
***kGPT_ClockSource_LowFreq*** GPT Clock Source from Low Frequency Reference Clock.
***kGPT_ClockSource_Osc*** GPT Clock Source from Crystal oscillator.

### 27.5.2 enum gpt_input_capture_channel_t

Enumerator

***kGPT_InputCapture_Channel1*** GPT Input Capture Channel1.
***kGPT_InputCapture_Channel2*** GPT Input Capture Channel2.

### 27.5.3 enum gpt_input_operation_mode_t

Enumerator

***kGPT_InputOperation_Disabled*** Don't capture.
***kGPT_InputOperation_RiseEdge*** Capture on rising edge of input pin.
***kGPT_InputOperation_FallEdge*** Capture on falling edge of input pin.
***kGPT_InputOperation_BothEdge*** Capture on both edges of input pin.

### 27.5.4 enum gpt_output_compare_channel_t

Enumerator

***kGPT_OutputCompare_Channel1*** Output Compare Channel1.
***kGPT_OutputCompare_Channel2*** Output Compare Channel2.
***kGPT_OutputCompare_Channel3*** Output Compare Channel3.

### 27.5.5 enum gpt_output_operation_mode_t

Enumerator

***kGPT_OutputOperation_Disconnected*** Don't change output pin.
***kGPT_OutputOperation_Toggle*** Toggle output pin.
***kGPT_OutputOperation_Clear*** Set output pin low.
***kGPT_OutputOperation_Set*** Set output pin high.
***kGPT_OutputOperation_Activelow*** Generate a active low pulse on output pin.

### 27.5.6 enum gpt_interrupt_enable_t

Enumerator

***kGPT_OutputCompare1InterruptEnable*** Output Compare Channel1 interrupt enable.
***kGPT_OutputCompare2InterruptEnable*** Output Compare Channel2 interrupt enable.
***kGPT_OutputCompare3InterruptEnable*** Output Compare Channel3 interrupt enable.
***kGPT_InputCapture1InterruptEnable*** Input Capture Channel1 interrupt enable.
***kGPT_InputCapture2InterruptEnable*** Input Capture Channel1 interrupt enable.
***kGPT_RollOverFlagInterruptEnable*** Counter rolled over interrupt enable.

### 27.5.7 enum gpt_status_flag_t

Enumerator

***kGPT_OutputCompare1Flag*** Output compare channel 1 event.
***kGPT_OutputCompare2Flag*** Output compare channel 2 event.
***kGPT_OutputCompare3Flag*** Output compare channel 3 event.
***kGPT_InputCapture1Flag*** Input Capture channel 1 event.
***kGPT_InputCapture2Flag*** Input Capture channel 2 event.
***kGPT_RollOverFlag*** Counter reaches maximum value and rolled over to 0 event.

## 27.6 Function Documentation

### 27.6.1 void GPT_Init ( GPT_Type * *base*, const gpt_config_t * *initConfig* )

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *initConfig* | GPT mode setting configuration. |

### 27.6.2 void GPT_Deinit ( GPT_Type * *base* )

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

### 27.6.3 void GPT_GetDefaultConfig ( gpt_config_t * *config* )

The default values are:

```
*    config->clockSource = kGPT_ClockSource_Periph;
*    config->divider = 1U;
*    config->enableRunInStop = true;
*    config->enableRunInWait = true;
*    config->enableRunInDoze = false;
*    config->enableRunInDbg = false;
*    config->enableFreeRun = true;
*    config->enableMode  = true;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to the user configuration structure. |

### 27.6.4 static void GPT_SoftwareReset ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

### 27.6.5 static void GPT_SetClockSource ( GPT_Type * *base*, gpt_clock_source_t *source* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *source* | Clock source (see gpt_clock_source_t typedef enumeration). |

### 27.6.6 static gpt_clock_source_t GPT_GetClockSource ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

Returns

clock source (see gpt_clock_source_t typedef enumeration).

### 27.6.7 static void GPT_SetClockDivider ( GPT_Type * *base*, uint32_t *divider* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *divider* | Divider of GPT (1-4096). |

### 27.6.8 static uint32_t GPT_GetClockDivider ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

Returns

clock divider in GPT module (1-4096).

### 27.6.9 static void GPT_SetOscClockDivider ( GPT_Type * *base,* uint32_t *divider* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *divider* | OSC Divider(1-16). |

### 27.6.10 static uint32_t GPT_GetOscClockDivider ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

Returns

OSC clock divider in GPT module (1-16).

### 27.6.11 static void GPT_StartTimer ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

### 27.6.12 static void GPT_StopTimer ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

### 27.6.13 static uint32_t GPT_GetCurrentTimerCount ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |

Returns

Current GPT counter value.

### 27.6.14 static void GPT_SetInputOperationMode ( GPT_Type * *base*, gpt_input_capture_channel_t *channel*, gpt_input_operation_mode_t *mode* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT capture channel (see gpt_input_capture_channel_t typedef enumeration). |
| *mode* | GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration). |

### 27.6.15 static gpt_input_operation_mode_t GPT_GetInputOperationMode ( GPT_Type * *base*, gpt_input_capture_channel_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT capture channel (see gpt_input_capture_channel_t typedef enumeration). |

Returns

GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

### 27.6.16 static uint32_t GPT_GetInputCaptureValue ( GPT_Type * *base*, gpt_input_capture_channel_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT capture channel (see gpt_input_capture_channel_t typedef enumeration). |

Returns

GPT input capture value.

### 27.6.17 static void GPT_SetOutputOperationMode ( GPT_Type * *base*, gpt_output_compare_channel_t *channel*, gpt_output_operation_mode_t *mode* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration). |
| *mode* | GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration). |

### 27.6.18 static gpt_output_operation_mode_t GPT_GetOutputOperationMode ( GPT_Type * *base*, gpt_output_compare_channel_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration). |

Returns

GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

### 27.6.19 static void GPT_SetOutputCompareValue ( GPT_Type * *base*, gpt_output_compare_channel_t *channel*, uint32_t *value* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration). |
| *value* | GPT output compare value. |

### 27.6.20 static uint32_t GPT_GetOutputCompareValue ( GPT_Type * *base*, gpt_output_compare_channel_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration). |

Returns

GPT output compare value.

### 27.6.21 static void GPT_ForceOutput ( GPT_Type * *base*, gpt_output_compare_channel_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *channel* | GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration). |

### 27.6.22 static void GPT_EnableInterrupts ( GPT_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration gpt_interrupt_enable_t |

### 27.6.23 static void GPT_DisableInterrupts ( GPT_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address |
| *mask* | The interrupts to disable. This is a logical OR of members of the enumeration gpt_interrupt_enable_t |

### 27.6.24 static uint32_t GPT_GetEnabledInterrupts ( GPT_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address |

Returns

The enabled interrupts. This is the logical OR of members of the enumeration gpt_interrupt_enable_t

### 27.6.25 static uint32_t GPT_GetStatusFlags ( GPT_Type * *base,* gpt_status_flag_t *flags* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *flags* | GPT status flag mask (see gpt_status_flag_t for bit definition). |

Returns

GPT status, each bit represents one status flag.

### 27.6.26 static void GPT_ClearStatusFlags ( GPT_Type * *base*, gpt_status_flag_t *flags* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | GPT peripheral base address. |
| *flags* | GPT status flag mask (see gpt_status_flag_t for bit definition). |

# Chapter 28
# KPP: KeyPad Port Driver

## 28.1 Overview

The MCUXpresso SDK provides a peripheral driver for the KeyPad Port block of MCUXpresso SDK devices.

The KPP Initialize is to initialize for common configure: gate the KPP clock, configure columns, and rows features. The KPP Deinitialize is to ungate the clock.

The KPP provide the function to enable/disable interrupts. The KPP provide key press scanning function KPP_keyPressScanning. This API should be called by the Interrupt handler in application. KPP still provides functions to get and clear status flags.

## 28.2 Typical use case

## Data Structures

- struct kpp_config_t
  *Lists of KPP status. More...*

## Enumerations

- enum kpp_interrupt_enable_t {
  kKPP_keyDepressInterrupt = KPP_KPSR_KDIE_MASK,
  kKPP_keyReleaseInterrupt = KPP_KPSR_KRIE_MASK }
  *List of interrupts supported by the peripheral.*
- enum kpp_sync_operation_t {
  kKPP_ClearKeyDepressSyncChain = KPP_KPSR_KDSC_MASK,
  kKPP_SetKeyReleasesSyncChain = KPP_KPSR_KRSS_MASK }
  *Lists of KPP synchronize chain operation.*

## Driver version

- #define FSL_KPP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *KPP driver version 2.0.0.*

## Initialization and De-initialization

- void KPP_Init (KPP_Type *base, kpp_config_t *configure)
  *KPP initialize.*
- void KPP_Deinit (KPP_Type *base)
  *Deinitializes the KPP module and gates the clock.*

### KPP Basic Operation

- static void KPP_EnableInterrupts (KPP_Type *base, uint16_t mask)
  *Enable the interrupt.*
- static void KPP_DisableInterrupts (KPP_Type *base, uint16_t mask)
  *Disable the interrupt.*
- static uint16_t KPP_GetStatusFlag (KPP_Type *base)
  *Gets the KPP interrupt event status.*
- static void KPP_ClearStatusFlag (KPP_Type *base, uint16_t mask)
  *Clears KPP status flag.*
- static void KPP_SetSynchronizeChain (KPP_Type *base, uint16_t mask)
  *Set KPP synchronization chain.*
- void KPP_keyPressScanning (KPP_Type *base, uint8_t *data, uint32_t clockSrc_Hz)
  *Keypad press scanning.*

## 28.3 Data Structure Documentation

### 28.3.1 struct kpp_config_t

#### Data Fields

- uint8_t activeRow
  *The row number: bit 7 ~ 0 represents the row 7 ~ 0.*
- uint8_t activeColumn
  *The column number: bit 7 ~ 0 represents the column 7 ~ 0.*
- uint16_t interrupt
  *KPP interrupt source.*

##### 28.3.1.0.0.8 Field Documentation

###### 28.3.1.0.0.8.1 uint8_t kpp_config_t::activeRow

###### 28.3.1.0.0.8.2 uint8_t kpp_config_t::activeColumn

###### 28.3.1.0.0.8.3 uint16_t kpp_config_t::interrupt

A logical OR of "kpp_interrupt_enable_t".

## 28.4 Macro Definition Documentation

### 28.4.1 #define FSL_KPP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

## 28.5 Enumeration Type Documentation

### 28.5.1 enum kpp_interrupt_enable_t

This enumeration uses one-bot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Enumerator

***kKPP_keyDepressInterrupt*** Keypad depress interrupt source.
***kKPP_keyReleaseInterrupt*** Keypad release interrupt source.

### 28.5.2 enum kpp_sync_operation_t

Enumerator

***kKPP_ClearKeyDepressSyncChain*** Keypad depress interrupt status.
***kKPP_SetKeyReleasesSyncChain*** Keypad release interrupt status.

## 28.6 Function Documentation

### 28.6.1 void KPP_Init ( KPP_Type * *base*, kpp_config_t * *configure* )

This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |
| *configure* | The KPP configuration structure pointer. |

### 28.6.2 void KPP_Deinit ( KPP_Type * *base* )

This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |

### 28.6.3 static void KPP_EnableInterrupts ( KPP_Type * *base*, uint16_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |
| *mask* | KPP interrupts to enable. This is a logical OR of the enumeration :: kpp_interrupt_-enable_t. |

### 28.6.4 static void KPP_DisableInterrupts ( KPP_Type * *base,* uint16_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |
| *mask* | KPP interrupts to disable. This is a logical OR of the enumeration :: kpp_interrupt_-enable_t. |

### 28.6.5 static uint16_t KPP_GetStatusFlag ( KPP_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |

Returns

The status of the KPP. Application can use the enum type in the "kpp_interrupt_enable_t" to get the right status of the related event.

### 28.6.6 static void KPP_ClearStatusFlag ( KPP_Type * *base,* uint16_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |
| *mask* | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_interrupt_-enable_t. |

### 28.6.7 static void KPP_SetSynchronizeChain ( KPP_Type * *base,* uint16_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |
| *mask* | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_sync_-operation_t. |

### 28.6.8 void KPP_keyPressScanning ( KPP_Type * *base,* uint8_t * *data,* uint32_t *clockSrc_Hz* )

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

Parameters

| | |
|---|---|
| *base* | KPP peripheral base address. |
| *data* | KPP key press scanning data. The data buffer should be prepared with length at least equal to KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX. the data pointer is recommended to be a array like uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]. for example the data[2] = 4, that means in column 1 row 2 has a key press event. |

# Chapter 29
# LPI2C: Low Power I2C Driver

## 29.1 Overview

### Modules

- LPI2C FreeRTOS Driver
- LPI2C Master DMA Driver
- LPI2C Master Driver
- LPI2C Slave Driver

### Macros

- #define LPI2C_WAIT_TIMEOUT 0U /* Define to zero means keep waiting until the flag is assert/deassert. */
  *Timeout times for waiting flag.*

### Enumerations

- enum _lpi2c_status {
  kStatus_LPI2C_Busy = MAKE_STATUS(kStatusGroup_LPI2C, 0),
  kStatus_LPI2C_Idle = MAKE_STATUS(kStatusGroup_LPI2C, 1),
  kStatus_LPI2C_Nak = MAKE_STATUS(kStatusGroup_LPI2C, 2),
  kStatus_LPI2C_FifoError = MAKE_STATUS(kStatusGroup_LPI2C, 3),
  kStatus_LPI2C_BitError = MAKE_STATUS(kStatusGroup_LPI2C, 4),
  kStatus_LPI2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_LPI2C, 5),
  kStatus_LPI2C_PinLowTimeout,
  kStatus_LPI2C_NoTransferInProgress,
  kStatus_LPI2C_DmaRequestFail = MAKE_STATUS(kStatusGroup_LPI2C, 8),
  kStatus_LPI2C_Timeout = MAKE_STATUS(kStatusGroup_LPI2C, 9) }
  *LPI2C status return codes.*

### Driver version

- #define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))
  *LPI2C driver version 2.1.9.*

## 29.2 Macro Definition Documentation

### 29.2.1 #define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))

### 29.2.2 #define LPI2C_WAIT_TIMEOUT 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

## 29.3 Enumeration Type Documentation

### 29.3.1 enum _lpi2c_status

Enumerator

***kStatus_LPI2C_Busy*** The master is already performing a transfer.
***kStatus_LPI2C_Idle*** The slave driver is idle.
***kStatus_LPI2C_Nak*** The slave device sent a NAK in response to a byte.
***kStatus_LPI2C_FifoError*** FIFO under run or overrun.
***kStatus_LPI2C_BitError*** Transferred bit was not seen on the bus.
***kStatus_LPI2C_ArbitrationLost*** Arbitration lost error.
***kStatus_LPI2C_PinLowTimeout*** SCL or SDA were held low longer than the timeout.
***kStatus_LPI2C_NoTransferInProgress*** Attempt to abort a transfer when one is not in progress.
***kStatus_LPI2C_DmaRequestFail*** DMA request failed.
***kStatus_LPI2C_Timeout*** Timeout poling status flags.

## 29.4 LPI2C Master Driver

### 29.4.1 Overview

#### Data Structures

- struct lpi2c_master_config_t
  *Structure with settings to initialize the LPI2C master module. More...*
- struct lpi2c_data_match_config_t
  *LPI2C master data match configuration structure. More...*
- struct lpi2c_master_transfer_t
  *Non-blocking transfer descriptor structure. More...*
- struct lpi2c_master_handle_t
  *Driver handle for master non-blocking APIs. More...*

#### Typedefs

- 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.*

#### Enumerations

- enum _lpi2c_master_flags {
  kLPI2C_MasterTxReadyFlag = LPI2C_MSR_TDF_MASK,
  kLPI2C_MasterRxReadyFlag = LPI2C_MSR_RDF_MASK,
  kLPI2C_MasterEndOfPacketFlag = LPI2C_MSR_EPF_MASK,
  kLPI2C_MasterStopDetectFlag = LPI2C_MSR_SDF_MASK,
  kLPI2C_MasterNackDetectFlag = LPI2C_MSR_NDF_MASK,
  kLPI2C_MasterArbitrationLostFlag = LPI2C_MSR_ALF_MASK,
  kLPI2C_MasterFifoErrFlag = LPI2C_MSR_FEF_MASK,
  kLPI2C_MasterPinLowTimeoutFlag = LPI2C_MSR_PLTF_MASK,
  kLPI2C_MasterDataMatchFlag = LPI2C_MSR_DMF_MASK,
  kLPI2C_MasterBusyFlag = LPI2C_MSR_MBF_MASK,
  kLPI2C_MasterBusBusyFlag = LPI2C_MSR_BBF_MASK }
  *LPI2C master peripheral flags.*
- enum lpi2c_direction_t {
  kLPI2C_Write = 0U,
  kLPI2C_Read = 1U }
  *Direction of master and slave transfers.*
- enum lpi2c_master_pin_config_t {

kLPI2C_2PinOpenDrain = 0x0U,
kLPI2C_2PinOutputOnly = 0x1U,
kLPI2C_2PinPushPull = 0x2U,
kLPI2C_4PinPushPull = 0x3U,
kLPI2C_2PinOpenDrainWithSeparateSlave,
kLPI2C_2PinOutputOnlyWithSeparateSlave,
kLPI2C_2PinPushPullWithSeparateSlave,
kLPI2C_4PinPushPullWithInvertedOutput = 0x7U }
*LPI2C pin configuration.*

- enum lpi2c_host_request_source_t {
  kLPI2C_HostRequestExternalPin = 0x0U,
  kLPI2C_HostRequestInputTrigger = 0x1U }
  *LPI2C master host request selection.*
- enum lpi2c_host_request_polarity_t {
  kLPI2C_HostRequestPinActiveLow = 0x0U,
  kLPI2C_HostRequestPinActiveHigh = 0x1U }
  *LPI2C master host request pin polarity configuration.*
- enum lpi2c_data_match_config_mode_t {
  kLPI2C_MatchDisabled = 0x0U,
  kLPI2C_1stWordEqualsM0OrM1 = 0x2U,
  kLPI2C_AnyWordEqualsM0OrM1 = 0x3U,
  kLPI2C_1stWordEqualsM0And2ndWordEqualsM1,
  kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1,
  kLPI2C_1stWordAndM1EqualsM0AndM1,
  kLPI2C_AnyWordAndM1EqualsM0AndM1 }
  *LPI2C master data match configuration modes.*
- enum _lpi2c_master_transfer_flags {
  kLPI2C_TransferDefaultFlag = 0x00U,
  kLPI2C_TransferNoStartFlag = 0x01U,
  kLPI2C_TransferRepeatedStartFlag = 0x02U,
  kLPI2C_TransferNoStopFlag = 0x04U }
  *Transfer option flags.*

## Initialization and deinitialization

- void LPI2C_MasterGetDefaultConfig (lpi2c_master_config_t *masterConfig)
  *Provides a default configuration for the LPI2C master peripheral.*
- void LPI2C_MasterInit (LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t sourceClock_Hz)
  *Initializes the LPI2C master peripheral.*
- void LPI2C_MasterDeinit (LPI2C_Type *base)
  *Deinitializes the LPI2C master peripheral.*
- void LPI2C_MasterConfigureDataMatch (LPI2C_Type *base, const lpi2c_data_match_config_t *config)
  *Configures LPI2C master data match feature.*
- status_t **LPI2C_MasterCheckAndClearError** (LPI2C_Type *base, uint32_t status)

- status_t **LPI2C_CheckForBusyBus** (LPI2C_Type *base)
- static void LPI2C_MasterReset (LPI2C_Type *base)
  *Performs a software reset.*
- static void LPI2C_MasterEnable (LPI2C_Type *base, bool enable)
  *Enables or disables the LPI2C module as master.*

### Status

- static uint32_t LPI2C_MasterGetStatusFlags (LPI2C_Type *base)
  *Gets the LPI2C master status flags.*
- static void LPI2C_MasterClearStatusFlags (LPI2C_Type *base, uint32_t statusMask)
  *Clears the LPI2C master status flag state.*

### Interrupts

- static void LPI2C_MasterEnableInterrupts (LPI2C_Type *base, uint32_t interruptMask)
  *Enables the LPI2C master interrupt requests.*
- static void LPI2C_MasterDisableInterrupts (LPI2C_Type *base, uint32_t interruptMask)
  *Disables the LPI2C master interrupt requests.*
- static uint32_t LPI2C_MasterGetEnabledInterrupts (LPI2C_Type *base)
  *Returns the set of currently enabled LPI2C master interrupt requests.*

### DMA control

- static void LPI2C_MasterEnableDMA (LPI2C_Type *base, bool enableTx, bool enableRx)
  *Enables or disables LPI2C master DMA requests.*
- static uint32_t LPI2C_MasterGetTxFifoAddress (LPI2C_Type *base)
  *Gets LPI2C master transmit data register address for DMA transfer.*
- static uint32_t LPI2C_MasterGetRxFifoAddress (LPI2C_Type *base)
  *Gets LPI2C master receive data register address for DMA transfer.*

### FIFO control

- static void LPI2C_MasterSetWatermarks (LPI2C_Type *base, size_t txWords, size_t rxWords)
  *Sets the watermarks for LPI2C master FIFOs.*
- static void LPI2C_MasterGetFifoCounts (LPI2C_Type *base, size_t *rxCount, size_t *txCount)
  *Gets the current number of words in the LPI2C master FIFOs.*

### Bus operations

- void LPI2C_MasterSetBaudRate (LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)
  *Sets the I2C bus frequency for master transactions.*
- static bool LPI2C_MasterGetBusIdleState (LPI2C_Type *base)

*Returns whether the 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.*
- static 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.*
- status_t LPI2C_MasterSend (LPI2C_Type *base, void *txBuff, size_t txSize)
  *Performs a polling send transfer on the I2C bus.*
- status_t LPI2C_MasterReceive (LPI2C_Type *base, void *rxBuff, size_t rxSize)
  *Performs a polling receive transfer on the I2C bus.*
- status_t LPI2C_MasterStop (LPI2C_Type *base)
  *Sends a STOP signal on the I2C bus.*
- status_t LPI2C_MasterTransferBlocking (LPI2C_Type *base, lpi2c_master_transfer_t *transfer)
  *Performs a master polling transfer on the I2C bus.*

### Non-blocking

- 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.*
- 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.*
- status_t LPI2C_MasterTransferGetCount (LPI2C_Type *base, lpi2c_master_handle_t *handle, size_t *count)
  *Returns number of bytes transferred so far.*
- void LPI2C_MasterTransferAbort (LPI2C_Type *base, lpi2c_master_handle_t *handle)
  *Terminates a non-blocking LPI2C master transmission early.*

### IRQ handler

- void LPI2C_MasterTransferHandleIRQ (LPI2C_Type *base, lpi2c_master_handle_t *handle)
  *Reusable routine to handle master interrupts.*

## 29.4.2 Data Structure Documentation

### 29.4.2.1 struct lpi2c_master_config_t

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.

### Data Fields

- 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.*
- uint32_t pinLowTimeout_ns
  *Pin low timeout in nanoseconds.*
- uint8_t sdaGlitchFilterWidth_ns
  *Width in nanoseconds of glitch filter on SDA pin.*
- uint8_t sclGlitchFilterWidth_ns
  *Width in nanoseconds of glitch filter on SCL pin.*
- struct {
  bool enable
  *Enable host request.*
  lpi2c_host_request_source_t source
  *Host request source.*
  lpi2c_host_request_polarity_t polarity
  *Host request pin polarity.*
  } hostRequest

  *Host request options.*

#### 29.4.2.1.0.9 Field Documentation

#### 29.4.2.1.0.9.1 bool lpi2c_master_config_t::enableMaster

#### 29.4.2.1.0.9.2 bool lpi2c_master_config_t::enableDoze

#### 29.4.2.1.0.9.3 bool lpi2c_master_config_t::debugEnable

#### 29.4.2.1.0.9.4 bool lpi2c_master_config_t::ignoreAck

#### 29.4.2.1.0.9.5 lpi2c_master_pin_config_t lpi2c_master_config_t::pinConfig

#### 29.4.2.1.0.9.6 uint32_t lpi2c_master_config_t::baudRate_Hz

#### 29.4.2.1.0.9.7 uint32_t lpi2c_master_config_t::busIdleTimeout_ns

Set to 0 to disable.

#### 29.4.2.1.0.9.8 uint32_t lpi2c_master_config_t::pinLowTimeout_ns

Set to 0 to disable.

#### 29.4.2.1.0.9.9 uint8_t lpi2c_master_config_t::sdaGlitchFilterWidth_ns

Set to 0 to disable.

#### 29.4.2.1.0.9.10 uint8_t lpi2c_master_config_t::sclGlitchFilterWidth_ns

Set to 0 to disable.

#### 29.4.2.1.0.9.11 bool lpi2c_master_config_t::enable

#### 29.4.2.1.0.9.12 lpi2c_host_request_source_t lpi2c_master_config_t::source

#### 29.4.2.1.0.9.13 lpi2c_host_request_polarity_t lpi2c_master_config_t::polarity

#### 29.4.2.1.0.9.14 struct { ... } lpi2c_master_config_t::hostRequest

### 29.4.2.2 struct lpi2c_data_match_config_t

### Data Fields

- 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.*

#### 29.4.2.2.0.10 Field Documentation

#### 29.4.2.2.0.10.1 lpi2c_data_match_config_mode_t lpi2c_data_match_config_t::matchMode

#### 29.4.2.2.0.10.2 bool lpi2c_data_match_config_t::rxDataMatchOnly

#### 29.4.2.2.0.10.3 uint32_t lpi2c_data_match_config_t::match0

#### 29.4.2.2.0.10.4 uint32_t lpi2c_data_match_config_t::match1

### 29.4.2.3 struct _lpi2c_master_transfer

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

### Data Fields

- uint32_t flags

*Bit mask of options for the transfer.*
- uint16_t slaveAddress
  *The 7-bit slave address.*
- lpi2c_direction_t direction
  *Either kLPI2C_Read or kLPI2C_Write.*
- uint32_t subaddress
  *Sub address.*
- size_t subaddressSize
  *Length of sub address to send in bytes.*
- void * data
  *Pointer to data to transfer.*
- size_t dataSize
  *Number of bytes to transfer.*

#### 29.4.2.3.0.11 Field Documentation

##### 29.4.2.3.0.11.1 uint32_t lpi2c_master_transfer_t::flags

See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefaultFlag for normal transfers.

##### 29.4.2.3.0.11.2 uint16_t lpi2c_master_transfer_t::slaveAddress

##### 29.4.2.3.0.11.3 lpi2c_direction_t lpi2c_master_transfer_t::direction

##### 29.4.2.3.0.11.4 uint32_t lpi2c_master_transfer_t::subaddress

Transferred MSB first.

##### 29.4.2.3.0.11.5 size_t lpi2c_master_transfer_t::subaddressSize

Maximum size is 4 bytes.

##### 29.4.2.3.0.11.6 void* lpi2c_master_transfer_t::data

##### 29.4.2.3.0.11.7 size_t lpi2c_master_transfer_t::dataSize

### 29.4.2.4 struct _lpi2c_master_handle

Note

The contents of this structure are private and subject to change.

### Data Fields

- 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 [7]
  *LPI2C command sequence.*
- 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.*

#### 29.4.2.4.0.12 Field Documentation

##### 29.4.2.4.0.12.1 uint8_t lpi2c_master_handle_t::state

##### 29.4.2.4.0.12.2 uint16_t lpi2c_master_handle_t::remainingBytes

##### 29.4.2.4.0.12.3 uint8_t∗ lpi2c_master_handle_t::buf

##### 29.4.2.4.0.12.4 uint16_t lpi2c_master_handle_t::commandBuffer[7]

##### 29.4.2.4.0.12.5 lpi2c_master_transfer_t lpi2c_master_handle_t::transfer

##### 29.4.2.4.0.12.6 lpi2c_master_transfer_callback_t lpi2c_master_handle_t::completionCallback

##### 29.4.2.4.0.12.7 void∗ lpi2c_master_handle_t::userData

### 29.4.3 Typedef Documentation

#### 29.4.3.1 typedef void(∗ lpi2c_master_transfer_callback_t)(LPI2C_Type ∗base, lpi2c_master_handle_t ∗handle, status_t completionStatus, void ∗userData)

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().

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *completion-Status* | Either #kStatus_Success or an error code describing how the transfer completed. |
| *userData* | Arbitrary pointer-sized value passed from the application. |

### 29.4.4 Enumeration Type Documentation

#### 29.4.4.1 enum _lpi2c_master_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.

Enumerator

***kLPI2C_MasterTxReadyFlag*** Transmit data flag.
***kLPI2C_MasterRxReadyFlag*** Receive data flag.
***kLPI2C_MasterEndOfPacketFlag*** End Packet flag.
***kLPI2C_MasterStopDetectFlag*** Stop detect flag.
***kLPI2C_MasterNackDetectFlag*** NACK detect flag.
***kLPI2C_MasterArbitrationLostFlag*** Arbitration lost flag.
***kLPI2C_MasterFifoErrFlag*** FIFO error flag.
***kLPI2C_MasterPinLowTimeoutFlag*** Pin low timeout flag.
***kLPI2C_MasterDataMatchFlag*** Data match flag.
***kLPI2C_MasterBusyFlag*** Master busy flag.
***kLPI2C_MasterBusBusyFlag*** Bus busy flag.

#### 29.4.4.2 enum lpi2c_direction_t

Enumerator

***kLPI2C_Write*** Master transmit.
***kLPI2C_Read*** Master receive.

#### 29.4.4.3 enum lpi2c_master_pin_config_t

Enumerator

***kLPI2C_2PinOpenDrain*** LPI2C Configured for 2-pin open drain mode.
***kLPI2C_2PinOutputOnly*** LPI2C Configured for 2-pin output only mode (ultra-fast mode)
***kLPI2C_2PinPushPull*** LPI2C Configured for 2-pin push-pull mode.
***kLPI2C_4PinPushPull*** LPI2C Configured for 4-pin push-pull mode.
***kLPI2C_2PinOpenDrainWithSeparateSlave*** LPI2C Configured for 2-pin open drain mode with separate LPI2C slave.

*kLPI2C_2PinOutputOnlyWithSeparateSlave* LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave.
*kLPI2C_2PinPushPullWithSeparateSlave* LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave.
*kLPI2C_4PinPushPullWithInvertedOutput* LPI2C Configured for 4-pin push-pull mode(inverted outputs)

#### 29.4.4.4 enum lpi2c_host_request_source_t

Enumerator

*kLPI2C_HostRequestExternalPin* Select the LPI2C_HREQ pin as the host request input.
*kLPI2C_HostRequestInputTrigger* Select the input trigger as the host request input.

#### 29.4.4.5 enum lpi2c_host_request_polarity_t

Enumerator

*kLPI2C_HostRequestPinActiveLow* Configure the LPI2C_HREQ pin active low.
*kLPI2C_HostRequestPinActiveHigh* Configure the LPI2C_HREQ pin active high.

#### 29.4.4.6 enum lpi2c_data_match_config_mode_t

Enumerator

*kLPI2C_MatchDisabled* LPI2C Match Disabled.
*kLPI2C_1stWordEqualsM0OrM1* LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1.
*kLPI2C_AnyWordEqualsM0OrM1* LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1.
*kLPI2C_1stWordEqualsM0And2ndWordEqualsM1* LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1.
*kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1* LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1.
*kLPI2C_1stWordAndM1EqualsM0AndM1* LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1.
*kLPI2C_AnyWordAndM1EqualsM0AndM1* LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1.

#### 29.4.4.7 enum _lpi2c_master_transfer_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.

Enumerator

***kLPI2C_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.
***kLPI2C_TransferNoStartFlag*** Don't send a start condition, address, and sub address.
***kLPI2C_TransferRepeatedStartFlag*** Send a repeated start condition.
***kLPI2C_TransferNoStopFlag*** Don't send a stop condition.

### 29.4.5 Function Documentation

#### 29.4.5.1 void LPI2C_MasterGetDefaultConfig ( lpi2c_master_config_t * *masterConfig* )

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

| | | |
|---|---|---|
| out | *masterConfig* | User provided configuration structure for default values. Refer to lpi2c-_master_config_t. |

#### 29.4.5.2 void LPI2C_MasterInit ( LPI2C_Type * *base*, const lpi2c_master_config_t * *masterConfig*, uint32_t *sourceClock_Hz* )

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. |

#### 29.4.5.3 void LPI2C_MasterDeinit ( LPI2C_Type * *base* )

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. |

#### 29.4.5.4 void LPI2C_MasterConfigureDataMatch ( LPI2C_Type * *base*, const lpi2c_data_match_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *config* | Settings for the data match feature. |

#### 29.4.5.5 static void LPI2C_MasterReset ( LPI2C_Type * *base* ) [inline], [static]

Restores the LPI2C master peripheral to reset conditions.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |

#### 29.4.5.6 static void LPI2C_MasterEnable ( LPI2C_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *enable* | Pass true to enable or false to disable the specified LPI2C as master. |

#### 29.4.5.7 static uint32_t LPI2C_MasterGetStatusFlags ( LPI2C_Type * *base* ) [inline], [static]

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.

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.

See Also

_lpi2c_master_flags

#### 29.4.5.8 static void LPI2C_MasterClearStatusFlags ( LPI2C_Type * *base*, uint32_t *statusMask* ) [inline], [static]

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.

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(). |

See Also

_lpi2c_master_flags.

#### 29.4.5.9 static void LPI2C_MasterEnableInterrupts ( LPI2C_Type * *base*, uint32_t *interruptMask* ) [inline], [static]

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. |

#### 29.4.5.10 static void LPI2C_MasterDisableInterrupts ( LPI2C_Type * *base*, uint32_t *interruptMask* ) [inline], [static]

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. |

#### 29.4.5.11 static uint32_t LPI2C_MasterGetEnabledInterrupts ( LPI2C_Type * *base* ) [inline], [static]

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.

#### 29.4.5.12 static void LPI2C_MasterEnableDMA ( LPI2C_Type * *base*, bool *enableTx*, bool *enableRx* ) [inline], [static]

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. |

#### 29.4.5.13 static uint32_t LPI2C_MasterGetTxFifoAddress ( LPI2C_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |

Returns

The LPI2C Master Transmit Data Register address.

#### 29.4.5.14 static uint32_t LPI2C_MasterGetRxFifoAddress ( LPI2C_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |

Returns

The LPI2C Master Receive Data Register address.

#### 29.4.5.15 static void LPI2C_MasterSetWatermarks ( LPI2C_Type * *base*, size_t *txWords*, size_t *rxWords* ) [inline], [static]

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. |

#### 29.4.5.16 static void LPI2C_MasterGetFifoCounts ( LPI2C_Type * *base*, size_t * *rxCount*, size_t * *txCount* ) [inline], [static]

Parameters

| | | |
|---|---|---|
| | *base* | The LPI2C peripheral base address. |
| out | *txCount* | Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | *rxCount* | Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required. |

#### 29.4.5.17 void LPI2C_MasterSetBaudRate ( LPI2C_Type * *base*, uint32_t *sourceClock_Hz*, uint32_t *baudRate_Hz* )

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. |

### 29.4.5.18 static bool LPI2C_MasterGetBusIdleState ( LPI2C_Type * *base* ) [inline], [static]

Requires the master mode to be enabled.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |

Return values

| | |
|---|---|
| *true* | Bus is busy. |
| *false* | Bus is idle. |

### 29.4.5.19 status_t LPI2C_MasterStart ( LPI2C_Type * *base*, uint8_t *address*, lpi2c_direction_t *dir* )

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. |

#### 29.4.5.20 static status_t LPI2C_MasterRepeatedStart ( LPI2C_Type * *base*, uint8_t *address*, lpi2c_direction_t *dir* ) [inline], [static]

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. |

#### 29.4.5.21 status_t LPI2C_MasterSend ( LPI2C_Type * *base*, void * *txBuff*, size_t *txSize* )

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_PinLow-Timeout* | SCL or SDA were held low longer than the timeout. |

#### 29.4.5.22 status_t LPI2C_MasterReceive ( LPI2C_Type * *base*, void * *rxBuff*, size_t *rxSize* )

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_PinLow-Timeout* | SCL or SDA were held low longer than the timeout. |

#### 29.4.5.23 status_t LPI2C_MasterStop ( LPI2C_Type * *base* )

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_PinLow-Timeout* | SCL or SDA were held low longer than the timeout. |

### 29.4.5.24 status_t LPI2C_MasterTransferBlocking ( LPI2C_Type * *base*, lpi2c_master_transfer_t * *transfer* )

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_PinLow-Timeout* | SCL or SDA were held low longer than the timeout. |

### 29.4.5.25 void LPI2C_MasterTransferCreateHandle ( LPI2C_Type * *base*, lpi2c_master_handle_t * *handle*, lpi2c_master_transfer_callback_t *callback*, void * *userData* )

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. |
| out | *handle* | 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. |

### 29.4.5.26 status_t LPI2C_MasterTransferNonBlocking ( LPI2C_Type * *base*, lpi2c_master_handle_t * *handle*, lpi2c_master_transfer_t * *transfer* )

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. |

### 29.4.5.27 status_t LPI2C_MasterTransferGetCount ( LPI2C_Type * *base*, lpi2c_master_handle_t * *handle*, size_t * *count* )

Parameters

| | | |
|---|---|---|
| | *base* | The LPI2C peripheral base address. |
| | *handle* | Pointer to the LPI2C master driver handle. |
| out | *count* | 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. |

### 29.4.5.28 void LPI2C_MasterTransferAbort ( LPI2C_Type * *base*, lpi2c_master_handle_t * *handle* )

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *handle* | Pointer to the LPI2C master driver handle. |

Return values

| | |
|---|---|
| *#kStatus_Success* | A transaction was successfully aborted. |
| *kStatus_LPI2C_Idle* | There is not a non-blocking transaction currently in progress. |

#### 29.4.5.29 void LPI2C_MasterTransferHandleIRQ ( LPI2C_Type * *base*, lpi2c_master_handle_t * *handle* )

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *handle* | Pointer to the LPI2C master driver handle. |

## 29.5 LPI2C Slave Driver

### 29.5.1 Overview

#### Data Structures

- struct lpi2c_slave_config_t
  *Structure with settings to initialize the LPI2C slave module. More...*
- struct lpi2c_slave_transfer_t
  *LPI2C slave transfer structure. More...*
- struct lpi2c_slave_handle_t
  *LPI2C slave handle structure. More...*

#### Typedefs

- typedef void(∗ lpi2c_slave_transfer_callback_t )(LPI2C_Type ∗base, lpi2c_slave_transfer_t ∗transfer, void ∗userData)
  *Slave event callback function pointer type.*

#### Enumerations

- enum _lpi2c_slave_flags {
  kLPI2C_SlaveTxReadyFlag = LPI2C_SSR_TDF_MASK,
  kLPI2C_SlaveRxReadyFlag = LPI2C_SSR_RDF_MASK,
  kLPI2C_SlaveAddressValidFlag = LPI2C_SSR_AVF_MASK,
  kLPI2C_SlaveTransmitAckFlag = LPI2C_SSR_TAF_MASK,
  kLPI2C_SlaveRepeatedStartDetectFlag = LPI2C_SSR_RSF_MASK,
  kLPI2C_SlaveStopDetectFlag = LPI2C_SSR_SDF_MASK,
  kLPI2C_SlaveBitErrFlag = LPI2C_SSR_BEF_MASK,
  kLPI2C_SlaveFifoErrFlag = LPI2C_SSR_FEF_MASK,
  kLPI2C_SlaveAddressMatch0Flag = LPI2C_SSR_AM0F_MASK,
  kLPI2C_SlaveAddressMatch1Flag = LPI2C_SSR_AM1F_MASK,
  kLPI2C_SlaveGeneralCallFlag = LPI2C_SSR_GCF_MASK,
  kLPI2C_SlaveBusyFlag = LPI2C_SSR_SBF_MASK,
  kLPI2C_SlaveBusBusyFlag = LPI2C_SSR_BBF_MASK }
  *LPI2C slave peripheral flags.*
- enum lpi2c_slave_address_match_t {
  kLPI2C_MatchAddress0 = 0U,
  kLPI2C_MatchAddress0OrAddress1 = 2U,
  kLPI2C_MatchAddress0ThroughAddress1 = 6U }
  *LPI2C slave address match options.*
- enum lpi2c_slave_transfer_event_t {

kLPI2C_SlaveAddressMatchEvent = 0x01U,
kLPI2C_SlaveTransmitEvent = 0x02U,
kLPI2C_SlaveReceiveEvent = 0x04U,
kLPI2C_SlaveTransmitAckEvent = 0x08U,
kLPI2C_SlaveRepeatedStartEvent = 0x10U,
kLPI2C_SlaveCompletionEvent = 0x20U,
kLPI2C_SlaveAllEvents }
*Set of events sent to the callback for non blocking slave transfers.*

## Slave initialization and deinitialization

- void LPI2C_SlaveGetDefaultConfig (lpi2c_slave_config_t *slaveConfig)
  *Provides a default configuration for the LPI2C slave peripheral.*
- void LPI2C_SlaveInit (LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t sourceClock_Hz)
  *Initializes the LPI2C slave peripheral.*
- void LPI2C_SlaveDeinit (LPI2C_Type *base)
  *Deinitializes the LPI2C slave peripheral.*
- static void LPI2C_SlaveReset (LPI2C_Type *base)
  *Performs a software reset of the LPI2C slave peripheral.*
- static void LPI2C_SlaveEnable (LPI2C_Type *base, bool enable)
  *Enables or disables the LPI2C module as slave.*

## Slave status

- static uint32_t LPI2C_SlaveGetStatusFlags (LPI2C_Type *base)
  *Gets the LPI2C slave status flags.*
- static void LPI2C_SlaveClearStatusFlags (LPI2C_Type *base, uint32_t statusMask)
  *Clears the LPI2C status flag state.*

## Slave interrupts

- static void LPI2C_SlaveEnableInterrupts (LPI2C_Type *base, uint32_t interruptMask)
  *Enables the LPI2C slave interrupt requests.*
- static void LPI2C_SlaveDisableInterrupts (LPI2C_Type *base, uint32_t interruptMask)
  *Disables the LPI2C slave interrupt requests.*
- static uint32_t LPI2C_SlaveGetEnabledInterrupts (LPI2C_Type *base)
  *Returns the set of currently enabled LPI2C slave interrupt requests.*

## Slave DMA control

- static void LPI2C_SlaveEnableDMA (LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)
  *Enables or disables the LPI2C slave peripheral DMA requests.*

### Slave bus operations

- static bool LPI2C_SlaveGetBusIdleState (LPI2C_Type *base)
  *Returns whether the bus is idle.*
- static 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.*
- static uint32_t LPI2C_SlaveGetReceivedAddress (LPI2C_Type *base)
  *Returns the slave address sent by the I2C master.*
- status_t LPI2C_SlaveSend (LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)
  *Performs a polling send transfer on the I2C bus.*
- status_t LPI2C_SlaveReceive (LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)
  *Performs a polling receive transfer on the I2C bus.*

### Slave non-blocking

- 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.*
- status_t LPI2C_SlaveTransferNonBlocking (LPI2C_Type *base, lpi2c_slave_handle_t *handle, uint32_t eventMask)
  *Starts accepting slave transfers.*
- 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.*
- void LPI2C_SlaveTransferAbort (LPI2C_Type *base, lpi2c_slave_handle_t *handle)
  *Aborts the slave non-blocking transfers.*

### Slave IRQ handler

- void LPI2C_SlaveTransferHandleIRQ (LPI2C_Type *base, lpi2c_slave_handle_t *handle)
  *Reusable routine to handle slave interrupts.*

## 29.5.2 Data Structure Documentation

### 29.5.2.1 struct lpi2c_slave_config_t

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.

### Data Fields

- bool enableSlave
  *Enable slave mode.*
- uint8_t address0
  *Slave's 7-bit address.*
- uint8_t address1
  *Alternate slave 7-bit address.*
- lpi2c_slave_address_match_t addressMatchMode
  *Address matching options.*
- bool filterDozeEnable
  *Enable digital glitch filter in doze mode.*
- bool filterEnable
  *Enable digital glitch filter.*
- bool enableGeneralCall
  *Enable general call address matching.*
- bool ignoreAck
  *Continue transfers after a NACK is detected.*
- bool enableReceivedAddressRead
  *Enable reading the address received address as the first byte of data.*
- uint32_t sdaGlitchFilterWidth_ns
  *Width in nanoseconds of the digital filter on the SDA signal.*
- uint32_t sclGlitchFilterWidth_ns
  *Width in nanoseconds of the digital filter on the SCL signal.*
- uint32_t dataValidDelay_ns
  *Width in nanoseconds of the data valid delay.*
- uint32_t clockHoldTime_ns
  *Width in nanoseconds of the clock hold time.*
- 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.*
- 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.*

#### 29.5.2.1.0.13 Field Documentation

##### 29.5.2.1.0.13.1 bool lpi2c_slave_config_t::enableSlave

##### 29.5.2.1.0.13.2 uint8_t lpi2c_slave_config_t::address0

##### 29.5.2.1.0.13.3 uint8_t lpi2c_slave_config_t::address1

##### 29.5.2.1.0.13.4 lpi2c_slave_address_match_t lpi2c_slave_config_t::addressMatchMode

##### 29.5.2.1.0.13.5 bool lpi2c_slave_config_t::filterDozeEnable

##### 29.5.2.1.0.13.6 bool lpi2c_slave_config_t::filterEnable

##### 29.5.2.1.0.13.7 bool lpi2c_slave_config_t::enableGeneralCall

##### 29.5.2.1.0.13.8 bool lpi2c_slave_config_t::enableAck

Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

##### 29.5.2.1.0.13.9 bool lpi2c_slave_config_t::enableTx

##### 29.5.2.1.0.13.10 bool lpi2c_slave_config_t::enableRx

##### 29.5.2.1.0.13.11 bool lpi2c_slave_config_t::enableAddress

##### 29.5.2.1.0.13.12 bool lpi2c_slave_config_t::ignoreAck

##### 29.5.2.1.0.13.13 bool lpi2c_slave_config_t::enableReceivedAddressRead

##### 29.5.2.1.0.13.14 uint32_t lpi2c_slave_config_t::sdaGlitchFilterWidth_ns

##### 29.5.2.1.0.13.15 uint32_t lpi2c_slave_config_t::sclGlitchFilterWidth_ns

##### 29.5.2.1.0.13.16 uint32_t lpi2c_slave_config_t::dataValidDelay_ns

##### 29.5.2.1.0.13.17 uint32_t lpi2c_slave_config_t::clockHoldTime_ns

### 29.5.2.2 struct lpi2c_slave_transfer_t

### Data Fields

- 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.*
- size_t transferredCount
  *Number of bytes actually transferred since start or last repeated start.*

#### 29.5.2.2.0.14 Field Documentation

##### 29.5.2.2.0.14.1 lpi2c_slave_transfer_event_t lpi2c_slave_transfer_t::event

##### 29.5.2.2.0.14.2 uint8_t lpi2c_slave_transfer_t::receivedAddress

##### 29.5.2.2.0.14.3 status_t lpi2c_slave_transfer_t::completionStatus

Only applies for kLPI2C_SlaveCompletionEvent.

##### 29.5.2.2.0.14.4 size_t lpi2c_slave_transfer_t::transferredCount

### 29.5.2.3 struct _lpi2c_slave_handle

Note

The contents of this structure are private and subject to change.

### Data Fields

- 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.*

#### 29.5.2.3.0.15 Field Documentation

##### 29.5.2.3.0.15.1 lpi2c_slave_transfer_t lpi2c_slave_handle_t::transfer

##### 29.5.2.3.0.15.2 bool lpi2c_slave_handle_t::isBusy

##### 29.5.2.3.0.15.3 bool lpi2c_slave_handle_t::wasTransmit

##### 29.5.2.3.0.15.4 uint32_t lpi2c_slave_handle_t::eventMask

##### 29.5.2.3.0.15.5 uint32_t lpi2c_slave_handle_t::transferredCount

##### 29.5.2.3.0.15.6 lpi2c_slave_transfer_callback_t lpi2c_slave_handle_t::callback

##### 29.5.2.3.0.15.7 void* lpi2c_slave_handle_t::userData

### 29.5.3 Typedef Documentation

#### 29.5.3.1 typedef void(* lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, lpi2c_slave_transfer_t *transfer, void *userData)

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.

Parameters

| | |
|---|---|
| *base* | Base address for the LPI2C instance on which the event occurred. |
| *transfer* | Pointer to transfer descriptor containing values passed to and/or from the callback. |
| *userData* | Arbitrary pointer-sized value passed from the application. |

### 29.5.4 Enumeration Type Documentation

#### 29.5.4.1 enum _lpi2c_slave_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.

Enumerator

***kLPI2C_SlaveTxReadyFlag*** Transmit data flag.
***kLPI2C_SlaveRxReadyFlag*** Receive data flag.
***kLPI2C_SlaveAddressValidFlag*** Address valid flag.
***kLPI2C_SlaveTransmitAckFlag*** Transmit ACK flag.
***kLPI2C_SlaveRepeatedStartDetectFlag*** Repeated start detect flag.
***kLPI2C_SlaveStopDetectFlag*** Stop detect flag.
***kLPI2C_SlaveBitErrFlag*** Bit error flag.
***kLPI2C_SlaveFifoErrFlag*** FIFO error flag.
***kLPI2C_SlaveAddressMatch0Flag*** Address match 0 flag.
***kLPI2C_SlaveAddressMatch1Flag*** Address match 1 flag.
***kLPI2C_SlaveGeneralCallFlag*** General call flag.
***kLPI2C_SlaveBusyFlag*** Master busy flag.
***kLPI2C_SlaveBusBusyFlag*** Bus busy flag.

#### 29.5.4.2 enum lpi2c_slave_address_match_t

Enumerator

***kLPI2C_MatchAddress0*** Match only address 0.

***kLPI2C_MatchAddress0OrAddress1*** Match either address 0 or address 1.
***kLPI2C_MatchAddress0ThroughAddress1*** Match a range of slave addresses from address 0 through address 1.

#### 29.5.4.3 enum lpi2c_slave_transfer_event_t

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.

Enumerator

***kLPI2C_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.
***kLPI2C_SlaveTransmitEvent*** Callback is requested to provide data to transmit (slave-transmitter role).
***kLPI2C_SlaveReceiveEvent*** Callback is requested to provide a buffer in which to place received data (slave-receiver role).
***kLPI2C_SlaveTransmitAckEvent*** Callback needs to either transmit an ACK or NACK.
***kLPI2C_SlaveRepeatedStartEvent*** A repeated start was detected.
***kLPI2C_SlaveCompletionEvent*** A stop was detected, completing the transfer.
***kLPI2C_SlaveAllEvents*** Bit mask of all available events.

### 29.5.5 Function Documentation

#### 29.5.5.1 void LPI2C_SlaveGetDefaultConfig ( lpi2c_slave_config_t * *slaveConfig* )

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

| | | |
|---|---|---|
| out | *slaveConfig* | User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t. |

#### 29.5.5.2 void LPI2C_SlaveInit ( LPI2C_Type * *base*, const lpi2c_slave_config_t * *slaveConfig*, uint32_t *sourceClock_Hz* )

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. |

#### 29.5.5.3 void LPI2C_SlaveDeinit ( LPI2C_Type * *base* )

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. |

#### 29.5.5.4 static void LPI2C_SlaveReset ( LPI2C_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |

#### 29.5.5.5 static void LPI2C_SlaveEnable ( LPI2C_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *enable* | Pass true to enable or false to disable the specified LPI2C as slave. |

#### 29.5.5.6 static uint32_t LPI2C_SlaveGetStatusFlags ( LPI2C_Type * *base* ) [inline], [static]

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.

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.

See Also

_lpi2c_slave_flags

#### 29.5.5.7 static void LPI2C_SlaveClearStatusFlags ( LPI2C_Type * *base*, uint32_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

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(). |

See Also

_lpi2c_slave_flags.

#### 29.5.5.8 static void LPI2C_SlaveEnableInterrupts ( LPI2C_Type * *base*, uint32_t *interruptMask* ) [inline], [static]

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. |

#### 29.5.5.9 static void LPI2C_SlaveDisableInterrupts ( LPI2C_Type * *base*, uint32_t *interruptMask* ) [inline], [static]

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. |

#### 29.5.5.10 static uint32_t LPI2C_SlaveGetEnabledInterrupts ( LPI2C_Type * *base* ) [inline], [static]

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.

#### 29.5.5.11 static void LPI2C_SlaveEnableDMA ( LPI2C_Type * *base*, bool *enableAddressValid*, bool *enableRx*, bool *enableTx* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *enableAddress-Valid* | 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. |

#### 29.5.5.12 static bool LPI2C_SlaveGetBusIdleState ( LPI2C_Type * *base* ) [inline], [static]

Requires the slave mode to be enabled.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |

Return values

| | |
|---|---|
| *true* | Bus is busy. |
| *false* | Bus is idle. |

#### 29.5.5.13 static void LPI2C_SlaveTransmitAck ( LPI2C_Type * *base*, bool *ackOrNack* ) [inline], [static]

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. |

#### 29.5.5.14 static uint32_t LPI2C_SlaveGetReceivedAddress ( LPI2C_Type * *base* ) [inline], [static]

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.

#### 29.5.5.15 status_t LPI2C_SlaveSend ( LPI2C_Type * *base*, void * *txBuff*, size_t *txSize*, size_t * *actualTxSize* )

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. |
| out | *actualTxSize* | |

Returns

Error or success status returned by API.

#### 29.5.5.16 status_t LPI2C_SlaveReceive ( LPI2C_Type * *base*, void * *rxBuff*, size_t *rxSize*, size_t * *actualRxSize* )

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. |
| out | *actualRxSize* | |

Returns

Error or success status returned by API.

### 29.5.5.17 void LPI2C_SlaveTransferCreateHandle ( LPI2C_Type * *base*, lpi2c_slave_handle_t * *handle*, lpi2c_slave_transfer_callback_t *callback*, void * *userData* )

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. |
| out | *handle* | 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. |

### 29.5.5.18 status_t LPI2C_SlaveTransferNonBlocking ( LPI2C_Type * *base*, lpi2c_slave_handle_t * *handle*, uint32_t *eventMask* )

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. |

#### 29.5.5.19 status_t LPI2C_SlaveTransferGetCount ( LPI2C_Type * *base*, lpi2c_slave_handle_t * *handle*, size_t * *count* )

Parameters

| | | |
|---|---|---|
| | *base* | The LPI2C peripheral base address. |
| | *handle* | Pointer to i2c_slave_handle_t structure. |
| out | *count* | 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* | |

#### 29.5.5.20 void LPI2C_SlaveTransferAbort ( LPI2C_Type * *base*, lpi2c_slave_handle_t * *handle* )

Note

This API could be called at any time to stop slave for handling the bus events.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *handle* | Pointer to #lpi2c_slave_handle_t structure which stores the transfer state. |

Return values

| | |
|---|---|
| *#kStatus_Success* | |
| *kStatus_LPI2C_Idle* | |

### 29.5.5.21 void LPI2C_SlaveTransferHandleIRQ ( LPI2C_Type * *base*, lpi2c_slave_handle_t * *handle* )

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. |

## 29.6 LPI2C Master DMA Driver

### 29.6.1 Overview

#### Data Structures

- struct lpi2c_master_edma_handle_t
  *Driver handle for master DMA APIs. More...*

#### Typedefs

- typedef void(∗ lpi2c_master_edma_transfer_callback_t )(LPI2C_Type ∗base, lpi2c_master_edma_handle_t ∗handle, status_t completionStatus, void ∗userData)
  *Master DMA completion callback function pointer type.*

#### Master DMA

- void LPI2C_MasterCreateEDMAHandle (LPI2C_Type ∗base, lpi2c_master_edma_handle_t ∗handle, edma_handle_t ∗rxDmaHandle, edma_handle_t ∗txDmaHandle, lpi2c_master_edma_transfer_callback_t callback, void ∗userData)
  *Create a new handle for the LPI2C master DMA APIs.*
- status_t LPI2C_MasterTransferEDMA (LPI2C_Type ∗base, lpi2c_master_edma_handle_t ∗handle, lpi2c_master_transfer_t ∗transfer)
  *Performs a non-blocking DMA-based transaction on the I2C bus.*
- status_t LPI2C_MasterTransferGetCountEDMA (LPI2C_Type ∗base, lpi2c_master_edma_handle_t ∗handle, size_t ∗count)
  *Returns number of bytes transferred so far.*
- status_t LPI2C_MasterTransferAbortEDMA (LPI2C_Type ∗base, lpi2c_master_edma_handle_t ∗handle)
  *Terminates a non-blocking LPI2C master transmission early.*

### 29.6.2 Data Structure Documentation

#### 29.6.2.1 struct _lpi2c_master_edma_handle

Note

The contents of this structure are private and subject to change.

#### Data Fields

- LPI2C_Type ∗ base
  *LPI2C base pointer.*
- bool isBusy

*Transfer state machine current state.*

- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- uint16_t commandBuffer [7]
  *LPI2C command sequence.*
- lpi2c_master_transfer_t transfer
  *Copy of the current transfer info.*
- lpi2c_master_edma_transfer_callback_t completionCallback
  *Callback function pointer.*
- void ∗ userData
  *Application data passed to callback.*
- edma_handle_t ∗ rx
  *Handle for receive DMA channel.*
- edma_handle_t ∗ tx
  *Handle for transmit DMA channel.*
- edma_tcd_t tcds [2]
  *Software TCD.*

#### 29.6.2.1.0.16 Field Documentation

##### 29.6.2.1.0.16.1 LPI2C_Type∗ lpi2c_master_edma_handle_t::base

##### 29.6.2.1.0.16.2 bool lpi2c_master_edma_handle_t::isBusy

##### 29.6.2.1.0.16.3 uint8_t lpi2c_master_edma_handle_t::nbytes

##### 29.6.2.1.0.16.4 uint16_t lpi2c_master_edma_handle_t::commandBuffer[7]

##### 29.6.2.1.0.16.5 lpi2c_master_transfer_t lpi2c_master_edma_handle_t::transfer

##### 29.6.2.1.0.16.6 lpi2c_master_edma_transfer_callback_t lpi2c_master_edma_handle_t::completionCallback

##### 29.6.2.1.0.16.7 void∗ lpi2c_master_edma_handle_t::userData

##### 29.6.2.1.0.16.8 edma_handle_t∗ lpi2c_master_edma_handle_t::rx

##### 29.6.2.1.0.16.9 edma_handle_t∗ lpi2c_master_edma_handle_t::tx

##### 29.6.2.1.0.16.10 edma_tcd_t lpi2c_master_edma_handle_t::tcds[2]

Two are allocated to provide enough room to align to 32-bytes.

### 29.6.3 Typedef Documentation

#### 29.6.3.1 typedef void(∗ lpi2c_master_edma_transfer_callback_t)(LPI2C_Type ∗base, lpi2c_master_edma_handle_t ∗handle, status_t completionStatus, void ∗userData)

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *handle* | Handle associated with the completed transfer. |
| *completion-Status* | Either #kStatus_Success or an error code describing how the transfer completed. |
| *userData* | Arbitrary pointer-sized value passed from the application. |

### 29.6.4 Function Documentation

#### 29.6.4.1 void LPI2C_MasterCreateEDMAHandle ( LPI2C_Type * *base*, lpi2c_master_-edma_handle_t * *handle*, edma_handle_t * *rxDmaHandle*, edma_handle_t * *txDmaHandle*, lpi2c_master_edma_transfer_callback_t *callback*, void * *userData* )

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

| | | |
|---|---|---|
| | *base* | The LPI2C peripheral base address. |
| out | *handle* | Pointer to the LPI2C master driver handle. |
| | *rxDmaHandle* | Handle for the eDMA receive channel. Created by the user prior to calling this function. |
| | *txDmaHandle* | Handle for the eDMA transmit channel. Created by the user prior to calling this function. |
| | *callback* | User provided pointer to the asynchronous callback function. |
| | *userData* | User provided pointer to the application callback data. |

#### 29.6.4.2 status_t LPI2C_MasterTransferEDMA ( LPI2C_Type * *base*, lpi2c_-master_edma_handle_t * *handle*, lpi2c_master_transfer_t * *transfer* )

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *handle* | Pointer to the LPI2C master driver handle. |
| *transfer* | The pointer to the transfer descriptor. |

Return values

| | |
|---|---|
| *#kStatus_Success* | The transaction was started successfully. |
| *kStatus_LPI2C_Busy* | Either another master is currently utilizing the bus, or another DMA transaction is already in progress. |

#### 29.6.4.3 status_t LPI2C_MasterTransferGetCountEDMA ( LPI2C_Type * *base*, lpi2c_master_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | | |
|---|---|---|
| | *base* | The LPI2C peripheral base address. |
| | *handle* | Pointer to the LPI2C master driver handle. |
| out | *count* | Number of bytes transferred so far by the non-blocking transaction. |

Return values

| | |
|---|---|
| *#kStatus_Success* | |
| *#kStatus_NoTransferInProgress* | There is not a DMA transaction currently in progress. |

#### 29.6.4.4 status_t LPI2C_MasterTransferAbortEDMA ( LPI2C_Type * *base*, lpi2c_master_edma_handle_t * *handle* )

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

| | |
|---|---|
| *base* | The LPI2C peripheral base address. |
| *handle* | Pointer to the LPI2C master driver handle. |

Return values

| | |
|---|---|
| *#kStatus_Success* | A transaction was successfully aborted. |
| *kStatus_LPI2C_Idle* | There is not a DMA transaction currently in progress. |

## 29.7 LPI2C FreeRTOS Driver

### 29.7.1 Overview

#### Driver version

- #define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))
  *LPI2C freertos driver version 2.1.5.*

#### LPI2C RTOS Operation

- status_t LPI2C_RTOS_Init (lpi2c_rtos_handle_t *handle, LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t srcClock_Hz)
  *Initializes LPI2C.*
- status_t LPI2C_RTOS_Deinit (lpi2c_rtos_handle_t *handle)
  *Deinitializes the LPI2C.*
- status_t LPI2C_RTOS_Transfer (lpi2c_rtos_handle_t *handle, lpi2c_master_transfer_t *transfer)
  *Performs I2C transfer.*

### 29.7.2 Macro Definition Documentation

#### 29.7.2.1 #define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))

### 29.7.3 Function Documentation

#### 29.7.3.1 status_t LPI2C_RTOS_Init ( lpi2c_rtos_handle_t * *handle*, LPI2C_Type * *base*, const lpi2c_master_config_t * *masterConfig*, uint32_t *srcClock_Hz* )

This function initializes the LPI2C module and related RTOS context.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPI2C handle, the pointer to an allocated space for RTOS context. |
| *base* | The pointer base address of the LPI2C instance to initialize. |
| *masterConfig* | Configuration structure to set-up LPI2C in master mode. |
| *srcClock_Hz* | Frequency of input clock of the LPI2C module. |

Returns

status of the operation.

#### 29.7.3.2 status_t LPI2C_RTOS_Deinit ( lpi2c_rtos_handle_t * *handle* )

This function deinitializes the LPI2C module and related RTOS context.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPI2C handle. |

#### 29.7.3.3 status_t LPI2C_RTOS_Transfer ( lpi2c_rtos_handle_t * *handle*, lpi2c_master_transfer_t * *transfer* )

This function performs an I2C transfer using LPI2C module according to data given in the transfer structure.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPI2C handle. |
| *transfer* | Structure specifying the transfer parameters. |

Returns

status of the operation.

# Chapter 30
# LPSPI: Low Power Serial Peripheral Interface

## 30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power Serial Peripheral Interface (LPSPI) module of MCUXpresso SDK devices.

## Modules

- LPSPI FreeRTOS Driver
- LPSPI Peripheral driver
- LPSPI eDMA Driver

## 30.2 LPSPI Peripheral driver

### 30.2.1 Overview

This section describes the programming interface of the LPSPI Peripheral driver. The LPSPI driver configures LPSPI module, provides the functional and transactional interfaces to build the LPSPI application.

### 30.2.2 Function groups

#### 30.2.2.1 LPSPI Initialization and De-initialization

This function group initializes the default configuration structure for master and slave, initializes the LPSPI master with a master configuration, initializes the LPSPI slave with a slave configuration, and de-initializes the LPSPI module.

#### 30.2.2.2 LPSPI Basic Operation

This function group enables/disables the LPSPI module both interrupt and DMA, gets the data register address for the DMA transfer, sets master and slave, starts and stops the transfer, and so on.

#### 30.2.2.3 LPSPI Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

#### 30.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

#### 30.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

### 30.2.3 Typical use case

#### 30.2.3.1 Master Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lpspi

#### 30.2.3.2 Slave Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lpspi

## Data Structures

- struct lpspi_master_config_t
  *LPSPI master configuration structure. More...*
- struct lpspi_slave_config_t
  *LPSPI slave configuration structure. More...*
- struct lpspi_transfer_t
  *LPSPI master/slave transfer structure. More...*
- struct lpspi_master_handle_t
  *LPSPI master transfer handle structure used for transactional API. More...*
- struct lpspi_slave_handle_t
  *LPSPI slave transfer handle structure used for transactional API. More...*

## Macros

- #define LPSPI_DUMMY_DATA (0x00U)
  *LPSPI dummy data if no Tx data.*
- #define LPSPI_MASTER_PCS_SHIFT (4U)
  *LPSPI master PCS shift macro , internal used.*
- #define LPSPI_MASTER_PCS_MASK (0xF0U)
  *LPSPI master PCS shift macro , internal used.*
- #define LPSPI_SLAVE_PCS_SHIFT (4U)
  *LPSPI slave PCS shift macro , internal used.*
- #define LPSPI_SLAVE_PCS_MASK (0xF0U)
  *LPSPI slave PCS shift macro , internal used.*

## 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.*
- 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.*

## Enumerations

- enum _lpspi_status {
  kStatus_LPSPI_Busy = MAKE_STATUS(kStatusGroup_LPSPI, 0),
  kStatus_LPSPI_Error = MAKE_STATUS(kStatusGroup_LPSPI, 1),
  kStatus_LPSPI_Idle = MAKE_STATUS(kStatusGroup_LPSPI, 2),

kStatus_LPSPI_OutOfRange = MAKE_STATUS(kStatusGroup_LPSPI, 3) }
*Status for the LPSPI driver.*

- enum _lpspi_flags {
  kLPSPI_TxDataRequestFlag = LPSPI_SR_TDF_MASK,
  kLPSPI_RxDataReadyFlag = LPSPI_SR_RDF_MASK,
  kLPSPI_WordCompleteFlag = LPSPI_SR_WCF_MASK,
  kLPSPI_FrameCompleteFlag = LPSPI_SR_FCF_MASK,
  kLPSPI_TransferCompleteFlag = LPSPI_SR_TCF_MASK,
  kLPSPI_TransmitErrorFlag = LPSPI_SR_TEF_MASK,
  kLPSPI_ReceiveErrorFlag = LPSPI_SR_REF_MASK,
  kLPSPI_DataMatchFlag = LPSPI_SR_DMF_MASK,
  kLPSPI_ModuleBusyFlag = LPSPI_SR_MBF_MASK,
  kLPSPI_AllStatusFlag }
  *LPSPI status flags in SPIx_SR register.*
- enum _lpspi_interrupt_enable {
  kLPSPI_TxInterruptEnable = LPSPI_IER_TDIE_MASK,
  kLPSPI_RxInterruptEnable = LPSPI_IER_RDIE_MASK,
  kLPSPI_WordCompleteInterruptEnable = LPSPI_IER_WCIE_MASK,
  kLPSPI_FrameCompleteInterruptEnable = LPSPI_IER_FCIE_MASK,
  kLPSPI_TransferCompleteInterruptEnable = LPSPI_IER_TCIE_MASK,
  kLPSPI_TransmitErrorInterruptEnable = LPSPI_IER_TEIE_MASK,
  kLPSPI_ReceiveErrorInterruptEnable = LPSPI_IER_REIE_MASK,
  kLPSPI_DataMatchInterruptEnable = LPSPI_IER_DMIE_MASK,
  kLPSPI_AllInterruptEnable }
  *LPSPI interrupt source.*
- enum _lpspi_dma_enable {
  kLPSPI_TxDmaEnable = LPSPI_DER_TDDE_MASK,
  kLPSPI_RxDmaEnable = LPSPI_DER_RDDE_MASK }
  *LPSPI DMA source.*
- enum lpspi_master_slave_mode_t {
  kLPSPI_Master = 1U,
  kLPSPI_Slave = 0U }
  *LPSPI master or slave mode configuration.*
- enum lpspi_which_pcs_t {
  kLPSPI_Pcs0 = 0U,
  kLPSPI_Pcs1 = 1U,
  kLPSPI_Pcs2 = 2U,
  kLPSPI_Pcs3 = 3U }
  *LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).*
- enum lpspi_pcs_polarity_config_t {
  kLPSPI_PcsActiveHigh = 1U,
  kLPSPI_PcsActiveLow = 0U }
  *LPSPI Peripheral Chip Select (PCS) Polarity configuration.*
- enum _lpspi_pcs_polarity {

kLPSPI_Pcs0ActiveLow = 1U << 0,
kLPSPI_Pcs1ActiveLow = 1U << 1,
kLPSPI_Pcs2ActiveLow = 1U << 2,
kLPSPI_Pcs3ActiveLow = 1U << 3,
kLPSPI_PcsAllActiveLow = 0xFU }
*LPSPI Peripheral Chip Select (PCS) Polarity.*

- enum lpspi_clock_polarity_t {
  kLPSPI_ClockPolarityActiveHigh = 0U,
  kLPSPI_ClockPolarityActiveLow = 1U }
  *LPSPI clock polarity configuration.*
- enum lpspi_clock_phase_t {
  kLPSPI_ClockPhaseFirstEdge = 0U,
  kLPSPI_ClockPhaseSecondEdge = 1U }
  *LPSPI clock phase configuration.*
- enum lpspi_shift_direction_t {
  kLPSPI_MsbFirst = 0U,
  kLPSPI_LsbFirst = 1U }
  *LPSPI data shifter direction options.*
- enum lpspi_host_request_select_t {
  kLPSPI_HostReqExtPin = 0U,
  kLPSPI_HostReqInternalTrigger = 1U }
  *LPSPI Host Request select configuration.*
- enum lpspi_match_config_t {
  kLPSI_MatchDisabled = 0x0U,
  kLPSI_1stWordEqualsM0orM1 = 0x2U,
  kLPSI_AnyWordEqualsM0orM1 = 0x3U,
  kLPSI_1stWordEqualsM0and2ndWordEqualsM1 = 0x4U,
  kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 = 0x5U,
  kLPSI_1stWordAndM1EqualsM0andM1 = 0x6U,
  kLPSI_AnyWordAndM1EqualsM0andM1 = 0x7U }
  *LPSPI Match configuration options.*
- enum lpspi_pin_config_t {
  kLPSPI_SdiInSdoOut = 0U,
  kLPSPI_SdiInSdiOut = 1U,
  kLPSPI_SdoInSdoOut = 2U,
  kLPSPI_SdoInSdiOut = 3U }
  *LPSPI pin (SDO and SDI) configuration.*
- enum lpspi_data_out_config_t {
  kLpspiDataOutRetained = 0U,
  kLpspiDataOutTristate = 1U }
  *LPSPI data output configuration.*
- enum lpspi_transfer_width_t {
  kLPSPI_SingleBitXfer = 0U,
  kLPSPI_TwoBitXfer = 1U,
  kLPSPI_FourBitXfer = 2U }
  *LPSPI transfer width configuration.*
- enum lpspi_delay_type_t {

kLPSPI_PcsToSck = 1U,
kLPSPI_LastSckToPcs,
kLPSPI_BetweenTransfer }
*LPSPI delay type selection.*

- enum _lpspi_transfer_config_flag_for_master {
kLPSPI_MasterPcs0 = 0U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs1 = 1U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs2 = 2U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs3 = 3U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcsContinuous = 1U << 20,
kLPSPI_MasterByteSwap }
*Use this enumeration for LPSPI master transfer configFlags.*
- enum _lpspi_transfer_config_flag_for_slave {
kLPSPI_SlavePcs0 = 0U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs1 = 1U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs2 = 2U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs3 = 3U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlaveByteSwap }
*Use this enumeration for LPSPI slave transfer configFlags.*
- enum _lpspi_transfer_state {
kLPSPI_Idle = 0x0U,
kLPSPI_Busy,
kLPSPI_Error }
*LPSPI transfer state, which is used for LPSPI transactional API state machine.*

## Variables

- volatile uint8_t g_lpspiDummyData [ ]
*Global variable for dummy data value setting.*

## Driver version

- #define FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
*LPSPI driver version 2.0.3.*

## Initialization and deinitialization

- void LPSPI_MasterInit (LPSPI_Type *base, const lpspi_master_config_t *masterConfig, uint32_t srcClock_Hz)
*Initializes the LPSPI master.*
- void LPSPI_MasterGetDefaultConfig (lpspi_master_config_t *masterConfig)
*Sets the lpspi_master_config_t structure to default values.*
- void LPSPI_SlaveInit (LPSPI_Type *base, const lpspi_slave_config_t *slaveConfig)
*LPSPI slave configuration.*

- void LPSPI_SlaveGetDefaultConfig (lpspi_slave_config_t *slaveConfig)
  *Sets the lpspi_slave_config_t structure to default values.*
- void LPSPI_Deinit (LPSPI_Type *base)
  *De-initializes the LPSPI peripheral.*
- void LPSPI_Reset (LPSPI_Type *base)
  *Restores the LPSPI peripheral to reset state.*
- static void LPSPI_Enable (LPSPI_Type *base, bool enable)
  *Enables the LPSPI peripheral and sets the MCR MDIS to 0.*

### Status

- static uint32_t LPSPI_GetStatusFlags (LPSPI_Type *base)
  *Gets the LPSPI status flag state.*
- static uint32_t LPSPI_GetTxFifoSize (LPSPI_Type *base)
  *Gets the LPSPI Tx FIFO size.*
- static uint32_t LPSPI_GetRxFifoSize (LPSPI_Type *base)
  *Gets the LPSPI Rx FIFO size.*
- static uint32_t LPSPI_GetTxFifoCount (LPSPI_Type *base)
  *Gets the LPSPI Tx FIFO count.*
- static uint32_t LPSPI_GetRxFifoCount (LPSPI_Type *base)
  *Gets the LPSPI Rx FIFO count.*
- static void LPSPI_ClearStatusFlags (LPSPI_Type *base, uint32_t statusFlags)
  *Clears the LPSPI status flag.*

### Interrupts

- static void LPSPI_EnableInterrupts (LPSPI_Type *base, uint32_t mask)
  *Enables the LPSPI interrupts.*
- static void LPSPI_DisableInterrupts (LPSPI_Type *base, uint32_t mask)
  *Disables the LPSPI interrupts.*

### DMA Control

- static void LPSPI_EnableDMA (LPSPI_Type *base, uint32_t mask)
  *Enables the LPSPI DMA request.*
- static void LPSPI_DisableDMA (LPSPI_Type *base, uint32_t mask)
  *Disables the LPSPI DMA request.*
- static uint32_t LPSPI_GetTxRegisterAddress (LPSPI_Type *base)
  *Gets the LPSPI Transmit Data Register address for a DMA operation.*
- static uint32_t LPSPI_GetRxRegisterAddress (LPSPI_Type *base)
  *Gets the LPSPI Receive Data Register address for a DMA operation.*

### Bus Operations

- bool LPSPI_CheckTransferArgument (lpspi_transfer_t *transfer, uint32_t bitsPerFrame, uint32_t bytesPerFrame)

*Check the argument for transfer .*
- static void LPSPI_SetMasterSlaveMode (LPSPI_Type *base, lpspi_master_slave_mode_t mode)
  *Configures the LPSPI for either master or slave.*
- static bool LPSPI_IsMaster (LPSPI_Type *base)
  *Returns whether the LPSPI module is in master mode.*
- static void LPSPI_FlushFifo (LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)
  *Flushes the LPSPI FIFOs.*
- static void LPSPI_SetFifoWatermarks (LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)
  *Sets the transmit and receive FIFO watermark values.*
- static void LPSPI_SetAllPcsPolarity (LPSPI_Type *base, uint32_t mask)
  *Configures all LPSPI peripheral chip select polarities simultaneously.*
- static void LPSPI_SetFrameSize (LPSPI_Type *base, uint32_t frameSize)
  *Configures the frame size.*
- 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.*
- 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).*
- 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).*
- static void LPSPI_WriteData (LPSPI_Type *base, uint32_t data)
  *Writes data into the transmit data buffer.*
- static uint32_t LPSPI_ReadData (LPSPI_Type *base)
  *Reads data from the data buffer.*
- void LPSPI_SetDummyData (LPSPI_Type *base, uint8_t dummyData)
  *Set up the dummy data.*

## Transactional

- void LPSPI_MasterTransferCreateHandle (LPSPI_Type *base, lpspi_master_handle_t *handle, lpspi_master_transfer_callback_t callback, void *userData)
  *Initializes the LPSPI master handle.*
- status_t LPSPI_MasterTransferBlocking (LPSPI_Type *base, lpspi_transfer_t *transfer)
  *LPSPI master transfer data using a polling method.*
- status_t LPSPI_MasterTransferNonBlocking (LPSPI_Type *base, lpspi_master_handle_t *handle, lpspi_transfer_t *transfer)
  *LPSPI master transfer data using an interrupt method.*
- status_t LPSPI_MasterTransferGetCount (LPSPI_Type *base, lpspi_master_handle_t *handle, size_t *count)
  *Gets the master transfer remaining bytes.*
- void LPSPI_MasterTransferAbort (LPSPI_Type *base, lpspi_master_handle_t *handle)
  *LPSPI master abort transfer which uses an interrupt method.*
- void LPSPI_MasterTransferHandleIRQ (LPSPI_Type *base, lpspi_master_handle_t *handle)
  *LPSPI Master IRQ handler function.*
- void LPSPI_SlaveTransferCreateHandle (LPSPI_Type *base, lpspi_slave_handle_t *handle, lpspi-

_slave_transfer_callback_t callback, void ∗userData)
*Initializes the LPSPI slave handle.*
- status_t LPSPI_SlaveTransferNonBlocking (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle, lpspi_transfer_t ∗transfer)
  *LPSPI slave transfer data using an interrupt method.*
- status_t LPSPI_SlaveTransferGetCount (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle, size_t ∗count)
  *Gets the slave transfer remaining bytes.*
- void LPSPI_SlaveTransferAbort (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle)
  *LPSPI slave aborts a transfer which uses an interrupt method.*
- void LPSPI_SlaveTransferHandleIRQ (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle)
  *LPSPI Slave IRQ handler function.*

### 30.2.4 Data Structure Documentation

#### 30.2.4.1 struct lpspi_master_config_t

#### Data Fields

- 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.*
- uint32_t lastSckToPcsDelayInNanoSec
  *Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay.*
- uint32_t betweenTransferDelayInNanoSec
  `After the SCK delay time with nanoseconds, setting to 0 sets the`
  *minimum delay.*
- 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).*

#### 30.2.4.1.0.17 Field Documentation

##### 30.2.4.1.0.17.1 uint32_t lpspi_master_config_t::baudRate

##### 30.2.4.1.0.17.2 uint32_t lpspi_master_config_t::bitsPerFrame

##### 30.2.4.1.0.17.3 lpspi_clock_polarity_t lpspi_master_config_t::cpol

##### 30.2.4.1.0.17.4 lpspi_clock_phase_t lpspi_master_config_t::cpha

##### 30.2.4.1.0.17.5 lpspi_shift_direction_t lpspi_master_config_t::direction

##### 30.2.4.1.0.17.6 uint32_t lpspi_master_config_t::pcsToSckDelayInNanoSec

It sets the boundary value if out of range.

##### 30.2.4.1.0.17.7 uint32_t lpspi_master_config_t::lastSckToPcsDelayInNanoSec

It sets the boundary value if out of range.

##### 30.2.4.1.0.17.8 uint32_t lpspi_master_config_t::betweenTransferDelayInNanoSec

It sets the boundary value if out of range.

##### 30.2.4.1.0.17.9 lpspi_which_pcs_t lpspi_master_config_t::whichPcs

##### 30.2.4.1.0.17.10 lpspi_pin_config_t lpspi_master_config_t::pinCfg

##### 30.2.4.1.0.17.11 lpspi_data_out_config_t lpspi_master_config_t::dataOutConfig

### 30.2.4.2 struct lpspi_slave_config_t

### Data Fields

- 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).*

#### 30.2.4.2.0.18 Field Documentation

##### 30.2.4.2.0.18.1 uint32_t lpspi_slave_config_t::bitsPerFrame

##### 30.2.4.2.0.18.2 lpspi_clock_polarity_t lpspi_slave_config_t::cpol

##### 30.2.4.2.0.18.3 lpspi_clock_phase_t lpspi_slave_config_t::cpha

##### 30.2.4.2.0.18.4 lpspi_shift_direction_t lpspi_slave_config_t::direction

##### 30.2.4.2.0.18.5 lpspi_pin_config_t lpspi_slave_config_t::pinCfg

##### 30.2.4.2.0.18.6 lpspi_data_out_config_t lpspi_slave_config_t::dataOutConfig

### 30.2.4.3 struct lpspi_transfer_t

#### Data Fields

- uint8_t * txData
  *Send buffer.*
- uint8_t * rxData
  *Receive buffer.*
- volatile size_t dataSize
  *Transfer bytes.*
- uint32_t configFlags
  *Transfer transfer configuration flags.*

#### 30.2.4.3.0.19 Field Documentation

##### 30.2.4.3.0.19.1 uint8_t* lpspi_transfer_t::txData

##### 30.2.4.3.0.19.2 uint8_t* lpspi_transfer_t::rxData

##### 30.2.4.3.0.19.3 volatile size_t lpspi_transfer_t::dataSize

##### 30.2.4.3.0.19.4 uint32_t lpspi_transfer_t::configFlags

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.

### 30.2.4.4 struct _lpspi_master_handle

Forward declaration of the _lpspi_master_handle typedefs.

#### Data Fields

- 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 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.*
- 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.*

#### 30.2.4.4.0.20 Field Documentation

##### 30.2.4.4.0.20.1 volatile bool lpspi_master_handle_t::isPcsContinuous

##### 30.2.4.4.0.20.2 volatile bool lpspi_master_handle_t::writeTcrInIsr

##### 30.2.4.4.0.20.3 volatile bool lpspi_master_handle_t::isByteSwap

##### 30.2.4.4.0.20.4 volatile uint8_t lpspi_master_handle_t::fifoSize

##### 30.2.4.4.0.20.5 volatile uint8_t lpspi_master_handle_t::rxWatermark

##### 30.2.4.4.0.20.6 volatile uint8_t lpspi_master_handle_t::bytesEachWrite

##### 30.2.4.4.0.20.7 volatile uint8_t lpspi_master_handle_t::bytesEachRead

##### 30.2.4.4.0.20.8 uint8_t* volatile lpspi_master_handle_t::txData

##### 30.2.4.4.0.20.9 uint8_t* volatile lpspi_master_handle_t::rxData

##### 30.2.4.4.0.20.10 volatile size_t lpspi_master_handle_t::txRemainingByteCount

##### 30.2.4.4.0.20.11 volatile size_t lpspi_master_handle_t::rxRemainingByteCount

##### 30.2.4.4.0.20.12 volatile uint32_t lpspi_master_handle_t::writeRegRemainingTimes

##### 30.2.4.4.0.20.13 volatile uint32_t lpspi_master_handle_t::readRegRemainingTimes

##### 30.2.4.4.0.20.14 uint32_t lpspi_master_handle_t::txBuffIfNull

##### 30.2.4.4.0.20.15 volatile uint8_t lpspi_master_handle_t::state

##### 30.2.4.4.0.20.16 lpspi_master_transfer_callback_t lpspi_master_handle_t::callback

##### 30.2.4.4.0.20.17 void* lpspi_master_handle_t::userData

### 30.2.4.5 struct _lpspi_slave_handle

Forward declaration of the _lpspi_slave_handle typedefs.

### Data Fields

- 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.*
- 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.*

#### 30.2.4.5.0.21 Field Documentation

##### 30.2.4.5.0.21.1 volatile bool lpspi_slave_handle_t::isByteSwap

##### 30.2.4.5.0.21.2 volatile uint8_t lpspi_slave_handle_t::fifoSize

##### 30.2.4.5.0.21.3 volatile uint8_t lpspi_slave_handle_t::rxWatermark

##### 30.2.4.5.0.21.4 volatile uint8_t lpspi_slave_handle_t::bytesEachWrite

##### 30.2.4.5.0.21.5 volatile uint8_t lpspi_slave_handle_t::bytesEachRead

##### 30.2.4.5.0.21.6 uint8_t* volatile lpspi_slave_handle_t::txData

##### 30.2.4.5.0.21.7 uint8_t* volatile lpspi_slave_handle_t::rxData

##### 30.2.4.5.0.21.8 volatile size_t lpspi_slave_handle_t::txRemainingByteCount

##### 30.2.4.5.0.21.9 volatile size_t lpspi_slave_handle_t::rxRemainingByteCount

##### 30.2.4.5.0.21.10 volatile uint32_t lpspi_slave_handle_t::writeRegRemainingTimes

##### 30.2.4.5.0.21.11 volatile uint32_t lpspi_slave_handle_t::readRegRemainingTimes

##### 30.2.4.5.0.21.12 volatile uint8_t lpspi_slave_handle_t::state

##### 30.2.4.5.0.21.13 volatile uint32_t lpspi_slave_handle_t::errorCount

##### 30.2.4.5.0.21.14 lpspi_slave_transfer_callback_t lpspi_slave_handle_t::callback

##### 30.2.4.5.0.21.15 void* lpspi_slave_handle_t::userData

### 30.2.5 Macro Definition Documentation

#### 30.2.5.1 #define FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

#### 30.2.5.2 #define LPSPI_DUMMY_DATA (0x00U)

Dummy data used for tx if there is not txData.

#### 30.2.5.3 #define LPSPI_MASTER_PCS_SHIFT (4U)

#### 30.2.5.4 #define LPSPI_MASTER_PCS_MASK (0xF0U)

#### 30.2.5.5 #define LPSPI_SLAVE_PCS_SHIFT (4U)

#### 30.2.5.6 #define LPSPI_SLAVE_PCS_MASK (0xF0U)

### 30.2.6 Typedef Documentation

#### 30.2.6.1 typedef void(∗ lpspi_master_transfer_callback_t)(LPSPI_Type ∗base, lpspi_master_handle_t ∗handle, status_t status, void ∗userData)

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |
| *handle* | Pointer to the handle for the LPSPI master. |
| *status* | Success or error code describing whether the transfer is completed. |
| *userData* | Arbitrary pointer-dataSized value passed from the application. |

#### 30.2.6.2 typedef void(∗ lpspi_slave_transfer_callback_t)(LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle, status_t status, void ∗userData)

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |
| *handle* | Pointer to the handle for the LPSPI slave. |
| *status* | Success or error code describing whether the transfer is completed. |
| *userData* | Arbitrary pointer-dataSized value passed from the application. |

### 30.2.7 Enumeration Type Documentation

#### 30.2.7.1 enum _lpspi_status

Enumerator

***kStatus_LPSPI_Busy*** LPSPI transfer is busy.
***kStatus_LPSPI_Error*** LPSPI driver error.
***kStatus_LPSPI_Idle*** LPSPI is idle.
***kStatus_LPSPI_OutOfRange*** LPSPI transfer out Of range.

#### 30.2.7.2 enum _lpspi_flags

Enumerator

***kLPSPI_TxDataRequestFlag*** Transmit data flag.
***kLPSPI_RxDataReadyFlag*** Receive data flag.
***kLPSPI_WordCompleteFlag*** Word Complete flag.
***kLPSPI_FrameCompleteFlag*** Frame Complete flag.
***kLPSPI_TransferCompleteFlag*** Transfer Complete flag.
***kLPSPI_TransmitErrorFlag*** Transmit Error flag (FIFO underrun)
***kLPSPI_ReceiveErrorFlag*** Receive Error flag (FIFO overrun)
***kLPSPI_DataMatchFlag*** Data Match flag.

***kLPSPI_ModuleBusyFlag*** Module Busy flag.
***kLPSPI_AllStatusFlag*** Used for clearing all w1c status flags.

#### 30.2.7.3 enum _lpspi_interrupt_enable

Enumerator

***kLPSPI_TxInterruptEnable*** Transmit data interrupt enable.
***kLPSPI_RxInterruptEnable*** Receive data interrupt enable.
***kLPSPI_WordCompleteInterruptEnable*** Word complete interrupt enable.
***kLPSPI_FrameCompleteInterruptEnable*** Frame complete interrupt enable.
***kLPSPI_TransferCompleteInterruptEnable*** Transfer complete interrupt enable.
***kLPSPI_TransmitErrorInterruptEnable*** Transmit error interrupt enable(FIFO underrun)
***kLPSPI_ReceiveErrorInterruptEnable*** Receive Error interrupt enable (FIFO overrun)
***kLPSPI_DataMatchInterruptEnable*** Data Match interrupt enable.
***kLPSPI_AllInterruptEnable*** All above interrupts enable.

#### 30.2.7.4 enum _lpspi_dma_enable

Enumerator

***kLPSPI_TxDmaEnable*** Transmit data DMA enable.
***kLPSPI_RxDmaEnable*** Receive data DMA enable.

#### 30.2.7.5 enum lpspi_master_slave_mode_t

Enumerator

***kLPSPI_Master*** LPSPI peripheral operates in master mode.
***kLPSPI_Slave*** LPSPI peripheral operates in slave mode.

#### 30.2.7.6 enum lpspi_which_pcs_t

Enumerator

***kLPSPI_Pcs0*** PCS[0].
***kLPSPI_Pcs1*** PCS[1].
***kLPSPI_Pcs2*** PCS[2].
***kLPSPI_Pcs3*** PCS[3].

#### 30.2.7.7 enum lpspi_pcs_polarity_config_t

Enumerator

***kLPSPI_PcsActiveHigh*** PCS Active High (idles low)
***kLPSPI_PcsActiveLow*** PCS Active Low (idles high)

#### 30.2.7.8 enum _lpspi_pcs_polarity

Enumerator

***kLPSPI_Pcs0ActiveLow*** Pcs0 Active Low (idles high).
***kLPSPI_Pcs1ActiveLow*** Pcs1 Active Low (idles high).
***kLPSPI_Pcs2ActiveLow*** Pcs2 Active Low (idles high).
***kLPSPI_Pcs3ActiveLow*** Pcs3 Active Low (idles high).
***kLPSPI_PcsAllActiveLow*** Pcs0 to Pcs5 Active Low (idles high).

#### 30.2.7.9 enum lpspi_clock_polarity_t

Enumerator

***kLPSPI_ClockPolarityActiveHigh*** CPOL=0. Active-high LPSPI clock (idles low)
***kLPSPI_ClockPolarityActiveLow*** CPOL=1. Active-low LPSPI clock (idles high)

#### 30.2.7.10 enum lpspi_clock_phase_t

Enumerator

***kLPSPI_ClockPhaseFirstEdge*** CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.
***kLPSPI_ClockPhaseSecondEdge*** CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

#### 30.2.7.11 enum lpspi_shift_direction_t

Enumerator

***kLPSPI_MsbFirst*** Data transfers start with most significant bit.
***kLPSPI_LsbFirst*** Data transfers start with least significant bit.

#### 30.2.7.12 enum lpspi_host_request_select_t

Enumerator

***kLPSPI_HostReqExtPin*** Host Request is an ext pin.
***kLPSPI_HostReqInternalTrigger*** Host Request is an internal trigger.

#### 30.2.7.13 enum lpspi_match_config_t

Enumerator

***kLPSI_MatchDisabled*** LPSPI Match Disabled.
***kLPSI_1stWordEqualsM0orM1*** LPSPI Match Enabled.
***kLPSI_AnyWordEqualsM0orM1*** LPSPI Match Enabled.
***kLPSI_1stWordEqualsM0and2ndWordEqualsM1*** LPSPI Match Enabled.
***kLPSI_AnyWordEqualsM0andNxtWordEqualsM1*** LPSPI Match Enabled.
***kLPSI_1stWordAndM1EqualsM0andM1*** LPSPI Match Enabled.
***kLPSI_AnyWordAndM1EqualsM0andM1*** LPSPI Match Enabled.

#### 30.2.7.14 enum lpspi_pin_config_t

Enumerator

***kLPSPI_SdiInSdoOut*** LPSPI SDI input, SDO output.
***kLPSPI_SdiInSdiOut*** LPSPI SDI input, SDI output.
***kLPSPI_SdoInSdoOut*** LPSPI SDO input, SDO output.
***kLPSPI_SdoInSdiOut*** LPSPI SDO input, SDI output.

#### 30.2.7.15 enum lpspi_data_out_config_t

Enumerator

***kLpspiDataOutRetained*** Data out retains last value when chip select is de-asserted.
***kLpspiDataOutTristate*** Data out is tristated when chip select is de-asserted.

#### 30.2.7.16 enum lpspi_transfer_width_t

Enumerator

***kLPSPI_SingleBitXfer*** 1-bit shift at a time, data out on SDO, in on SDI (normal mode)
***kLPSPI_TwoBitXfer*** 2-bits shift out on SDO/SDI and in on SDO/SDI
***kLPSPI_FourBitXfer*** 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

#### 30.2.7.17 enum lpspi_delay_type_t

Enumerator

***kLPSPI_PcsToSck*** PCS-to-SCK delay.
***kLPSPI_LastSckToPcs*** Last SCK edge to PCS delay.
***kLPSPI_BetweenTransfer*** Delay between transfers.

#### 30.2.7.18 enum _lpspi_transfer_config_flag_for_master

Enumerator

***kLPSPI_MasterPcs0*** LPSPI master transfer use PCS0 signal.
***kLPSPI_MasterPcs1*** LPSPI master transfer use PCS1 signal.
***kLPSPI_MasterPcs2*** LPSPI master transfer use PCS2 signal.
***kLPSPI_MasterPcs3*** LPSPI master transfer use PCS3 signal.
***kLPSPI_MasterPcsContinuous*** Is PCS signal continuous.
***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).

1. 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.
2. 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.
3. 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.

#### 30.2.7.19 enum _lpspi_transfer_config_flag_for_slave

Enumerator

***kLPSPI_SlavePcs0*** LPSPI slave transfer use PCS0 signal.
***kLPSPI_SlavePcs1*** LPSPI slave transfer use PCS1 signal.
***kLPSPI_SlavePcs2*** LPSPI slave transfer use PCS2 signal.
***kLPSPI_SlavePcs3*** LPSPI slave transfer use PCS3 signal.
***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).

1. 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.
2. 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.

3. 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.

#### 30.2.7.20 enum _lpspi_transfer_state

Enumerator

***kLPSPI_Idle*** Nothing in the transmitter/receiver.
***kLPSPI_Busy*** Transfer queue is not finished.
***kLPSPI_Error*** Transfer error.

### 30.2.8 Function Documentation

#### 30.2.8.1 void LPSPI_MasterInit ( LPSPI_Type * *base*, const lpspi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz* )

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |
| *masterConfig* | Pointer to structure lpspi_master_config_t. |
| *srcClock_Hz* | Module source input clock in Hertz |

#### 30.2.8.2 void LPSPI_MasterGetDefaultConfig ( lpspi_master_config_t * *masterConfig* )

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 |

#### 30.2.8.3 void LPSPI_SlaveInit ( LPSPI_Type * *base*, const lpspi_slave_config_t * *slaveConfig* )

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |
| *slaveConfig* | Pointer to a structure lpspi_slave_config_t. |

#### 30.2.8.4 void LPSPI_SlaveGetDefaultConfig ( lpspi_slave_config_t * *slaveConfig* )

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. |

#### 30.2.8.5 void LPSPI_Deinit ( LPSPI_Type * *base* )

Call this API to disable the LPSPI clock.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |

#### 30.2.8.6 void LPSPI_Reset ( LPSPI_Type * *base* )

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. |

#### 30.2.8.7 static void LPSPI_Enable ( LPSPI_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |
| *enable* | Pass true to enable module, false to disable module. |

#### 30.2.8.8 static uint32_t LPSPI_GetStatusFlags ( LPSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |

Returns

The LPSPI status(in SR register).

#### 30.2.8.9 static uint32_t LPSPI_GetTxFifoSize ( LPSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |

Returns

The LPSPI Tx FIFO size.

#### 30.2.8.10 static uint32_t LPSPI_GetRxFifoSize ( LPSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |

Returns

The LPSPI Rx FIFO size.

#### 30.2.8.11 static uint32_t LPSPI_GetTxFifoCount ( LPSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |

Returns

The number of words in the transmit FIFO.

#### 30.2.8.12 static uint32_t LPSPI_GetRxFifoCount ( LPSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |

Returns

The number of words in the receive FIFO.

#### 30.2.8.13 static void LPSPI_ClearStatusFlags ( LPSPI_Type * *base*, uint32_t *statusFlags* ) [inline], [static]

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. |

< The status flags are cleared by writing 1 (w1c).

#### 30.2.8.14 static void LPSPI_EnableInterrupts ( LPSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

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. |

### 30.2.8.15 static void LPSPI_DisableInterrupts ( LPSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

```
*  LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable |
       kLPSPI_RxInterruptEnable );
*
```

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |
| *mask* | The interrupt mask; Use the enum _lpspi_interrupt_enable. |

### 30.2.8.16 static void LPSPI_EnableDMA ( LPSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

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. |

### 30.2.8.17 static void LPSPI_DisableDMA ( LPSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

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. |

#### 30.2.8.18 static uint32_t LPSPI_GetTxRegisterAddress ( LPSPI_Type * *base* ) [inline], [static]

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.

#### 30.2.8.19 static uint32_t LPSPI_GetRxRegisterAddress ( LPSPI_Type * *base* ) [inline], [static]

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.

#### 30.2.8.20 bool LPSPI_CheckTransferArgument ( lpspi_transfer_t * *transfer,* uint32_t *bitsPerFrame,* uint32_t *bytesPerFrame* )

Parameters

| | |
|---|---|
| *transfer* | the transfer struct to be used. |
| *bitPerFrame* | The bit size of one frame. |
| *bytePerFrame* | The byte size of one frame. |

Returns

Return true for right and false for wrong.

#### 30.2.8.21 static void LPSPI_SetMasterSlaveMode ( LPSPI_Type * *base*, lpspi_master_slave_mode_t *mode* ) [inline], [static]

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. |

#### 30.2.8.22 static bool LPSPI_IsMaster ( LPSPI_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral address. |

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

#### 30.2.8.23 static void LPSPI_FlushFifo ( LPSPI_Type * *base*, bool *flushTxFifo*, bool *flushRxFifo* ) [inline], [static]

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. |

#### 30.2.8.24 static void LPSPI_SetFifoWatermarks ( LPSPI_Type * *base*, uint32_t *txWater*, uint32_t *rxWater* ) [inline], [static]

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. |

#### 30.2.8.25 static void LPSPI_SetAllPcsPolarity ( LPSPI_Type * *base*, uint32_t *mask* ) [inline], [static]

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. |

#### 30.2.8.26 static void LPSPI_SetFrameSize ( LPSPI_Type * *base*, uint32_t *frameSize* ) [inline], [static]

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. |

#### 30.2.8.27 uint32_t LPSPI_MasterSetBaudRate ( LPSPI_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*, uint32_t * *tcrPrescaleValue* )

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. |
| *tcrPrescale-Value* | 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.

#### 30.2.8.28 void LPSPI_MasterSetDelayScaler ( LPSPI_Type * *base*, uint32_t *scaler*, lpspi_delay_type_t *whichDelay* )

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. |

#### 30.2.8.29 uint32_t LPSPI_MasterSetDelayTimes ( LPSPI_Type * *base*, uint32_t *delayTimeInNanoSec*, lpspi_delay_type_t *whichDelay*, uint32_t *srcClock_Hz* )

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.

#### 30.2.8.30 static void LPSPI_WriteData ( LPSPI_Type * *base,* uint32_t *data* ) [inline], [static]

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. |

#### 30.2.8.31 static uint32_t LPSPI_ReadData ( LPSPI_Type * *base* ) [inline], [static]

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.

#### 30.2.8.32 void LPSPI_SetDummyData ( LPSPI_Type * *base,* uint8_t *dummyData* )

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. |

#### 30.2.8.33 void LPSPI_MasterTransferCreateHandle ( LPSPI_Type * *base*, lpspi_master_handle_t * *handle*, lpspi_master_transfer_callback_t *callback*, void * *userData* )

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. |

#### 30.2.8.34 status_t LPSPI_MasterTransferBlocking ( LPSPI_Type * *base*, lpspi_transfer_t * *transfer* )

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.

#### 30.2.8.35 status_t LPSPI_MasterTransferNonBlocking ( LPSPI_Type * *base*, lpspi_master_handle_t * *handle*, lpspi_transfer_t * *transfer* )

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.

#### 30.2.8.36 status_t LPSPI_MasterTransferGetCount ( LPSPI_Type * *base*, lpspi_master_handle_t * *handle*, size_t * *count* )

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.

#### 30.2.8.37 void LPSPI_MasterTransferAbort ( LPSPI_Type * *base*, lpspi_master_handle_t * *handle* )

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. |

#### 30.2.8.38 void LPSPI_MasterTransferHandleIRQ ( LPSPI_Type * *base*, lpspi_master_handle_t * *handle* )

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. |

#### 30.2.8.39 void LPSPI_SlaveTransferCreateHandle ( LPSPI_Type * *base*, lpspi_slave_handle_t * *handle*, lpspi_slave_transfer_callback_t *callback*, void * *userData* )

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. |

#### 30.2.8.40 status_t LPSPI_SlaveTransferNonBlocking ( LPSPI_Type * *base*, lpspi_slave_handle_t * *handle*, lpspi_transfer_t * *transfer* )

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.

#### 30.2.8.41 status_t LPSPI_SlaveTransferGetCount ( LPSPI_Type * *base*, lpspi_slave_handle_t * *handle*, size_t * *count* )

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.

#### 30.2.8.42 void LPSPI_SlaveTransferAbort ( LPSPI_Type * *base*, lpspi_slave_handle_t * *handle* )

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. |

#### 30.2.8.43 void LPSPI_SlaveTransferHandleIRQ ( LPSPI_Type * *base*, lpspi_slave_handle_t * *handle* )

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. |

### 30.2.9 Variable Documentation

#### 30.2.9.1 volatile uint8_t g_lpspiDummyData[]

## 30.3 LPSPI eDMA Driver

### 30.3.1 Overview

#### Data Structures

- struct lpspi_master_edma_handle_t
  *LPSPI master eDMA transfer handle structure used for transactional API. More...*
- struct lpspi_slave_edma_handle_t
  *LPSPI slave eDMA transfer handle structure used for transactional API. More...*

#### Typedefs

- typedef void(∗ lpspi_master_edma_transfer_callback_t )(LPSPI_Type ∗base, lpspi_master_edma_-handle_t ∗handle, status_t status, void ∗userData)
  *Completion callback function pointer type.*
- typedef void(∗ lpspi_slave_edma_transfer_callback_t )(LPSPI_Type ∗base, lpspi_slave_edma_-handle_t ∗handle, status_t status, void ∗userData)
  *Completion callback function pointer type.*

#### Functions

- void LPSPI_MasterTransferCreateHandleEDMA (LPSPI_Type ∗base, lpspi_master_edma_handle-_t ∗handle, lpspi_master_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗edmaRxRegToRxDataHandle, edma_handle_t ∗edmaTxDataToTxRegHandle)
  *Initializes the LPSPI master eDMA handle.*
- status_t LPSPI_MasterTransferEDMA (LPSPI_Type ∗base, lpspi_master_edma_handle_t ∗handle, lpspi_transfer_t ∗transfer)
  *LPSPI master transfer data using eDMA.*
- void LPSPI_MasterTransferAbortEDMA (LPSPI_Type ∗base, lpspi_master_edma_handle_-t ∗handle)
  *LPSPI master aborts a transfer which is using eDMA.*
- status_t LPSPI_MasterTransferGetCountEDMA (LPSPI_Type ∗base, lpspi_master_edma_handle_t ∗handle, size_t ∗count)
  *Gets the master eDMA transfer remaining bytes.*
- void LPSPI_SlaveTransferCreateHandleEDMA (LPSPI_Type ∗base, lpspi_slave_edma_handle_t ∗handle, lpspi_slave_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗edma-RxRegToRxDataHandle, edma_handle_t ∗edmaTxDataToTxRegHandle)
  *Initializes the LPSPI slave eDMA handle.*
- status_t LPSPI_SlaveTransferEDMA (LPSPI_Type ∗base, lpspi_slave_edma_handle_t ∗handle, lpspi_transfer_t ∗transfer)
  *LPSPI slave transfers data using eDMA.*
- void LPSPI_SlaveTransferAbortEDMA (LPSPI_Type ∗base, lpspi_slave_edma_handle_t ∗handle)
  *LPSPI slave aborts a transfer which is using eDMA.*
- status_t LPSPI_SlaveTransferGetCountEDMA (LPSPI_Type ∗base, lpspi_slave_edma_handle_-t ∗handle, size_t ∗count)

*Gets the slave eDMA transfer remaining bytes.*

### Driver version

- #define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *LPSPI EDMA driver version 2.0.2.*

## 30.3.2 Data Structure Documentation

### 30.3.2.1 struct _lpspi_master_edma_handle

Forward declaration of the _lpspi_master_edma_handle typedefs.

### Data Fields

- volatile bool isPcsContinuous
  *Is PCS continuous in transfer.*
- volatile bool isByteSwap
  *A flag that whether should byte swap.*
- volatile uint8_t fifoSize
  *FIFO dataSize.*
- volatile uint8_t rxWatermark
  *Rx watermark.*
- volatile uint8_t bytesEachWrite
  *Bytes for each write TDR.*
- volatile uint8_t bytesEachRead
  *Bytes for each read RDR.*
- volatile uint8_t bytesLastRead
  *Bytes for last read RDR.*
- volatile uint8_t isThereExtraRxBytes
  *Is there extra RX byte.*
- uint8_t *volatile txData
  *Send buffer.*
- uint8_t *volatile rxData
  *Receive buffer.*
- volatile size_t txRemainingByteCount
  *Number of bytes remaining to send.*
- volatile size_t rxRemainingByteCount
  *Number of bytes remaining to receive.*
- volatile uint32_t writeRegRemainingTimes
  *Write TDR register remaining times.*
- volatile uint32_t readRegRemainingTimes
  *Read RDR register remaining times.*
- uint32_t totalByteCount
  *Number of transfer bytes.*
- uint32_t txBuffIfNull
  *Used if there is not txData for DMA purpose.*

- uint32_t rxBuffIfNull
  *Used if there is not rxData for DMA purpose.*
- uint32_t transmitCommand
  *Used to write TCR for DMA purpose.*
- volatile uint8_t state
  *LPSPI transfer state , _lpspi_transfer_state.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- lpspi_master_edma_transfer_callback_t callback
  *Completion callback.*
- void * userData
  *Callback user data.*
- edma_handle_t * edmaRxRegToRxDataHandle
  *edma_handle_t handle point used for RxReg to RxData buff*
- edma_handle_t * edmaTxDataToTxRegHandle
  *edma_handle_t handle point used for TxData to TxReg buff*
- edma_tcd_t lpspiSoftwareTCD [3]
  *SoftwareTCD, internal used.*

#### 30.3.2.1.0.22 Field Documentation

##### 30.3.2.1.0.22.1 volatile bool lpspi_master_edma_handle_t::isPcsContinuous

##### 30.3.2.1.0.22.2 volatile bool lpspi_master_edma_handle_t::isByteSwap

##### 30.3.2.1.0.22.3 volatile uint8_t lpspi_master_edma_handle_t::fifoSize

##### 30.3.2.1.0.22.4 volatile uint8_t lpspi_master_edma_handle_t::rxWatermark

##### 30.3.2.1.0.22.5 volatile uint8_t lpspi_master_edma_handle_t::bytesEachWrite

##### 30.3.2.1.0.22.6 volatile uint8_t lpspi_master_edma_handle_t::bytesEachRead

##### 30.3.2.1.0.22.7 volatile uint8_t lpspi_master_edma_handle_t::bytesLastRead

##### 30.3.2.1.0.22.8 volatile uint8_t lpspi_master_edma_handle_t::isThereExtraRxBytes

##### 30.3.2.1.0.22.9 uint8_t* volatile lpspi_master_edma_handle_t::txData

##### 30.3.2.1.0.22.10 uint8_t* volatile lpspi_master_edma_handle_t::rxData

##### 30.3.2.1.0.22.11 volatile size_t lpspi_master_edma_handle_t::txRemainingByteCount

##### 30.3.2.1.0.22.12 volatile size_t lpspi_master_edma_handle_t::rxRemainingByteCount

##### 30.3.2.1.0.22.13 volatile uint32_t lpspi_master_edma_handle_t::writeRegRemainingTimes

##### 30.3.2.1.0.22.14 volatile uint32_t lpspi_master_edma_handle_t::readRegRemainingTimes

##### 30.3.2.1.0.22.15 uint32_t lpspi_master_edma_handle_t::txBuffIfNull

##### 30.3.2.1.0.22.16 uint32_t lpspi_master_edma_handle_t::rxBuffIfNull

##### 30.3.2.1.0.22.17 uint32_t lpspi_master_edma_handle_t::transmitCommand

##### 30.3.2.1.0.22.18 volatile uint8_t lpspi_master_edma_handle_t::state

##### 30.3.2.1.0.22.19 uint8_t lpspi_master_edma_handle_t::nbytes

##### 30.3.2.1.0.22.20 lpspi_master_edma_transfer_callback_t lpspi_master_edma_handle_t::callback

##### 30.3.2.1.0.22.21 void* lpspi_master_edma_handle_t::userData

### 30.3.2.2 struct _lpspi_slave_edma_handle

Forward declaration of the _lpspi_slave_edma_handle typedefs.

### Data Fields

- volatile bool isByteSwap
  *A flag that whether should byte swap.*
- volatile uint8_t fifoSize
  *FIFO dataSize.*
- volatile uint8_t rxWatermark
  *Rx watermark.*
- volatile uint8_t bytesEachWrite
  *Bytes for each write TDR.*
- volatile uint8_t bytesEachRead
  *Bytes for each read RDR.*
- volatile uint8_t bytesLastRead
  *Bytes for last read RDR.*
- volatile uint8_t isThereExtraRxBytes
  *Is there extra RX byte.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- uint8_t ∗volatile txData
  *Send buffer.*
- uint8_t ∗volatile rxData
  *Receive buffer.*
- volatile size_t txRemainingByteCount
  *Number of bytes remaining to send.*
- volatile size_t rxRemainingByteCount
  *Number of bytes remaining to receive.*
- volatile uint32_t writeRegRemainingTimes
  *Write TDR register remaining times.*
- volatile uint32_t readRegRemainingTimes
  *Read RDR register remaining times.*
- uint32_t totalByteCount
  *Number of transfer bytes.*
- uint32_t txBuffIfNull
  *Used if there is not txData for DMA purpose.*
- uint32_t rxBuffIfNull
  *Used if there is not rxData for DMA purpose.*
- volatile uint8_t state
  *LPSPI transfer state.*
- uint32_t errorCount
  *Error count for slave transfer.*
- lpspi_slave_edma_transfer_callback_t callback
  *Completion callback.*
- void ∗ userData
  *Callback user data.*
- edma_handle_t ∗ edmaRxRegToRxDataHandle
  *edma_handle_t handle point used for RxReg to RxData buff*
- edma_handle_t ∗ edmaTxDataToTxRegHandle
  *edma_handle_t handle point used for TxData to TxReg*
- edma_tcd_t lpspiSoftwareTCD [2]
  *SoftwareTCD, internal used.*

#### 30.3.2.2.0.23 Field Documentation

##### 30.3.2.2.0.23.1 volatile bool lpspi_slave_edma_handle_t::isByteSwap

##### 30.3.2.2.0.23.2 volatile uint8_t lpspi_slave_edma_handle_t::fifoSize

##### 30.3.2.2.0.23.3 volatile uint8_t lpspi_slave_edma_handle_t::rxWatermark

##### 30.3.2.2.0.23.4 volatile uint8_t lpspi_slave_edma_handle_t::bytesEachWrite

##### 30.3.2.2.0.23.5 volatile uint8_t lpspi_slave_edma_handle_t::bytesEachRead

##### 30.3.2.2.0.23.6 volatile uint8_t lpspi_slave_edma_handle_t::bytesLastRead

##### 30.3.2.2.0.23.7 volatile uint8_t lpspi_slave_edma_handle_t::isThereExtraRxBytes

##### 30.3.2.2.0.23.8 uint8_t lpspi_slave_edma_handle_t::nbytes

##### 30.3.2.2.0.23.9 uint8_t* volatile lpspi_slave_edma_handle_t::txData

##### 30.3.2.2.0.23.10 uint8_t* volatile lpspi_slave_edma_handle_t::rxData

##### 30.3.2.2.0.23.11 volatile size_t lpspi_slave_edma_handle_t::txRemainingByteCount

##### 30.3.2.2.0.23.12 volatile size_t lpspi_slave_edma_handle_t::rxRemainingByteCount

##### 30.3.2.2.0.23.13 volatile uint32_t lpspi_slave_edma_handle_t::writeRegRemainingTimes

##### 30.3.2.2.0.23.14 volatile uint32_t lpspi_slave_edma_handle_t::readRegRemainingTimes

##### 30.3.2.2.0.23.15 uint32_t lpspi_slave_edma_handle_t::txBuffIfNull

##### 30.3.2.2.0.23.16 uint32_t lpspi_slave_edma_handle_t::rxBuffIfNull

##### 30.3.2.2.0.23.17 volatile uint8_t lpspi_slave_edma_handle_t::state

##### 30.3.2.2.0.23.18 uint32_t lpspi_slave_edma_handle_t::errorCount

##### 30.3.2.2.0.23.19 lpspi_slave_edma_transfer_callback_t lpspi_slave_edma_handle_t::callback

##### 30.3.2.2.0.23.20 void* lpspi_slave_edma_handle_t::userData

### 30.3.3 Macro Definition Documentation

#### 30.3.3.1 #define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

### 30.3.4 Typedef Documentation

#### 30.3.4.1 typedef void(* lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | Pointer to the handle for the LPSPI master. |
| *status* | Success or error code describing whether the transfer completed. |
| *userData* | Arbitrary pointer-dataSized value passed from the application. |

#### 30.3.4.2 typedef void(∗ lpspi_slave_edma_transfer_callback_t)(LPSPI_Type ∗base, lpspi_slave_edma_handle_t ∗handle, status_t status, void ∗userData)

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | Pointer to the handle for the LPSPI slave. |
| *status* | Success or error code describing whether the transfer completed. |
| *userData* | Arbitrary pointer-dataSized value passed from the application. |

### 30.3.5 Function Documentation

#### 30.3.5.1 void LPSPI_MasterTransferCreateHandleEDMA ( LPSPI_Type ∗ *base*, lpspi_master_edma_handle_t ∗ *handle*, lpspi_master_edma_transfer_callback_t *callback*, void ∗ *userData*, edma_handle_t ∗ *edmaRxRegToRxDataHandle*, edma_handle_t ∗ *edmaTxDataToTxRegHandle* )

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Rx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaIntermediaryToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |

| | |
|---|---|
| *handle* | LPSPI handle pointer to lpspi_master_edma_handle_t. |
| *callback* | LPSPI callback. |
| *userData* | callback function parameter. |
| *edmaRxRegToRxDataHandle* | edmaRxRegToRxDataHandle pointer to edma_handle_t. |
| *edmaTxDataToTxRegHandle* | edmaTxDataToTxRegHandle pointer to edma_handle_t. |

#### 30.3.5.2 status_t LPSPI_MasterTransferEDMA ( LPSPI_Type * *base*, lpspi_master_edma_handle_t * *handle*, lpspi_transfer_t * *transfer* )

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | pointer to lpspi_master_edma_handle_t structure which stores the transfer state. |
| *transfer* | pointer to lpspi_transfer_t structure. |

Returns

status of status_t.

#### 30.3.5.3 void LPSPI_MasterTransferAbortEDMA ( LPSPI_Type * *base*, lpspi_master_edma_handle_t * *handle* )

This function aborts a transfer which is using eDMA.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | pointer to lpspi_master_edma_handle_t structure which stores the transfer state. |

#### 30.3.5.4 status_t LPSPI_MasterTransferGetCountEDMA ( LPSPI_Type * *base*, lpspi_master_edma_handle_t * *handle*, size_t * *count* )

This function gets the master eDMA transfer remaining bytes.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | pointer to lpspi_master_edma_handle_t structure which stores the transfer state. |
| *count* | Number of bytes transferred so far by the EDMA transaction. |

Returns

status of status_t.

#### 30.3.5.5 void LPSPI_SlaveTransferCreateHandleEDMA ( LPSPI_Type * *base*, lpspi_slave_edma_handle_t * *handle*, lpspi_slave_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *edmaRxRegToRxDataHandle*, edma_handle_t * *edmaTxDataToTxRegHandle* )

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | LPSPI handle pointer to lpspi_slave_edma_handle_t. |

| | |
|---|---|
| *callback* | LPSPI callback. |
| *userData* | callback function parameter. |
| *edmaRxRegToRxDataHandle* | edmaRxRegToRxDataHandle pointer to edma_handle_t. |
| *edmaTxDataToTxRegHandle* | edmaTxDataToTxRegHandle pointer to edma_handle_t. |

#### 30.3.5.6 status_t LPSPI_SlaveTransferEDMA ( LPSPI_Type * *base*, lpspi_slave_edma_handle_t * *handle*, lpspi_transfer_t * *transfer* )

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | pointer to lpspi_slave_edma_handle_t structure which stores the transfer state. |
| *transfer* | pointer to lpspi_transfer_t structure. |

Returns

status of status_t.

#### 30.3.5.7 void LPSPI_SlaveTransferAbortEDMA ( LPSPI_Type * *base*, lpspi_slave_edma_handle_t * *handle* )

This function aborts a transfer which is using eDMA.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | pointer to lpspi_slave_edma_handle_t structure which stores the transfer state. |

#### 30.3.5.8 status_t LPSPI_SlaveTransferGetCountEDMA ( LPSPI_Type * *base*, lpspi_slave_edma_handle_t * *handle*, size_t * *count* )

This function gets the slave eDMA transfer remaining bytes.

Parameters

| | |
|---|---|
| *base* | LPSPI peripheral base address. |
| *handle* | pointer to lpspi_slave_edma_handle_t structure which stores the transfer state. |
| *count* | Number of bytes transferred so far by the eDMA transaction. |

Returns

status of status_t.

## 30.4 LPSPI FreeRTOS Driver

### 30.4.1 Overview

#### Driver version

- #define FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *LPSPI freertos driver version 2.0.2.*

#### LPSPI RTOS Operation

- status_t LPSPI_RTOS_Init (lpspi_rtos_handle_t *handle, LPSPI_Type *base, const lpspi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  *Initializes LPSPI.*
- status_t LPSPI_RTOS_Deinit (lpspi_rtos_handle_t *handle)
  *Deinitializes the LPSPI.*
- status_t LPSPI_RTOS_Transfer (lpspi_rtos_handle_t *handle, lpspi_transfer_t *transfer)
  *Performs SPI transfer.*

### 30.4.2 Macro Definition Documentation

#### 30.4.2.1 #define FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

### 30.4.3 Function Documentation

#### 30.4.3.1 status_t LPSPI_RTOS_Init ( lpspi_rtos_handle_t * *handle*, LPSPI_Type * *base*, const lpspi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz* )

This function initializes the LPSPI module and related RTOS context.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPSPI handle, the pointer to an allocated space for RTOS context. |
| *base* | The pointer base address of the LPSPI instance to initialize. |
| *masterConfig* | Configuration structure to set-up LPSPI in master mode. |
| *srcClock_Hz* | Frequency of input clock of the LPSPI module. |

Returns

status of the operation.

#### 30.4.3.2 status_t LPSPI_RTOS_Deinit ( lpspi_rtos_handle_t * *handle* )

This function deinitializes the LPSPI module and related RTOS context.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPSPI handle. |

#### 30.4.3.3 status_t LPSPI_RTOS_Transfer ( lpspi_rtos_handle_t * *handle,* lpspi_transfer_t * *transfer* )

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPSPI handle. |
| *transfer* | Structure specifying the transfer parameters. |

Returns

status of the operation.

# Chapter 31
# LPUART: Low Power UART Driver

## 31.1 Overview

### Modules

- LPUART DMA Driver
- LPUART Driver
- LPUART FreeRTOS Driver
- LPUART eDMA Driver

## 31.2 LPUART Driver

### 31.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

### 31.2.2 Typical use case

#### 31.2.2.1 LPUART Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lpuart

## Data Structures

- struct lpuart_config_t
  *LPUART configuration structure.* *More...*
- struct lpuart_transfer_t
  *LPUART transfer structure.* *More...*
- struct lpuart_handle_t
  *LPUART handle structure.* *More...*

## Typedefs

- typedef void(∗ lpuart_transfer_callback_t )(LPUART_Type ∗base, lpuart_handle_t ∗handle, status_t status, void ∗userData)
  *LPUART transfer callback function.*

## Enumerations

- enum _lpuart_status {
  kStatus_LPUART_TxBusy = MAKE_STATUS(kStatusGroup_LPUART, 0),
  kStatus_LPUART_RxBusy = MAKE_STATUS(kStatusGroup_LPUART, 1),
  kStatus_LPUART_TxIdle = MAKE_STATUS(kStatusGroup_LPUART, 2),
  kStatus_LPUART_RxIdle = MAKE_STATUS(kStatusGroup_LPUART, 3),
  kStatus_LPUART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 4),
  kStatus_LPUART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 5),
  kStatus_LPUART_FlagCannotClearManually = MAKE_STATUS(kStatusGroup_LPUART, 6),
  kStatus_LPUART_Error = MAKE_STATUS(kStatusGroup_LPUART, 7),
  kStatus_LPUART_RxRingBufferOverrun,
  kStatus_LPUART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_LPUART, 9),
  kStatus_LPUART_NoiseError = MAKE_STATUS(kStatusGroup_LPUART, 10),
  kStatus_LPUART_FramingError = MAKE_STATUS(kStatusGroup_LPUART, 11),
  kStatus_LPUART_ParityError = MAKE_STATUS(kStatusGroup_LPUART, 12),
  kStatus_LPUART_BaudrateNotSupport,
  kStatus_LPUART_IdleLineDetected = MAKE_STATUS(kStatusGroup_LPUART, 14) }
  *Error codes for the LPUART driver.*
- enum lpuart_parity_mode_t {
  kLPUART_ParityDisabled = 0x0U,
  kLPUART_ParityEven = 0x2U,
  kLPUART_ParityOdd = 0x3U }
  *LPUART parity mode.*
- enum lpuart_data_bits_t {
  kLPUART_EightDataBits = 0x0U,
  kLPUART_SevenDataBits = 0x1U }
  *LPUART data bits count.*
- enum lpuart_stop_bit_count_t {
  kLPUART_OneStopBit = 0U,
  kLPUART_TwoStopBit = 1U }
  *LPUART stop bit count.*
- enum lpuart_transmit_cts_source_t {
  kLPUART_CtsSourcePin = 0U,
  kLPUART_CtsSourceMatchResult = 1U }
  *LPUART transmit CTS source.*
- enum lpuart_transmit_cts_config_t {
  kLPUART_CtsSampleAtStart = 0U,
  kLPUART_CtsSampleAtIdle = 1U }
  *LPUART transmit CTS configure.*
- enum lpuart_idle_type_select_t {
  kLPUART_IdleTypeStartBit = 0U,
  kLPUART_IdleTypeStopBit = 1U }
  *LPUART idle flag type defines when the receiver starts counting.*
- enum lpuart_idle_config_t {

kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
*LPUART idle detected configuration.*

- enum _lpuart_interrupt_enable {
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIE_MASK >> 8),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK),
kLPUART_TxFifoOverflowInterruptEnable = (LPUART_FIFO_TXOFE_MASK >> 8),
kLPUART_RxFifoUnderflowInterruptEnable = (LPUART_FIFO_RXUFE_MASK >> 8) }
*LPUART interrupt configuration structure, default settings all disabled.*
- enum _lpuart_flags {
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag,
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (int)(LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag,
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag = LPUART_STAT_MA1F_MASK,
kLPUART_DataMatch2Flag = LPUART_STAT_MA2F_MASK,
kLPUART_NoiseErrorInRxDataRegFlag,
kLPUART_ParityErrorInRxDataRegFlag,
kLPUART_TxFifoEmptyFlag = (LPUART_FIFO_TXEMPT_MASK >> 16),
kLPUART_RxFifoEmptyFlag = (LPUART_FIFO_RXEMPT_MASK >> 16),
kLPUART_TxFifoOverflowFlag,
kLPUART_RxFifoUnderflowFlag }
*LPUART status flags.*

### Driver version

- #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 2, 7))
  *LPUART driver version 2.2.7.*

### Software Reset

- static void LPUART_SoftwareReset (LPUART_Type *base)
  *Resets the LPUART using software.*

### Initialization and deinitialization

- 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.*
- void LPUART_Deinit (LPUART_Type *base)
  *Deinitializes a LPUART instance.*
- void LPUART_GetDefaultConfig (lpuart_config_t *config)
  *Gets the default configuration structure.*
- status_t LPUART_SetBaudRate (LPUART_Type *base, uint32_t baudRate_Bps, uint32_t src-Clock_Hz)
  *Sets the LPUART instance baudrate.*

### Status

- uint32_t LPUART_GetStatusFlags (LPUART_Type *base)
  *Gets LPUART status flags.*
- status_t LPUART_ClearStatusFlags (LPUART_Type *base, uint32_t mask)
  *Clears status flags with a provided mask.*

### Interrupts

- void LPUART_EnableInterrupts (LPUART_Type *base, uint32_t mask)
  *Enables LPUART interrupts according to a provided mask.*
- void LPUART_DisableInterrupts (LPUART_Type *base, uint32_t mask)
  *Disables LPUART interrupts according to a provided mask.*
- uint32_t LPUART_GetEnabledInterrupts (LPUART_Type *base)
  *Gets enabled LPUART interrupts.*
- static uint32_t LPUART_GetDataRegisterAddress (LPUART_Type *base)
  *Gets the LPUART data register address.*
- static void LPUART_EnableTxDMA (LPUART_Type *base, bool enable)
  *Enables or disables the LPUART transmitter DMA request.*
- static void LPUART_EnableRxDMA (LPUART_Type *base, bool enable)
  *Enables or disables the LPUART receiver DMA.*

## Bus Operations

- uint32_t LPUART_GetInstance (LPUART_Type *base)
  *Get the LPUART instance from peripheral base address.*
- static void LPUART_EnableTx (LPUART_Type *base, bool enable)
  *Enables or disables the LPUART transmitter.*
- static void LPUART_EnableRx (LPUART_Type *base, bool enable)
  *Enables or disables the LPUART receiver.*
- static void LPUART_WriteByte (LPUART_Type *base, uint8_t data)
  *Writes to the transmitter register.*
- static uint8_t LPUART_ReadByte (LPUART_Type *base)
  *Reads the receiver register.*
- void LPUART_WriteBlocking (LPUART_Type *base, const uint8_t *data, size_t length)
  *Writes to the transmitter register using a blocking method.*
- status_t LPUART_ReadBlocking (LPUART_Type *base, uint8_t *data, size_t length)
  *Reads the receiver data register using a blocking method.*

## Transactional

- void LPUART_TransferCreateHandle (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_callback_t callback, void *userData)
  *Initializes the LPUART handle.*
- status_t LPUART_TransferSendNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer)
  *Transmits a buffer of data using the interrupt method.*
- void LPUART_TransferStartRingBuffer (LPUART_Type *base, lpuart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  *Sets up the RX ring buffer.*
- void LPUART_TransferStopRingBuffer (LPUART_Type *base, lpuart_handle_t *handle)
  *Aborts the background transfer and uninstalls the ring buffer.*
- size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type *base, lpuart_handle_t *handle)
  *Get the length of received data in RX ring buffer.*
- void LPUART_TransferAbortSend (LPUART_Type *base, lpuart_handle_t *handle)
  *Aborts the interrupt-driven data transmit.*
- status_t LPUART_TransferGetSendCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)
  *Gets the number of bytes that have been written to the LPUART transmitter register.*
- 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.*
- void LPUART_TransferAbortReceive (LPUART_Type *base, lpuart_handle_t *handle)
  *Aborts the interrupt-driven data receiving.*
- status_t LPUART_TransferGetReceiveCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)
  *Gets the number of bytes that have been received.*
- void LPUART_TransferHandleIRQ (LPUART_Type *base, lpuart_handle_t *handle)
  *LPUART IRQ handle function.*
- void LPUART_TransferHandleErrorIRQ (LPUART_Type *base, lpuart_handle_t *handle)

*LPUART Error IRQ handle function.*

### 31.2.3 Data Structure Documentation

#### 31.2.3.1 struct lpuart_config_t

#### Data Fields

- uint32_t baudRate_Bps
  *LPUART baud rate.*
- lpuart_parity_mode_t parityMode
  *Parity mode, disabled (default), even, odd.*
- lpuart_data_bits_t dataBitsCount
  *Data bits count, eight (default), seven.*
- bool isMsb
  *Data bits order, LSB (default), MSB.*
- lpuart_stop_bit_count_t stopBitCount
  *Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- uint8_t txFifoWatermark
  *TX FIFO watermark.*
- uint8_t rxFifoWatermark
  *RX FIFO watermark.*
- bool enableRxRTS
  *RX RTS enable.*
- bool enableTxCTS
  *TX CTS enable.*
- lpuart_transmit_cts_source_t txCtsSource
  *TX CTS source.*
- lpuart_transmit_cts_config_t txCtsConfig
  *TX CTS configure.*
- lpuart_idle_type_select_t rxIdleType
  *RX IDLE type.*
- lpuart_idle_config_t rxIdleConfig
  *RX IDLE configuration.*
- bool enableTx
  *Enable TX.*
- bool enableRx
  *Enable RX.*

#### 31.2.3.1.0.24 Field Documentation

##### 31.2.3.1.0.24.1 lpuart_idle_type_select_t lpuart_config_t::rxIdleType

##### 31.2.3.1.0.24.2 lpuart_idle_config_t lpuart_config_t::rxIdleConfig

### 31.2.3.2 struct lpuart_transfer_t

### Data Fields

- uint8_t * data
  *The buffer of data to be transfer.*
- size_t dataSize
  *The byte count to be transfer.*

#### 31.2.3.2.0.25 Field Documentation

##### 31.2.3.2.0.25.1 uint8_t* lpuart_transfer_t::data

##### 31.2.3.2.0.25.2 size_t lpuart_transfer_t::dataSize

### 31.2.3.3 struct _lpuart_handle

### Data Fields

- uint8_t *volatile txData
  *Address of remaining data to send.*
- volatile size_t txDataSize
  *Size of the remaining data to send.*
- size_t txDataSizeAll
  *Size of the data to send out.*
- uint8_t *volatile rxData
  *Address of remaining data to receive.*
- volatile size_t rxDataSize
  *Size of the remaining data to receive.*
- size_t rxDataSizeAll
  *Size of the data to receive.*
- uint8_t * rxRingBuffer
  *Start address of the receiver ring buffer.*
- size_t rxRingBufferSize
  *Size of the ring buffer.*
- volatile uint16_t rxRingBufferHead
  *Index for the driver to store received data into ring buffer.*
- volatile uint16_t rxRingBufferTail
  *Index for the user to get data from the ring buffer.*
- 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.*

#### 31.2.3.3.0.26 Field Documentation

##### 31.2.3.3.0.26.1 uint8_t* volatile lpuart_handle_t::txData

##### 31.2.3.3.0.26.2 volatile size_t lpuart_handle_t::txDataSize

##### 31.2.3.3.0.26.3 size_t lpuart_handle_t::txDataSizeAll

##### 31.2.3.3.0.26.4 uint8_t* volatile lpuart_handle_t::rxData

##### 31.2.3.3.0.26.5 volatile size_t lpuart_handle_t::rxDataSize

##### 31.2.3.3.0.26.6 size_t lpuart_handle_t::rxDataSizeAll

##### 31.2.3.3.0.26.7 uint8_t* lpuart_handle_t::rxRingBuffer

##### 31.2.3.3.0.26.8 size_t lpuart_handle_t::rxRingBufferSize

##### 31.2.3.3.0.26.9 volatile uint16_t lpuart_handle_t::rxRingBufferHead

##### 31.2.3.3.0.26.10 volatile uint16_t lpuart_handle_t::rxRingBufferTail

##### 31.2.3.3.0.26.11 lpuart_transfer_callback_t lpuart_handle_t::callback

##### 31.2.3.3.0.26.12 void* lpuart_handle_t::userData

##### 31.2.3.3.0.26.13 volatile uint8_t lpuart_handle_t::txState

##### 31.2.3.3.0.26.14 volatile uint8_t lpuart_handle_t::rxState

##### 31.2.3.3.0.26.15 bool lpuart_handle_t::isSevenDataBits

### 31.2.4 Macro Definition Documentation

#### 31.2.4.1 #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 2, 7))

### 31.2.5 Typedef Documentation

#### 31.2.5.1 typedef void(* lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle, status_t status, void *userData)

### 31.2.6 Enumeration Type Documentation

#### 31.2.6.1 enum _lpuart_status

Enumerator

***kStatus_LPUART_TxBusy*** TX busy.

*kStatus_LPUART_RxBusy* RX busy.
*kStatus_LPUART_TxIdle* LPUART transmitter is idle.
*kStatus_LPUART_RxIdle* LPUART receiver is idle.
*kStatus_LPUART_TxWatermarkTooLarge* TX FIFO watermark too large.
*kStatus_LPUART_RxWatermarkTooLarge* RX FIFO watermark too large.
*kStatus_LPUART_FlagCannotClearManually* Some flag can't manually clear.
*kStatus_LPUART_Error* Error happens on LPUART.
*kStatus_LPUART_RxRingBufferOverrun* LPUART RX software ring buffer overrun.
*kStatus_LPUART_RxHardwareOverrun* LPUART RX receiver overrun.
*kStatus_LPUART_NoiseError* LPUART noise error.
*kStatus_LPUART_FramingError* LPUART framing error.
*kStatus_LPUART_ParityError* LPUART parity error.
*kStatus_LPUART_BaudrateNotSupport* Baudrate is not support in current clock source.
*kStatus_LPUART_IdleLineDetected* IDLE flag.

#### 31.2.6.2 enum lpuart_parity_mode_t

Enumerator

*kLPUART_ParityDisabled* Parity disabled.
*kLPUART_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.
*kLPUART_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

#### 31.2.6.3 enum lpuart_data_bits_t

Enumerator

*kLPUART_EightDataBits* Eight data bit.
*kLPUART_SevenDataBits* Seven data bit.

#### 31.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

*kLPUART_OneStopBit* One stop bit.
*kLPUART_TwoStopBit* Two stop bits.

#### 31.2.6.5 enum lpuart_transmit_cts_source_t

Enumerator

*kLPUART_CtsSourcePin* CTS resource is the LPUART_CTS pin.
*kLPUART_CtsSourceMatchResult* CTS resource is the match result.

#### 31.2.6.6 enum lpuart_transmit_cts_config_t

Enumerator

***kLPUART_CtsSampleAtStart*** CTS input is sampled at the start of each character.
***kLPUART_CtsSampleAtIdle*** CTS input is sampled when the transmitter is idle.

#### 31.2.6.7 enum lpuart_idle_type_select_t

Enumerator

***kLPUART_IdleTypeStartBit*** Start counting after a valid start bit.
***kLPUART_IdleTypeStopBit*** Start counting after a stop bit.

#### 31.2.6.8 enum lpuart_idle_config_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

***kLPUART_IdleCharacter1*** the number of idle characters.
***kLPUART_IdleCharacter2*** the number of idle characters.
***kLPUART_IdleCharacter4*** the number of idle characters.
***kLPUART_IdleCharacter8*** the number of idle characters.
***kLPUART_IdleCharacter16*** the number of idle characters.
***kLPUART_IdleCharacter32*** the number of idle characters.
***kLPUART_IdleCharacter64*** the number of idle characters.
***kLPUART_IdleCharacter128*** the number of idle characters.

#### 31.2.6.9 enum _lpuart_interrupt_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

***kLPUART_LinBreakInterruptEnable*** LIN break detect.
***kLPUART_RxActiveEdgeInterruptEnable*** Receive Active Edge.
***kLPUART_TxDataRegEmptyInterruptEnable*** Transmit data register empty.
***kLPUART_TransmissionCompleteInterruptEnable*** Transmission complete.
***kLPUART_RxDataRegFullInterruptEnable*** Receiver data register full.
***kLPUART_IdleLineInterruptEnable*** Idle line.
***kLPUART_RxOverrunInterruptEnable*** Receiver Overrun.
***kLPUART_NoiseErrorInterruptEnable*** Noise error flag.

***kLPUART_FramingErrorInterruptEnable*** Framing error flag.
***kLPUART_ParityErrorInterruptEnable*** Parity error flag.
***kLPUART_TxFifoOverflowInterruptEnable*** Transmit FIFO Overflow.
***kLPUART_RxFifoUnderflowInterruptEnable*** Receive FIFO Underflow.

#### 31.2.6.10 enum _lpuart_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

***kLPUART_TxDataRegEmptyFlag*** Transmit data register empty flag, sets when transmit buffer is empty.
***kLPUART_TransmissionCompleteFlag*** Transmission complete flag, sets when transmission activity complete.
***kLPUART_RxDataRegFullFlag*** Receive data register full flag, sets when the receive data buffer is full.
***kLPUART_IdleLineFlag*** Idle line detect flag, sets when idle line detected.
***kLPUART_RxOverrunFlag*** Receive Overrun, sets when new data is received before data is read from receive register.
***kLPUART_NoiseErrorFlag*** Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets
***kLPUART_FramingErrorFlag*** Frame error flag, sets if logic 0 was detected where stop bit expected.
***kLPUART_ParityErrorFlag*** If parity enabled, sets upon parity error detection.
***kLPUART_LinBreakFlag*** LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
***kLPUART_RxActiveEdgeFlag*** Receive pin active edge interrupt flag, sets when active edge detected.
***kLPUART_RxActiveFlag*** Receiver Active Flag (RAF), sets at beginning of valid start bit.
***kLPUART_DataMatch1Flag*** The next character to be read from LPUART_DATA matches MA1.
***kLPUART_DataMatch2Flag*** The next character to be read from LPUART_DATA matches MA2.
***kLPUART_NoiseErrorInRxDataRegFlag*** NOISY bit, sets if noise detected in current data word.
***kLPUART_ParityErrorInRxDataRegFlag*** PARITY bit, sets if noise detected in current data word.

***kLPUART_TxFifoEmptyFlag*** TXEMPT bit, sets if transmit buffer is empty.
***kLPUART_RxFifoEmptyFlag*** RXEMPT bit, sets if receive buffer is empty.
***kLPUART_TxFifoOverflowFlag*** TXOF bit, sets if transmit buffer overflow occurred.
***kLPUART_RxFifoUnderflowFlag*** RXUF bit, sets if receive buffer underflow occurred.

### 31.2.7 Function Documentation

#### 31.2.7.1 static void LPUART_SoftwareReset ( LPUART_Type * *base* ) [inline], [static]

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |

#### 31.2.7.2 status_t LPUART_Init ( LPUART_Type * *base*, const lpuart_config_t * *config*, uint32_t *srcClock_Hz* )

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefault-Config() 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 |

#### 31.2.7.3 void LPUART_Deinit ( LPUART_Type * *base* )

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |

#### 31.2.7.4 void LPUART_GetDefaultConfig ( lpuart_config_t * *config* )

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. |

#### 31.2.7.5 status_t LPUART_SetBaudRate ( LPUART_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz* )

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. |

#### 31.2.7.6 uint32_t LPUART_GetStatusFlags ( LPUART_Type * *base* )

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.

#### 31.2.7.7 status_t LPUART_ClearStatusFlags ( LPUART_Type * *base*, uint32_t *mask* )

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorInRxDataRegFlag, kLPUART_ParityErrorInRxDataRegFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *mask* | the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask. |

Returns

0 succeed, others failed.

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. |

#### 31.2.7.8 void LPUART_EnableInterrupts ( LPUART_Type * *base*, uint32_t *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 _uart_interrupt_enable. |

#### 31.2.7.9 void LPUART_DisableInterrupts ( LPUART_Type * *base*, uint32_t *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. |

#### 31.2.7.10 uint32_t LPUART_GetEnabledInterrupts ( LPUART_Type * *base* )

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.

#### 31.2.7.11 static uint32_t LPUART_GetDataRegisterAddress ( LPUART_Type * *base* ) [inline], [static]

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.

#### 31.2.7.12 static void LPUART_EnableTxDMA ( LPUART_Type * *base*, bool *enable* ) [inline], [static]

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. |

#### 31.2.7.13 static void LPUART_EnableRxDMA ( LPUART_Type * *base*, bool *enable* ) [inline], [static]

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. |

#### 31.2.7.14 uint32_t LPUART_GetInstance ( LPUART_Type * *base* )

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |

Returns

LPUART instance.

#### 31.2.7.15 static void LPUART_EnableTx ( LPUART_Type * *base,* bool *enable* ) [inline], [static]

This function enables or disables the LPUART transmitter.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *enable* | True to enable, false to disable. |

#### 31.2.7.16 static void LPUART_EnableRx ( LPUART_Type * *base,* bool *enable* ) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *enable* | True to enable, false to disable. |

#### 31.2.7.17 static void LPUART_WriteByte ( LPUART_Type * *base,* uint8_t *data* ) [inline], [static]

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. |

#### 31.2.7.18 static uint8_t LPUART_ReadByte ( LPUART_Type * *base* ) [inline], [static]

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.

#### 31.2.7.19 void LPUART_WriteBlocking ( LPUART_Type * *base*, const uint8_t * *data*, size_t *length* )

This function polls the transmitter register, waits for the register to be empty or for TX FIFO to have room, and writes data to the transmitter buffer.

Note

This function does not check whether all data has been sent out to the bus. Before disabling the transmitter, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *data* | Start address of the data to write. |
| *length* | Size of the data to write. |

#### 31.2.7.20 status_t LPUART_ReadBlocking ( LPUART_Type * *base*, uint8_t * *data*, size_t *length* )

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_Success* | Successfully received all data. |

#### 31.2.7.21 void LPUART_TransferCreateHandle ( LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_transfer_callback_t *callback*, void * *userData* )

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. |

#### 31.2.7.22 status_t LPUART_TransferSendNonBlocking ( LPUART_Type * *base,* lpuart_handle_t * *handle,* lpuart_transfer_t * *xfer* )

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. |

#### 31.2.7.23 void LPUART_TransferStartRingBuffer ( LPUART_Type * *base,* lpuart_handle_t * *handle,* uint8_t * *ringBuffer,* size_t *ringBufferSize* )

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. |

#### 31.2.7.24 void LPUART_TransferStopRingBuffer ( LPUART_Type * *base,* lpuart_handle_t * *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |

#### 31.2.7.25 size_t LPUART_TransferGetRxRingBufferLength ( LPUART_Type * *base,* lpuart_handle_t * *handle* )

Parameters

| | |
|---|---|
| *handle* | LPUART handle pointer. |

Returns

Length of received data in RX ring buffer.

#### 31.2.7.26 void LPUART_TransferAbortSend ( LPUART_Type * *base,* lpuart_handle_t * *handle* )

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. |

#### 31.2.7.27 status_t LPUART_TransferGetSendCount ( LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count* )

This function gets the number of bytes that have been written to LPUART TX register 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`; |

#### 31.2.7.28 status_t LPUART_TransferReceiveNonBlocking ( LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_transfer_t * *xfer*, size_t * *receivedBytes* )

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_Rx-Busy* | Previous receive request is not finished. |
| *kStatus_InvalidArgument* | Invalid argument. |

#### 31.2.7.29 void LPUART_TransferAbortReceive ( LPUART_Type * *base*, lpuart_handle_t * *handle* )

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. |

#### 31.2.7.30 status_t LPUART_TransferGetReceiveCount ( LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count* )

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`; |

#### 31.2.7.31 void LPUART_TransferHandleIRQ ( LPUART_Type * *base*, lpuart_handle_t * *handle* )

This function handles the LPUART transmit and receive IRQ request.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |

#### 31.2.7.32 void LPUART_TransferHandleErrorIRQ ( LPUART_Type * *base*, lpuart_handle_t * *handle* )

This function handles the LPUART error IRQ request.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |

## 31.3 LPUART DMA Driver

### 31.3.1 Overview

#### Data Structures

- struct lpuart_dma_handle_t
  *LPUART DMA handle. More...*

#### Typedefs

- typedef void(∗ lpuart_dma_transfer_callback_t )(LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, status_t status, void ∗userData)
  *LPUART transfer callback function.*

#### Driver version

- #define FSL_LPUART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 6))
  *LPUART DMA driver version 2.2.6.*

#### EDMA transactional

- void LPUART_TransferCreateHandleDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  *Initializes the LPUART handle which is used in transactional functions.*
- status_t LPUART_TransferSendDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  *Sends data using DMA.*
- status_t LPUART_TransferReceiveDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  *Receives data using DMA.*
- void LPUART_TransferAbortSendDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle)
  *Aborts the sent data using DMA.*
- void LPUART_TransferAbortReceiveDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle)
  *Aborts the received data using DMA.*
- status_t LPUART_TransferGetSendCountDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, uint32_t ∗count)
  *Gets the number of bytes written to the LPUART TX register.*
- status_t LPUART_TransferGetReceiveCountDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, uint32_t ∗count)
  *Gets the number of received bytes.*

## 31.3.2 Data Structure Documentation

### 31.3.2.1 struct _lpuart_dma_handle

#### Data Fields

- lpuart_dma_transfer_callback_t callback
  *Callback function.*
- void * userData
  *LPUART callback function parameter.*
- size_t rxDataSizeAll
  *Size of the data to receive.*
- size_t txDataSizeAll
  *Size of the data to send out.*
- dma_handle_t * txDmaHandle
  *The DMA TX channel used.*
- dma_handle_t * rxDmaHandle
  *The DMA RX channel used.*
- volatile uint8_t txState
  *TX transfer state.*
- volatile uint8_t rxState
  *RX transfer state.*

#### 31.3.2.1.0.27 Field Documentation

#### 31.3.2.1.0.27.1 lpuart_dma_transfer_callback_t lpuart_dma_handle_t::callback

#### 31.3.2.1.0.27.2 void* lpuart_dma_handle_t::userData

#### 31.3.2.1.0.27.3 size_t lpuart_dma_handle_t::rxDataSizeAll

#### 31.3.2.1.0.27.4 size_t lpuart_dma_handle_t::txDataSizeAll

#### 31.3.2.1.0.27.5 dma_handle_t* lpuart_dma_handle_t::txDmaHandle

#### 31.3.2.1.0.27.6 dma_handle_t* lpuart_dma_handle_t::rxDmaHandle

#### 31.3.2.1.0.27.7 volatile uint8_t lpuart_dma_handle_t::txState

### 31.3.3 Macro Definition Documentation

#### 31.3.3.1 #define FSL_LPUART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 6))

### 31.3.4 Typedef Documentation

#### 31.3.4.1 typedef void(* lpuart_dma_transfer_callback_t)(LPUART_Type *base, lpuart_dma_handle_t *handle, status_t status, void *userData)

### 31.3.5 Function Documentation

#### 31.3.5.1 void LPUART_TransferCreateHandleDMA ( LPUART_Type * *base,* lpuart_dma_handle_t * *handle,* lpuart_dma_transfer_callback_t *callback,* void * *userData,* dma_handle_t * *txDmaHandle,* dma_handle_t * *rxDmaHandle* )

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | Pointer to lpuart_dma_handle_t structure. |
| *callback* | Callback function. |
| *userData* | User data. |
| *txDmaHandle* | User-requested DMA handle for TX DMA transfer. |
| *rxDmaHandle* | User-requested DMA handle for RX DMA transfer. |

#### 31.3.5.2 status_t LPUART_TransferSendDMA ( LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, lpuart_transfer_t * *xfer* )

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |
| *xfer* | LPUART DMA transfer structure. See lpuart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_LPUART_TxBusy* | Previous transfer on going. |
| *kStatus_InvalidArgument* | Invalid argument. |

#### 31.3.5.3 status_t LPUART_TransferReceiveDMA ( LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, lpuart_transfer_t * *xfer* )

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | Pointer to lpuart_dma_handle_t structure. |
| *xfer* | LPUART DMA transfer structure. See lpuart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_LPUART_Rx-Busy* | Previous transfer on going. |
| *kStatus_InvalidArgument* | Invalid argument. |

#### 31.3.5.4 void LPUART_TransferAbortSendDMA ( LPUART_Type * *base*, lpuart_dma_handle_t * *handle* )

This function aborts send data using DMA.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address |
| *handle* | Pointer to lpuart_dma_handle_t structure |

#### 31.3.5.5 void LPUART_TransferAbortReceiveDMA ( LPUART_Type * *base*, lpuart_dma_handle_t * *handle* )

This function aborts the received data using DMA.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address |
| *handle* | Pointer to lpuart_dma_handle_t structure |

#### 31.3.5.6 status_t LPUART_TransferGetSendCountDMA ( LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, uint32_t * *count* )

This function gets the number of bytes that have been written to LPUART TX register by DMA.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |
| *count* | Send bytes count. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | No send in progress. |
| *kStatus_InvalidArgument* | Parameter is invalid. |
| *kStatus_Success* | Get successfully through the parameter `count`; |

#### 31.3.5.7 status_t LPUART_TransferGetReceiveCountDMA ( LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, uint32_t * *count* )

This function gets the number of received bytes.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |
| *count* | Receive bytes count. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | No receive in progress. |
| *kStatus_InvalidArgument* | Parameter is invalid. |
| *kStatus_Success* | Get successfully through the parameter `count`; |

## 31.4 LPUART eDMA Driver

### 31.4.1 Overview

#### Data Structures

- struct lpuart_edma_handle_t
  *LPUART eDMA handle. More...*

#### Typedefs

- typedef void(∗ lpuart_edma_transfer_callback_t )(LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, status_t status, void ∗userData)
  *LPUART transfer callback function.*

#### Driver version

- #define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 7))
  *LPUART EDMA driver version 2.2.7.*

#### eDMA transactional

- void LPUART_TransferCreateHandleEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗txEdmaHandle, edma_handle_t ∗rxEdmaHandle)
  *Initializes the LPUART handle which is used in transactional functions.*
- status_t LPUART_SendEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  *Sends data using eDMA.*
- status_t LPUART_ReceiveEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  *Receives data using eDMA.*
- void LPUART_TransferAbortSendEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle)
  *Aborts the sent data using eDMA.*
- void LPUART_TransferAbortReceiveEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle)
  *Aborts the received data using eDMA.*
- status_t LPUART_TransferGetSendCountEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, uint32_t ∗count)
  *Gets the number of bytes written to the LPUART TX register.*
- status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, uint32_t ∗count)
  *Gets the number of received bytes.*

## 31.4.2 Data Structure Documentation

### 31.4.2.1 struct _lpuart_edma_handle

#### Data Fields

- lpuart_edma_transfer_callback_t callback
  *Callback function.*
- void * userData
  *LPUART callback function parameter.*
- size_t rxDataSizeAll
  *Size of the data to receive.*
- size_t txDataSizeAll
  *Size of the data to send out.*
- edma_handle_t * txEdmaHandle
  *The eDMA TX channel used.*
- edma_handle_t * rxEdmaHandle
  *The eDMA RX channel used.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- volatile uint8_t txState
  *TX transfer state.*
- volatile uint8_t rxState
  *RX transfer state.*

#### 31.4.2.1.0.28 Field Documentation

##### 31.4.2.1.0.28.1 lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback

##### 31.4.2.1.0.28.2 void* lpuart_edma_handle_t::userData

##### 31.4.2.1.0.28.3 size_t lpuart_edma_handle_t::rxDataSizeAll

##### 31.4.2.1.0.28.4 size_t lpuart_edma_handle_t::txDataSizeAll

##### 31.4.2.1.0.28.5 edma_handle_t* lpuart_edma_handle_t::txEdmaHandle

##### 31.4.2.1.0.28.6 edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle

##### 31.4.2.1.0.28.7 uint8_t lpuart_edma_handle_t::nbytes

##### 31.4.2.1.0.28.8 volatile uint8_t lpuart_edma_handle_t::txState

### 31.4.3 Macro Definition Documentation

#### 31.4.3.1 #define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 7))

### 31.4.4 Typedef Documentation

#### 31.4.4.1 typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)

### 31.4.5 Function Documentation

#### 31.4.5.1 void LPUART_TransferCreateHandleEDMA ( LPUART_Type * *base,* lpuart_edma_handle_t * *handle,* lpuart_edma_transfer_callback_t *callback,* void * *userData,* edma_handle_t * *txEdmaHandle,* edma_handle_t * *rxEdmaHandle* )

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | Pointer to lpuart_edma_handle_t structure. |
| *callback* | Callback function. |
| *userData* | User data. |
| *txEdmaHandle* | User requested DMA handle for TX DMA transfer. |
| *rxEdmaHandle* | User requested DMA handle for RX DMA transfer. |

#### 31.4.5.2 status_t LPUART_SendEDMA ( LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, lpuart_transfer_t * *xfer* )

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |
| *xfer* | LPUART eDMA transfer structure. See lpuart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others failed. |
| *kStatus_LPUART_TxBusy* | Previous transfer on going. |
| *kStatus_InvalidArgument* | Invalid argument. |

#### 31.4.5.3 status_t LPUART_ReceiveEDMA ( LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, lpuart_transfer_t * *xfer* )

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | Pointer to lpuart_edma_handle_t structure. |
| *xfer* | LPUART eDMA transfer structure, see lpuart_transfer_t. |

Return values

| | |
|---|---|
| *kStatus_Success* | if succeed, others fail. |
| *kStatus_LPUART_RxBusy* | Previous transfer ongoing. |
| *kStatus_InvalidArgument* | Invalid argument. |

#### 31.4.5.4 void LPUART_TransferAbortSendEDMA ( LPUART_Type * *base*, lpuart_edma_handle_t * *handle* )

This function aborts the sent data using eDMA.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | Pointer to lpuart_edma_handle_t structure. |

#### 31.4.5.5 void LPUART_TransferAbortReceiveEDMA ( LPUART_Type * *base*, lpuart_edma_handle_t * *handle* )

This function aborts the received data using eDMA.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | Pointer to lpuart_edma_handle_t structure. |

#### 31.4.5.6 status_t LPUART_TransferGetSendCountEDMA ( LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, uint32_t * *count* )

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |
| *count* | Send bytes count. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | No send in progress. |
| *kStatus_InvalidArgument* | Parameter is invalid. |
| *kStatus_Success* | Get successfully through the parameter `count`; |

#### 31.4.5.7 status_t LPUART_TransferGetReceiveCountEDMA ( LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, uint32_t * *count* )

This function gets the number of received bytes.

Parameters

| | |
|---|---|
| *base* | LPUART peripheral base address. |
| *handle* | LPUART handle pointer. |
| *count* | Receive bytes count. |

Return values

| | |
|---|---|
| *kStatus_NoTransferInProgress* | No receive in progress. |
| *kStatus_InvalidArgument* | Parameter is invalid. |
| *kStatus_Success* | Get successfully through the parameter `count`; |

## 31.5 LPUART FreeRTOS Driver

### 31.5.1 Overview

#### Data Structures

- struct lpuart_rtos_config_t
  *LPUART RTOS configuration structure. More...*

#### Driver version

- #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 6))
  *LPUART freertos driver version 2.2.6.*

#### LPUART RTOS Operation

- int LPUART_RTOS_Init (lpuart_rtos_handle_t *handle, lpuart_handle_t *t_handle, const lpuart_rtos_config_t *cfg)
  *Initializes an LPUART instance for operation in RTOS.*
- int LPUART_RTOS_Deinit (lpuart_rtos_handle_t *handle)
  *Deinitializes an LPUART instance for operation.*

#### LPUART transactional Operation

- int LPUART_RTOS_Send (lpuart_rtos_handle_t *handle, const uint8_t *buffer, uint32_t length)
  *Sends data in the background.*
- int LPUART_RTOS_Receive (lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received)
  *Receives data.*

### 31.5.2 Data Structure Documentation

#### 31.5.2.1 struct lpuart_rtos_config_t

#### Data Fields

- LPUART_Type * base
  *UART base address.*
- uint32_t srcclk
  *UART source clock in Hz.*
- uint32_t baudrate
  *Desired communication speed.*
- lpuart_parity_mode_t parity
  *Parity setting.*

- lpuart_stop_bit_count_t stopbits
  *Number of stop bits to use.*
- uint8_t * buffer
  *Buffer for background reception.*
- uint32_t buffer_size
  *Size of buffer for background reception.*

### 31.5.3 Macro Definition Documentation

#### 31.5.3.1 #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 6))

### 31.5.4 Function Documentation

#### 31.5.4.1 int LPUART_RTOS_Init ( lpuart_rtos_handle_t * *handle*, lpuart_handle_t * *t_handle*, const lpuart_rtos_config_t * *cfg* )

Parameters

| | |
|---|---|
| *handle* | The RTOS LPUART handle, the pointer to an allocated space for RTOS context. |
| *t_handle* | The pointer to an allocated space to store the transactional layer internal state. |
| *cfg* | The pointer to the parameters required to configure the LPUART after initialization. |

Returns

0 succeed, others failed

#### 31.5.4.2 int LPUART_RTOS_Deinit ( lpuart_rtos_handle_t * *handle* )

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPUART handle. |

#### 31.5.4.3 int LPUART_RTOS_Send ( lpuart_rtos_handle_t * *handle*, const uint8_t * *buffer*, uint32_t *length* )

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPUART handle. |
| *buffer* | The pointer to buffer to send. |
| *length* | The number of bytes to send. |

#### 31.5.4.4 int LPUART_RTOS_Receive ( lpuart_rtos_handle_t * *handle*, uint8_t * *buffer*, uint32_t *length*, size_t * *received* )

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

| | |
|---|---|
| *handle* | The RTOS LPUART handle. |
| *buffer* | The pointer to buffer where to write received data. |
| *length* | The number of bytes to receive. |
| *received* | The pointer to a variable of size_t where the number of received data is filled. |

# Chapter 32
# PIT: Periodic Interrupt Timer

## 32.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

## 32.2 Function groups

The PIT driver supports operating the module as a time counter.

### 32.2.1 Initialization and deinitialization

The function PIT_Init() initializes the PIT with specified configurations. The function PIT_GetDefaultConfig() gets the default configurations. The initialization function configures the PIT operation in debug mode.

The function PIT_SetTimerChainMode() configures the chain mode operation of each PIT channel.

The function PIT_Deinit() disables the PIT timers and disables the module clock.

### 32.2.2 Timer period Operations

The function PITR_SetTimerPeriod() sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function PIT_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. Users can call the utility macros provided in fsl_common.h to convert to microseconds or milliseconds.

### 32.2.3 Start and Stop timer operations

The function PIT_StartTimer() starts the timer counting. After calling this function, the timer loads the period value set earlier via the PIT_SetPeriod() function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function PIT_StopTimer() stops the timer counting.

### 32.2.4 Status

Provides functions to get and clear the PIT status.

### 32.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 32.3 Typical use case

### 32.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pit

## Data Structures

- struct pit_config_t
  *PIT configuration structure.* *More...*

## Enumerations

- enum pit_chnl_t {
  kPIT_Chnl_0 = 0U,
  kPIT_Chnl_1,
  kPIT_Chnl_2,
  kPIT_Chnl_3 }
  *List of PIT channels.*
- enum pit_interrupt_enable_t { kPIT_TimerInterruptEnable = PIT_TCTRL_TIE_MASK }
  *List of PIT interrupts.*
- enum pit_status_flags_t { kPIT_TimerFlag = PIT_TFLG_TIF_MASK }
  *List of PIT status flags.*

## Functions

- uint64_t PIT_GetLifetimeTimerCount (PIT_Type *base)
  *Reads the current lifetime counter value.*

## Driver version

- #define FSL_PIT_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *PIT Driver Version 2.0.2.*

## Initialization and deinitialization

- void PIT_Init (PIT_Type *base, const pit_config_t *config)

*Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.*
- void PIT_Deinit (PIT_Type *base)
  *Gates the PIT clock and disables the PIT module.*
- static void PIT_GetDefaultConfig (pit_config_t *config)
  *Fills in the PIT configuration structure with the default settings.*
- static void PIT_SetTimerChainMode (PIT_Type *base, pit_chnl_t channel, bool enable)
  *Enables or disables chaining a timer with the previous timer.*

### Interrupt Interface

- static void PIT_EnableInterrupts (PIT_Type *base, pit_chnl_t channel, uint32_t mask)
  *Enables the selected PIT interrupts.*
- static void PIT_DisableInterrupts (PIT_Type *base, pit_chnl_t channel, uint32_t mask)
  *Disables the selected PIT interrupts.*
- static uint32_t PIT_GetEnabledInterrupts (PIT_Type *base, pit_chnl_t channel)
  *Gets the enabled PIT interrupts.*

### Status Interface

- static uint32_t PIT_GetStatusFlags (PIT_Type *base, pit_chnl_t channel)
  *Gets the PIT status flags.*
- static void PIT_ClearStatusFlags (PIT_Type *base, pit_chnl_t channel, uint32_t mask)
  *Clears the PIT status flags.*

### Read and Write the timer period

- static void PIT_SetTimerPeriod (PIT_Type *base, pit_chnl_t channel, uint32_t count)
  *Sets the timer period in units of count.*
- static uint32_t PIT_GetCurrentTimerCount (PIT_Type *base, pit_chnl_t channel)
  *Reads the current timer counting value.*

### Timer Start and Stop

- static void PIT_StartTimer (PIT_Type *base, pit_chnl_t channel)
  *Starts the timer counting.*
- static void PIT_StopTimer (PIT_Type *base, pit_chnl_t channel)
  *Stops the timer counting.*

## 32.4 Data Structure Documentation

### 32.4.1 struct pit_config_t

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the PIT_GetDefaultConfig() function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

### Data Fields

- bool enableRunInDebug
  *true: Timers run in debug mode; false: Timers stop in debug mode*

## 32.5 Enumeration Type Documentation

### 32.5.1 enum pit_chnl_t

Note

Actual number of available channels is SoC dependent

Enumerator

***kPIT_Chnl_0*** PIT channel number 0.
***kPIT_Chnl_1*** PIT channel number 1.
***kPIT_Chnl_2*** PIT channel number 2.
***kPIT_Chnl_3*** PIT channel number 3.

### 32.5.2 enum pit_interrupt_enable_t

Enumerator

***kPIT_TimerInterruptEnable*** Timer interrupt enable.

### 32.5.3 enum pit_status_flags_t

Enumerator

***kPIT_TimerFlag*** Timer flag.

## 32.6 Function Documentation

### 32.6.1 void PIT_Init ( PIT_Type * *base*, const pit_config_t * *config* )

Note

This API should be called at the beginning of the application using the PIT driver.

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *config* | Pointer to the user's PIT config structure |

### 32.6.2 void PIT_Deinit ( PIT_Type * *base* )

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |

### 32.6.3 static void PIT_GetDefaultConfig ( pit_config_t * *config* ) [inline], [static]

The default values are as follows.

```
*     config->enableRunInDebug = false;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to the configuration structure. |

### 32.6.4 static void PIT_SetTimerChainMode ( PIT_Type * *base,* pit_chnl_t *channel,* bool *enable* ) [inline], [static]

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |

| | |
|---|---|
| *channel* | Timer channel number which is chained with the previous timer |
| *enable* | Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers. |

### 32.6.5 static void PIT_EnableInterrupts ( PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration pit_interrupt_enable_t |

### 32.6.6 static void PIT_DisableInterrupts ( PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number |
| *mask* | The interrupts to disable. This is a logical OR of members of the enumeration pit_interrupt_enable_t |

### 32.6.7 static uint32_t PIT_GetEnabledInterrupts ( PIT_Type * *base*, pit_chnl_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number |

Returns

The enabled interrupts. This is the logical OR of members of the enumeration pit_interrupt_enable_t

### 32.6.8 static uint32_t PIT_GetStatusFlags ( PIT_Type * *base,* pit_chnl_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number |

Returns

The status flags. This is the logical OR of members of the enumeration pit_status_flags_t

### 32.6.9 static void PIT_ClearStatusFlags ( PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number |
| *mask* | The status flags to clear. This is a logical OR of members of the enumeration pit_status_flags_t |

### 32.6.10 static void PIT_SetTimerPeriod ( PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *count* ) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load 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

Users can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number |

| | |
|---|---|
| *count* | Timer period in units of ticks |

### 32.6.11 static uint32_t PIT_GetCurrentTimerCount ( PIT_Type * *base*, pit_chnl_t *channel* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

Users can call the utility macros provided in fsl_common.h to convert ticks to usec or msec.

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number |

Returns

Current timer counting value in ticks

### 32.6.12 static void PIT_StartTimer ( PIT_Type * *base*, pit_chnl_t *channel* ) [inline], [static]

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number. |

### 32.6.13 static void PIT_StopTimer ( PIT_Type * *base*, pit_chnl_t *channel* ) [inline], [static]

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |
| *channel* | Timer channel number. |

### 32.6.14 uint64_t PIT_GetLifetimeTimerCount ( PIT_Type * *base* )

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the PIT_SetTimerChainMode before using this timer. The period of lifetime timer is equal to the "period of timer 0 * period of timer 1". For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

| | |
|---|---|
| *base* | PIT peripheral base address |

Returns

Current lifetime timer value

# Chapter 33
# PMU: Power Management Unit

## 33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Power Management Unit (PMU) module of MCUXpresso SDK devices. The power management unit (PMU) is designed to simplify the external power interface. The power system can be split into the input power sources and their characteristics, the integrated power transforming and controlling elements, and the final load interconnection and requirements. By using the internal LDO regulators, the number of external supplies is greatly reduced.

The PMU driver provides the APIs to adjust the work condition of each regulator, and can gate the power of some modules.

## Enumerations

- enum _pmu_status_flags {
  kPMU_1P1RegulatorOutputOK = (1U << 0U),
  kPMU_1P1BrownoutOnOutput = (1U << 1U),
  kPMU_3P0RegulatorOutputOK = (1U << 2U),
  kPMU_3P0BrownoutOnOutput = (1U << 3U),
  kPMU_2P5RegulatorOutputOK = (1U << 4U),
  kPMU_2P5BrownoutOnOutput = (1U << 5U) }
  *Status flags.*
- enum pmu_1p1_weak_reference_source_t {
  kPMU_1P1WeakReferenceSourceAlt0 = 0U,
  kPMU_1P1WeakReferenceSourceAlt1 = 1U }
  *The source for the reference voltage of the weak 1P1 regulator.*
- enum pmu_3p0_vbus_voltage_source_t {
  kPMU_3P0VBusVoltageSourceAlt0 = 0U,
  kPMU_3P0VBusVoltageSourceAlt1 = 1U }
  *Input voltage source for LDO_3P0 from USB VBus.*
- enum pmu_core_reg_voltage_ramp_rate_t {
  kPMU_CoreRegVoltageRampRateFast = 0U,
  kPMU_CoreRegVoltageRampRateMediumFast = 1U,
  kPMU_CoreRegVoltageRampRateMediumSlow = 2U,
  kPMU_CoreRegVoltageRampRateSlow = 0U }
  *Regulator voltage ramp rate.*
- enum pmu_power_bandgap_t {
  kPMU_NormalPowerBandgap = 0U,
  kPMU_LowPowerBandgap = 1U }
  *Bandgap select.*

## Driver version

- #define FSL_PMU_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *PMU driver version.*

## Status.

- uint32_t PMU_GetStatusFlags (PMU_Type *base)
  *Get PMU status flags.*

## 1P1 Regular

- static void PMU_1P1SetWeakReferenceSource (PMU_Type *base, pmu_1p1_weak_reference_source_t option)
  *Selects the source for the reference voltage of the weak 1P1 regulator.*
- static void PMU_1P1EnableWeakRegulator (PMU_Type *base, bool enable)
  *Enables the weak 1P1 regulator.*
- static void PMU_1P1SetRegulatorOutputVoltage (PMU_Type *base, uint32_t value)
  *Adjust the 1P1 regulator output voltage.*
- static void PMU_1P1SetBrownoutOffsetVoltage (PMU_Type *base, uint32_t value)
  *Adjust the 1P1 regulator brownout offset voltage.*
- static void PMU_1P1EnablePullDown (PMU_Type *base, bool enable)
  *Enable the pull-down circuitry in the regulator.*
- static void PMU_1P1EnableCurrentLimit (PMU_Type *base, bool enable)
  *Enable the current-limit circuitry in the regulator.*
- static void PMU_1P1EnableBrownout (PMU_Type *base, bool enable)
  *Enable the brownout circuitry in the regulator.*
- static void PMU_1P1EnableOutput (PMU_Type *base, bool enable)
  *Enable the regulator output.*

## 3P0 Regular

- static void PMU_3P0SetRegulatorOutputVoltage (PMU_Type *base, uint32_t value)
  *Adjust the 3P0 regulator output voltage.*
- static void PMU_3P0SetVBusVoltageSource (PMU_Type *base, pmu_3p0_vbus_voltage_source_t option)
  *Select input voltage source for LDO_3P0.*
- static void PMU_3P0SetBrownoutOffsetVoltage (PMU_Type *base, uint32_t value)
  *Adjust the 3P0 regulator brownout offset voltage.*
- static void PMU_3P0EnableCurrentLimit (PMU_Type *base, bool enable)
  *Enable the current-limit circuitry in the 3P0 regulator.*
- static void PMU_3P0EnableBrownout (PMU_Type *base, bool enable)
  *Enable the brownout circuitry in the 3P0 regulator.*
- static void PMU_3P0EnableOutput (PMU_Type *base, bool enable)
  *Enable the 3P0 regulator output.*

## 2P5 Regulator

- static void PMU_2P5EnableWeakRegulator (PMU_Type *base, bool enable)
  *Enables the weak 2P5 regulator.*
- static void PMU_2P5SetRegulatorOutputVoltage (PMU_Type *base, uint32_t value)

*Adjust the 1P1 regulator output voltage.*
- static void PMU_2P5SetBrownoutOffsetVoltage (PMU_Type *base, uint32_t value)
  *Adjust the 2P5 regulator brownout offset voltage.*
- static void PMU_2P5EnablePullDown (PMU_Type *base, bool enable)
  *Enable the pull-down circuitry in the 2P5 regulator.*
- static void PMU_2P1EnablePullDown (PMU_Type *base, bool enable)
  *Enable the pull-down circuitry in the 2P5 regulator.*
- static void PMU_2P5EnableCurrentLimit (PMU_Type *base, bool enable)
  *Enable the current-limit circuitry in the 2P5 regulator.*
- static void PMU_2P5nableBrownout (PMU_Type *base, bool enable)
  *Enable the brownout circuitry in the 2P5 regulator.*
- static void PMU_2P5EnableOutput (PMU_Type *base, bool enable)
  *Enable the 2P5 regulator output.*

### Core Regulator

- static void PMU_CoreEnableIncreaseGateDrive (PMU_Type *base, bool enable)
  *Increase the gate drive on power gating FETs.*
- static void PMU_CoreSetRegulatorVoltageRampRate (PMU_Type *base, pmu_core_reg_voltage_-ramp_rate_t option)
  *Set the CORE regulator voltage ramp rate.*
- static void PMU_CoreSetSOCDomainVoltage (PMU_Type *base, uint32_t value)
  *Define the target voltage for the SOC power domain.*
- static void PMU_CoreSetARMCoreDomainVoltage (PMU_Type *base, uint32_t value)
  *Define the target voltage for the ARM Core power domain.*

## 33.2 Macro Definition Documentation

### 33.2.1 #define FSL_PMU_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

Version 2.0.0.

## 33.3 Enumeration Type Documentation

### 33.3.1 enum _pmu_status_flags

Enumerator

***kPMU_1P1RegulatorOutputOK*** Status bit that signals when the 1p1 regulator output is ok. 1 = regulator output > brownout target.

***kPMU_1P1BrownoutOnOutput*** Status bit that signals when a 1p1 brownout is detected on the regulator output.

***kPMU_3P0RegulatorOutputOK*** Status bit that signals when the 3p0 regulator output is ok. 1 = regulator output > brownout target.

***kPMU_3P0BrownoutOnOutput*** Status bit that signals when a 3p0 brownout is detected on the regulator output.

***kPMU_2P5RegulatorOutputOK*** Status bit that signals when the 2p5 regulator output is ok. 1 = regulator output > brownout target.

***kPMU_2P5BrownoutOnOutput*** Status bit that signals when a 2p5 brownout is detected on the regulator output.

### 33.3.2 enum pmu_1p1_weak_reference_source_t

Enumerator

***kPMU_1P1WeakReferenceSourceAlt0*** Weak-linreg output tracks low-power-bandgap voltage.
***kPMU_1P1WeakReferenceSourceAlt1*** Weak-linreg output tracks VDD_SOC_CAP voltage.

### 33.3.3 enum pmu_3p0_vbus_voltage_source_t

Enumerator

***kPMU_3P0VBusVoltageSourceAlt0*** USB_OTG1_VBUS - Utilize VBUS OTG1 for power.
***kPMU_3P0VBusVoltageSourceAlt1*** USB_OTG2_VBUS - Utilize VBUS OTG2 for power.

### 33.3.4 enum pmu_core_reg_voltage_ramp_rate_t

Enumerator

***kPMU_CoreRegVoltageRampRateFast*** Fast.
***kPMU_CoreRegVoltageRampRateMediumFast*** Medium Fast.
***kPMU_CoreRegVoltageRampRateMediumSlow*** Medium Slow.
***kPMU_CoreRegVoltageRampRateSlow*** Slow.

### 33.3.5 enum pmu_power_bandgap_t

Enumerator

***kPMU_NormalPowerBandgap*** Normal power bandgap.
***kPMU_LowPowerBandgap*** Low power bandgap.

## 33.4 Function Documentation

### 33.4.1 uint32_t PMU_GetStatusFlags ( PMU_Type * *base* )

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |

Returns

PMU status flags.It indicate if regulator output of 1P1,3P0 and 2P5 is ok and brownout output of 1P1,3P0 and 2P5 is detected.

### 33.4.2 static void PMU_1P1SetWeakReferenceSource ( PMU_Type * *base*, pmu_1p1_weak_reference_source_t *option* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *option* | The option for reference voltage source, see to pmu_1p1_weak_reference_source_t. |

### 33.4.3 static void PMU_1P1EnableWeakRegulator ( PMU_Type * *base*, bool *enable* ) [inline], [static]

This regulator can be used when the main 1P1 regulator is disabled, under low-power conditions.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.4 static void PMU_1P1SetRegulatorOutputVoltage ( PMU_Type * *base*, uint32_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x1b(1.375V) >= output_trg >= 0x04(0.8V)
- 0x04 : 0.8V
- 0x10 : 1.1V (typical)
- 0x1b : 1.375V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for the output. |

### 33.4.5 static void PMU_1P1SetBrownoutOffsetVoltage ( PMU_Type * *base,* uint32_t *value* ) [inline], [static]

Control bits to adjust the regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT_TRG - BO_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for the brownout offset. The available range is in 3-bit. |

### 33.4.6 static void PMU_1P1EnablePullDown ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.7 static void PMU_1P1EnableCurrentLimit ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.8 static void PMU_1P1EnableBrownout ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.9 static void PMU_1P1EnableOutput ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.10 static void PMU_3P0SetRegulatorOutputVoltage ( PMU_Type * *base,* uint32_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.625V) >= output_trg >= 0x1f(3.4V)
- 0x00 : 2.625V
- 0x0f : 3.0V (typical)
- 0x1f : 3.4V

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for the output. |

### 33.4.11 static void PMU_3P0SetVBusVoltageSource ( PMU_Type * *base,* pmu_3p0_vbus_voltage_source_t *option* ) [inline], [static]

Select input voltage source for LDO_3P0 from either USB_OTG1_VBUS or USB_OTG2_VBUS. If only one of the two VBUS voltages is present, it is automatically selected.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *option* | User-defined input voltage source for LDO_3P0. |

### 33.4.12 static void PMU_3P0SetBrownoutOffsetVoltage ( PMU_Type * *base,* uint32_t *value* ) [inline], [static]

Control bits to adjust the 3P0 regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT_TRG - BO_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for the brownout offset. The available range is in 3-bit. |

### 33.4.13 static void PMU_3P0EnableCurrentLimit ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.14 static void PMU_3P0EnableBrownout ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.15 static void PMU_3P0EnableOutput ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.16 static void PMU_2P5EnableWeakRegulator ( PMU_Type * *base*, bool *enable* ) [inline], [static]

This low power regulator is used when the main 2P5 regulator is disabled to keep the 2.5V output roughly at 2.5V. Scales directly with the value of VDDHIGH_IN.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.17 static void PMU_2P5SetRegulatorOutputVoltage ( PMU_Type * *base*, uint32_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.1V) >= output_trg >= 0x1f(2.875V)
- 0x00 : 2.1V
- 0x10 : 2.5V (typical)
- 0x1f : 2.875V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for the output. |

### 33.4.18 static void PMU_2P5SetBrownoutOffsetVoltage ( PMU_Type * *base*, uint32_t *value* ) [inline], [static]

Adjust the regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT_TRG - BO_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for the brownout offset. The available range is in 3-bit. |

### 33.4.19 static void PMU_2P5EnablePullDown ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.20 static void PMU_2P1EnablePullDown ( PMU_Type * *base,* bool *enable* ) [inline], [static]

### 33.4.21 static void PMU_2P5EnableCurrentLimit ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.22 static void PMU_2P5nableBrownout ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.23 static void PMU_2P5EnableOutput ( PMU_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.24 static void PMU_CoreEnableIncreaseGateDrive ( PMU_Type * *base*, bool *enable* ) [inline], [static]

If set, increases the gate drive on power gating FETs to reduce leakage in the off state. Care must be taken to apply this bit only when the input supply voltage to the power FET is less than 1.1V. NOTE: This bit should only be used in low-power modes where the external input supply voltage is nominally 0.9V.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *enable* | Enable the feature or not. |

### 33.4.25 static void PMU_CoreSetRegulatorVoltageRampRate ( PMU_Type * *base*, pmu_core_reg_voltage_ramp_rate_t *option* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *option* | User-defined option for voltage ramp rate, see to pmu_core_reg_voltage_ramp_rate_t. |

### 33.4.26 static void PMU_CoreSetSOCDomainVoltage ( PMU_Type * *base*, uint32_t *value* ) [inline], [static]

Define the target voltage for the SOC power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V

- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for target voltage. 5-bit available |

### 33.4.27 static void PMU_CoreSetARMCoreDomainVoltage ( PMU_Type * *base*, uint32_t *value* ) [inline], [static]

Define the target voltage for the ARM Core power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V
- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

Parameters

| | |
|---|---|
| *base* | PMU peripheral base address. |
| *value* | Setting value for target voltage. 5-bit available |

# Chapter 34
# PWM: Pulse Width Modulator

## 34.1 Overview

The MCUXpresso SDK provides a driver for the Pulse Width Modulator (PWM) of MCUXpresso SDK devices.

The function PWM_Init() initializes the PWM sub module with specified configurations, the function PWM_GetDefaultConfig() could help to get the default configurations. The initialization function configures the sub module for the requested register update mode for registers with buffers. It also sets up the sub module operation in debug and wait modes.

The function PWM_SetupPwm() sets up PWM channels for PWM output, the function can set up PWM signal properties for multiple channels. The PWM has 2 channels: A and B. Each channel has its own duty cycle and level-mode specified, however the same PWM period and PWM mode is applied to all channels requesting PWM output. The signal duty cycle is provided as a percentage of the PWM period, its value should be between 0 and 100; 0=inactive signal(0% duty cycle) and 100=always active signal (100% duty cycle). The function also sets up the channel dead time value which is used when the user selects complementary mode of operation.

The function PWM_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular PWM channel.

The function PWM_SetupInputCapture() sets up a PWM channel for input capture. The user can specify the capture edge and the mode; one-shot capture or free-running capture.

The function PWM_SetupFault() sets up the properties for each fault.

The function PWM_StartTimer() can be used to start one or multiple sub modules. The function PWM_StopTimer() can be used to stop one or multiple sub modules.

Provide functions to get and clear the PWM status.

Provide functions to enable/disable PWM interrupts and get current enabled interrupts.

## 34.2 Register Update

Some of the PWM registers have buffers, the driver support various methods to update these registers with the content of the register buffer. The update mechanism for register with buffers can be specified through the following fields available in the configuration structure. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pwmThe user can select one of the reload options provided in enumeration pwm_register_reload_t. When using immediate reload, the reloadFrequency field is not used.

The driver initialization function sets up the appropriate bits in the PWM module based on the register update options selected.

The below function should be used to initiate a register reload. The example shows register reload ini-

tiated on PWM sub modules 0, 1, and 2. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pwm

## 34.3 Typical use case

### 34.3.1 PWM output

Output PWM signal on 3 PWM sub module with different dutycycles. Periodically update the PWM signal duty cycle. Each sub module runs in Complementary output mode with PWM A used to generate the complementary PWM pair. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pwm

## Data Structures

- struct pwm_signal_param_t
  *Structure for the user to define the PWM signal characteristics. More...*
- struct pwm_config_t
  *PWM config structure. More...*
- struct pwm_fault_param_t
  *Structure is used to hold the parameters to configure a PWM fault. More...*
- struct pwm_input_capture_param_t
  *Structure is used to hold parameters to configure the capture capability of a signal pin. More...*

## Macros

- #define PWM_SUBMODULE_SWCONTROL_WIDTH 2
  *Number of bits per submodule for software output control.*

## Enumerations

- enum pwm_submodule_t {
  kPWM_Module_0 = 0U,
  kPWM_Module_1,
  kPWM_Module_2,
  kPWM_Module_3 }
  *List of PWM submodules.*
- enum pwm_channels_t
  *List of PWM channels in each module.*
- enum pwm_value_register_t {
  kPWM_ValueRegister_0 = 0U,
  kPWM_ValueRegister_1,
  kPWM_ValueRegister_2,
  kPWM_ValueRegister_3,
  kPWM_ValueRegister_4,
  kPWM_ValueRegister_5 }
  *List of PWM value registers.*

- enum pwm_clock_source_t {
  kPWM_BusClock = 0U,
  kPWM_ExternalClock,
  kPWM_Submodule0Clock }
  *PWM clock source selection.*
- enum pwm_clock_prescale_t {
  kPWM_Prescale_Divide_1 = 0U,
  kPWM_Prescale_Divide_2,
  kPWM_Prescale_Divide_4,
  kPWM_Prescale_Divide_8,
  kPWM_Prescale_Divide_16,
  kPWM_Prescale_Divide_32,
  kPWM_Prescale_Divide_64,
  kPWM_Prescale_Divide_128 }
  *PWM prescaler factor selection for clock source.*
- enum pwm_force_output_trigger_t {
  kPWM_Force_Local = 0U,
  kPWM_Force_Master,
  kPWM_Force_LocalReload,
  kPWM_Force_MasterReload,
  kPWM_Force_LocalSync,
  kPWM_Force_MasterSync,
  kPWM_Force_External,
  kPWM_Force_ExternalSync }
  *Options that can trigger a PWM FORCE_OUT.*
- enum pwm_init_source_t {
  kPWM_Initialize_LocalSync = 0U,
  kPWM_Initialize_MasterReload,
  kPWM_Initialize_MasterSync,
  kPWM_Initialize_ExtSync }
  *PWM counter initialization options.*
- enum pwm_load_frequency_t {

kPWM_LoadEveryOportunity = 0U,
kPWM_LoadEvery2Oportunity,
kPWM_LoadEvery3Oportunity,
kPWM_LoadEvery4Oportunity,
kPWM_LoadEvery5Oportunity,
kPWM_LoadEvery6Oportunity,
kPWM_LoadEvery7Oportunity,
kPWM_LoadEvery8Oportunity,
kPWM_LoadEvery9Oportunity,
kPWM_LoadEvery10Oportunity,
kPWM_LoadEvery11Oportunity,
kPWM_LoadEvery12Oportunity,
kPWM_LoadEvery13Oportunity,
kPWM_LoadEvery14Oportunity,
kPWM_LoadEvery15Oportunity,
kPWM_LoadEvery16Oportunity }

*PWM load frequency selection.*

- enum pwm_fault_input_t {
kPWM_Fault_0 = 0U,
kPWM_Fault_1,
kPWM_Fault_2,
kPWM_Fault_3 }

  *List of PWM fault selections.*
- enum pwm_input_capture_edge_t {
kPWM_Disable = 0U,
kPWM_FallingEdge,
kPWM_RisingEdge,
kPWM_RiseAndFallEdge }

  *PWM capture edge select.*
- enum pwm_force_signal_t {
kPWM_UsePwm = 0U,
kPWM_InvertedPwm,
kPWM_SoftwareControl,
kPWM_UseExternal }

  *PWM output options when a FORCE_OUT signal is asserted.*
- enum pwm_chnl_pair_operation_t {
kPWM_Independent = 0U,
kPWM_ComplementaryPwmA,
kPWM_ComplementaryPwmB }

  *Options available for the PWM A & B pair operation.*
- enum pwm_register_reload_t {
kPWM_ReloadImmediate = 0U,
kPWM_ReloadPwmHalfCycle,
kPWM_ReloadPwmFullCycle,
kPWM_ReloadPwmHalfAndFullCycle }

  *Options available on how to load the buffered-registers with new values.*

- enum pwm_fault_recovery_mode_t {
  kPWM_NoRecovery = 0U,
  kPWM_RecoverHalfCycle,
  kPWM_RecoverFullCycle,
  kPWM_RecoverHalfAndFullCycle }
  *Options available on how to re-enable the PWM output when recovering from a fault.*
- enum pwm_interrupt_enable_t {
  kPWM_CompareVal0InterruptEnable = (1U << 0),
  kPWM_CompareVal1InterruptEnable = (1U << 1),
  kPWM_CompareVal2InterruptEnable = (1U << 2),
  kPWM_CompareVal3InterruptEnable = (1U << 3),
  kPWM_CompareVal4InterruptEnable = (1U << 4),
  kPWM_CompareVal5InterruptEnable = (1U << 5),
  kPWM_CaptureX0InterruptEnable = (1U << 6),
  kPWM_CaptureX1InterruptEnable = (1U << 7),
  kPWM_CaptureB0InterruptEnable = (1U << 8),
  kPWM_CaptureB1InterruptEnable = (1U << 9),
  kPWM_CaptureA0InterruptEnable = (1U << 10),
  kPWM_CaptureA1InterruptEnable = (1U << 11),
  kPWM_ReloadInterruptEnable = (1U << 12),
  kPWM_ReloadErrorInterruptEnable = (1U << 13),
  kPWM_Fault0InterruptEnable = (1U << 16),
  kPWM_Fault1InterruptEnable = (1U << 17),
  kPWM_Fault2InterruptEnable = (1U << 18),
  kPWM_Fault3InterruptEnable = (1U << 19) }
  *List of PWM interrupt options.*
- enum pwm_status_flags_t {
  kPWM_CompareVal0Flag = (1U << 0),
  kPWM_CompareVal1Flag = (1U << 1),
  kPWM_CompareVal2Flag = (1U << 2),
  kPWM_CompareVal3Flag = (1U << 3),
  kPWM_CompareVal4Flag = (1U << 4),
  kPWM_CompareVal5Flag = (1U << 5),
  kPWM_CaptureX0Flag = (1U << 6),
  kPWM_CaptureX1Flag = (1U << 7),
  kPWM_CaptureB0Flag = (1U << 8),
  kPWM_CaptureB1Flag = (1U << 9),
  kPWM_CaptureA0Flag = (1U << 10),
  kPWM_CaptureA1Flag = (1U << 11),
  kPWM_ReloadFlag = (1U << 12),
  kPWM_ReloadErrorFlag = (1U << 13),
  kPWM_RegUpdatedFlag = (1U << 14),
  kPWM_Fault0Flag = (1U << 16),
  kPWM_Fault1Flag = (1U << 17),
  kPWM_Fault2Flag = (1U << 18),

kPWM_Fault3Flag = (1U << 19) }
*List of PWM status flags.*
- enum pwm_mode_t {
kPWM_SignedCenterAligned = 0U,
kPWM_CenterAligned,
kPWM_SignedEdgeAligned,
kPWM_EdgeAligned }
*PWM operation mode.*
- enum pwm_level_select_t {
kPWM_HighTrue = 0U,
kPWM_LowTrue }
*PWM output pulse mode, high-true or low-true.*
- enum pwm_reload_source_select_t {
kPWM_LocalReload = 0U,
kPWM_MasterReload }
*PWM reload source select.*
- enum pwm_fault_clear_t {
kPWM_Automatic = 0U,
kPWM_ManualNormal,
kPWM_ManualSafety }
*PWM fault clearing options.*
- enum pwm_module_control_t {
kPWM_Control_Module_0 = (1U << 0),
kPWM_Control_Module_1 = (1U << 1),
kPWM_Control_Module_2 = (1U << 2),
kPWM_Control_Module_3 = (1U << 3) }
*Options for submodule master control operation.*

## Functions

- void PWM_SetupInputCapture (PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, const pwm_input_capture_param_t *inputCaptureParams)
*Sets up the PWM input capture.*
- void PWM_SetupFaults (PWM_Type *base, pwm_fault_input_t faultNum, const pwm_fault_param_t *faultParams)
*Sets up the PWM fault protection.*
- void PWM_SetupForceSignal (PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, pwm_force_signal_t mode)
*Selects the signal to output on a PWM pin when a FORCE_OUT signal is asserted.*
- static void PWM_OutputTriggerEnable (PWM_Type *base, pwm_submodule_t subModule, pwm_value_register_t valueRegister, bool activate)
*Enables or disables the PWM output trigger.*
- static void PWM_SetupSwCtrlOut (PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, bool value)
*Sets the software control output for a pin to high or low.*
- static void PWM_SetPwmLdok (PWM_Type *base, uint8_t subModulesToUpdate, bool value)
*Sets or clears the PWM LDOK bit on a single or multiple submodules.*

## Driver version

- #define FSL_PWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *Version 2.0.0.*

## Initialization and deinitialization

- status_t PWM_Init (PWM_Type *base, pwm_submodule_t subModule, const pwm_config_t *config)
  *Ungates the PWM submodule clock and configures the peripheral for basic operation.*
- void PWM_Deinit (PWM_Type *base, pwm_submodule_t subModule)
  *Gate the PWM submodule clock.*
- void PWM_GetDefaultConfig (pwm_config_t *config)
  *Fill in the PWM config struct with the default settings.*

## Module PWM output

- status_t PWM_SetupPwm (PWM_Type *base, pwm_submodule_t subModule, const pwm_signal_param_t *chnlParams, uint8_t numOfChnls, pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)
  *Sets up the PWM signals for a PWM submodule.*
- void PWM_UpdatePwmDutycycle (PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t pwmSignal, pwm_mode_t currPwmMode, uint8_t dutyCyclePercent)
  *Updates the PWM signal's dutycycle.*

## Interrupts Interface

- void PWM_EnableInterrupts (PWM_Type *base, pwm_submodule_t subModule, uint32_t mask)
  *Enables the selected PWM interrupts.*
- void PWM_DisableInterrupts (PWM_Type *base, pwm_submodule_t subModule, uint32_t mask)
  *Disables the selected PWM interrupts.*
- uint32_t PWM_GetEnabledInterrupts (PWM_Type *base, pwm_submodule_t subModule)
  *Gets the enabled PWM interrupts.*

## Status Interface

- uint32_t PWM_GetStatusFlags (PWM_Type *base, pwm_submodule_t subModule)
  *Gets the PWM status flags.*
- void PWM_ClearStatusFlags (PWM_Type *base, pwm_submodule_t subModule, uint32_t mask)
  *Clears the PWM status flags.*

## Timer Start and Stop

- static void PWM_StartTimer (PWM_Type *base, uint8_t subModulesToStart)
  *Starts the PWM counter for a single or multiple submodules.*
- static void PWM_StopTimer (PWM_Type *base, uint8_t subModulesToStop)
  *Stops the PWM counter for a single or multiple submodules.*

## 34.4 Data Structure Documentation

### 34.4.1 struct pwm_signal_param_t

#### Data Fields

- pwm_channels_t pwmChannel
  *PWM channel being configured; PWM A or PWM B.*
- uint8_t dutyCyclePercent
  *PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*
- pwm_level_select_t level
  *PWM output active level select.*
- uint16_t deadtimeValue
  *The deadtime value; only used if channel pair is operating in complementary mode.*

##### 34.4.1.0.0.29 Field Documentation

###### 34.4.1.0.0.29.1 uint8_t pwm_signal_param_t::dutyCyclePercent

100=always active signal (100% duty cycle)

### 34.4.2 struct pwm_config_t

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the PWM_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

#### Data Fields

- bool enableDebugMode
  *true: PWM continues to run in debug mode; false: PWM is paused in debug mode*
- bool enableWait
  *true: PWM continues to run in WAIT mode; false: PWM is paused in WAIT mode*
- uint8_t faultFilterCount
  *Fault filter count.*
- uint8_t faultFilterPeriod
  *Fault filter period;value of 0 will bypass the filter.*
- pwm_init_source_t initializationControl
  *Option to initialize the counter.*
- pwm_clock_source_t clockSource
  *Clock source for the counter.*
- pwm_clock_prescale_t prescale
  *Pre-scaler to divide down the clock.*
- pwm_chnl_pair_operation_t pairOperation
  *Channel pair in indepedent or complementary mode.*
- pwm_register_reload_t reloadLogic

*PWM Reload logic setup.*
- pwm_reload_source_select_t reloadSelect
  *Reload source select.*
- pwm_load_frequency_t reloadFrequency
  *Specifies when to reload, used when user's choice is not immediate reload.*
- pwm_force_output_trigger_t forceTrigger
  *Specify which signal will trigger a FORCE_OUT.*

### 34.4.3 struct pwm_fault_param_t

## Data Fields

- pwm_fault_clear_t faultClearingMode
  *Fault clearing mode to use.*
- bool faultLevel
  *true: Logic 1 indicates fault; false: Logic 0 indicates fault*
- bool enableCombinationalPath
  *true: Combinational Path from fault input is enabled; false: No combination path is available*
- pwm_fault_recovery_mode_t recoverMode
  *Specify when to re-enable the PWM output.*

### 34.4.4 struct pwm_input_capture_param_t

## Data Fields

- bool captureInputSel
  *true: Use the edge counter signal as source false: Use the raw input signal from the pin as source*
- uint8_t edgeCompareValue
  *Compare value, used only if edge counter is used as source.*
- pwm_input_capture_edge_t edge0
  *Specify which edge causes a capture for input circuitry 0.*
- pwm_input_capture_edge_t edge1
  *Specify which edge causes a capture for input circuitry 1.*
- bool enableOneShotCapture
  *true: Use one-shot capture mode; false: Use free-running capture mode*
- uint8_t fifoWatermark
  *Watermark level for capture FIFO.*

#### 34.4.4.0.0.30 Field Documentation

##### 34.4.4.0.0.30.1 uint8_t pwm_input_capture_param_t::fifoWatermark

The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

## 34.5 Enumeration Type Documentation

### 34.5.1 enum pwm_submodule_t

Enumerator

***kPWM_Module_0*** Submodule 0.
***kPWM_Module_1*** Submodule 1.
***kPWM_Module_2*** Submodule 2.
***kPWM_Module_3*** Submodule 3.

### 34.5.2 enum pwm_value_register_t

Enumerator

***kPWM_ValueRegister_0*** PWM Value0 register.
***kPWM_ValueRegister_1*** PWM Value1 register.
***kPWM_ValueRegister_2*** PWM Value2 register.
***kPWM_ValueRegister_3*** PWM Value3 register.
***kPWM_ValueRegister_4*** PWM Value4 register.
***kPWM_ValueRegister_5*** PWM Value5 register.

### 34.5.3 enum pwm_clock_source_t

Enumerator

***kPWM_BusClock*** The IPBus clock is used as the clock.
***kPWM_ExternalClock*** EXT_CLK is used as the clock.
***kPWM_Submodule0Clock*** Clock of the submodule 0 (AUX_CLK) is used as the source clock.

### 34.5.4 enum pwm_clock_prescale_t

Enumerator

***kPWM_Prescale_Divide_1*** PWM clock frequency = fclk/1.
***kPWM_Prescale_Divide_2*** PWM clock frequency = fclk/2.
***kPWM_Prescale_Divide_4*** PWM clock frequency = fclk/4.
***kPWM_Prescale_Divide_8*** PWM clock frequency = fclk/8.
***kPWM_Prescale_Divide_16*** PWM clock frequency = fclk/16.
***kPWM_Prescale_Divide_32*** PWM clock frequency = fclk/32.
***kPWM_Prescale_Divide_64*** PWM clock frequency = fclk/64.
***kPWM_Prescale_Divide_128*** PWM clock frequency = fclk/128.

### 34.5.5 enum pwm_force_output_trigger_t

Enumerator

***kPWM_Force_Local*** The local force signal, CTRL2[FORCE], from the submodule is used to force updates.
***kPWM_Force_Master*** The master force signal from submodule 0 is used to force updates.
***kPWM_Force_LocalReload*** The local reload signal from this submodule is used to force updates without regard to the state of LDOK.
***kPWM_Force_MasterReload*** The master reload signal from submodule 0 is used to force updates if LDOK is set.
***kPWM_Force_LocalSync*** The local sync signal from this submodule is used to force updates.
***kPWM_Force_MasterSync*** The master sync signal from submodule0 is used to force updates.
***kPWM_Force_External*** The external force signal, EXT_FORCE, from outside the PWM module causes updates.
***kPWM_Force_ExternalSync*** The external sync signal, EXT_SYNC, from outside the PWM module causes updates.

### 34.5.6 enum pwm_init_source_t

Enumerator

***kPWM_Initialize_LocalSync*** Local sync causes initialization.
***kPWM_Initialize_MasterReload*** Master reload from submodule 0 causes initialization.
***kPWM_Initialize_MasterSync*** Master sync from submodule 0 causes initialization.
***kPWM_Initialize_ExtSync*** EXT_SYNC causes initialization.

### 34.5.7 enum pwm_load_frequency_t

Enumerator

***kPWM_LoadEveryOportunity*** Every PWM opportunity.
***kPWM_LoadEvery2Oportunity*** Every 2 PWM opportunities.
***kPWM_LoadEvery3Oportunity*** Every 3 PWM opportunities.
***kPWM_LoadEvery4Oportunity*** Every 4 PWM opportunities.
***kPWM_LoadEvery5Oportunity*** Every 5 PWM opportunities.
***kPWM_LoadEvery6Oportunity*** Every 6 PWM opportunities.
***kPWM_LoadEvery7Oportunity*** Every 7 PWM opportunities.
***kPWM_LoadEvery8Oportunity*** Every 8 PWM opportunities.
***kPWM_LoadEvery9Oportunity*** Every 9 PWM opportunities.
***kPWM_LoadEvery10Oportunity*** Every 10 PWM opportunities.
***kPWM_LoadEvery11Oportunity*** Every 11 PWM opportunities.
***kPWM_LoadEvery12Oportunity*** Every 12 PWM opportunities.

***kPWM_LoadEvery13Oportunity*** Every 13 PWM opportunities.
***kPWM_LoadEvery14Oportunity*** Every 14 PWM opportunities.
***kPWM_LoadEvery15Oportunity*** Every 15 PWM opportunities.
***kPWM_LoadEvery16Oportunity*** Every 16 PWM opportunities.

### 34.5.8 enum pwm_fault_input_t

Enumerator

***kPWM_Fault_0*** Fault 0 input pin.
***kPWM_Fault_1*** Fault 1 input pin.
***kPWM_Fault_2*** Fault 2 input pin.
***kPWM_Fault_3*** Fault 3 input pin.

### 34.5.9 enum pwm_input_capture_edge_t

Enumerator

***kPWM_Disable*** Disabled.
***kPWM_FallingEdge*** Capture on falling edge only.
***kPWM_RisingEdge*** Capture on rising edge only.
***kPWM_RiseAndFallEdge*** Capture on rising or falling edge.

### 34.5.10 enum pwm_force_signal_t

Enumerator

***kPWM_UsePwm*** Generated PWM signal is used by the deadtime logic.
***kPWM_InvertedPwm*** Inverted PWM signal is used by the deadtime logic.
***kPWM_SoftwareControl*** Software controlled value is used by the deadtime logic.
***kPWM_UseExternal*** PWM_EXTA signal is used by the deadtime logic.

### 34.5.11 enum pwm_chnl_pair_operation_t

Enumerator

***kPWM_Independent*** PWM A & PWM B operate as 2 independent channels.
***kPWM_ComplementaryPwmA*** PWM A & PWM B are complementary channels, PWM A generates the signal.
***kPWM_ComplementaryPwmB*** PWM A & PWM B are complementary channels, PWM B generates the signal.

### 34.5.12 enum pwm_register_reload_t

Enumerator

***kPWM_ReloadImmediate*** Buffered-registers get loaded with new values as soon as LDOK bit is set.
***kPWM_ReloadPwmHalfCycle*** Registers loaded on a PWM half cycle.
***kPWM_ReloadPwmFullCycle*** Registers loaded on a PWM full cycle.
***kPWM_ReloadPwmHalfAndFullCycle*** Registers loaded on a PWM half & full cycle.

### 34.5.13 enum pwm_fault_recovery_mode_t

Enumerator

***kPWM_NoRecovery*** PWM output will stay inactive.
***kPWM_RecoverHalfCycle*** PWM output re-enabled at the first half cycle.
***kPWM_RecoverFullCycle*** PWM output re-enabled at the first full cycle.
***kPWM_RecoverHalfAndFullCycle*** PWM output re-enabled at the first half or full cycle.

### 34.5.14 enum pwm_interrupt_enable_t

Enumerator

***kPWM_CompareVal0InterruptEnable*** PWM VAL0 compare interrupt.
***kPWM_CompareVal1InterruptEnable*** PWM VAL1 compare interrupt.
***kPWM_CompareVal2InterruptEnable*** PWM VAL2 compare interrupt.
***kPWM_CompareVal3InterruptEnable*** PWM VAL3 compare interrupt.
***kPWM_CompareVal4InterruptEnable*** PWM VAL4 compare interrupt.
***kPWM_CompareVal5InterruptEnable*** PWM VAL5 compare interrupt.
***kPWM_CaptureX0InterruptEnable*** PWM capture X0 interrupt.
***kPWM_CaptureX1InterruptEnable*** PWM capture X1 interrupt.
***kPWM_CaptureB0InterruptEnable*** PWM capture B0 interrupt.
***kPWM_CaptureB1InterruptEnable*** PWM capture B1 interrupt.
***kPWM_CaptureA0InterruptEnable*** PWM capture A0 interrupt.
***kPWM_CaptureA1InterruptEnable*** PWM capture A1 interrupt.
***kPWM_ReloadInterruptEnable*** PWM reload interrupt.
***kPWM_ReloadErrorInterruptEnable*** PWM reload error interrupt.
***kPWM_Fault0InterruptEnable*** PWM fault 0 interrupt.
***kPWM_Fault1InterruptEnable*** PWM fault 1 interrupt.
***kPWM_Fault2InterruptEnable*** PWM fault 2 interrupt.
***kPWM_Fault3InterruptEnable*** PWM fault 3 interrupt.

### 34.5.15 enum pwm_status_flags_t

Enumerator

***kPWM_CompareVal0Flag*** PWM VAL0 compare flag.
***kPWM_CompareVal1Flag*** PWM VAL1 compare flag.
***kPWM_CompareVal2Flag*** PWM VAL2 compare flag.
***kPWM_CompareVal3Flag*** PWM VAL3 compare flag.
***kPWM_CompareVal4Flag*** PWM VAL4 compare flag.
***kPWM_CompareVal5Flag*** PWM VAL5 compare flag.
***kPWM_CaptureX0Flag*** PWM capture X0 flag.
***kPWM_CaptureX1Flag*** PWM capture X1 flag.
***kPWM_CaptureB0Flag*** PWM capture B0 flag.
***kPWM_CaptureB1Flag*** PWM capture B1 flag.
***kPWM_CaptureA0Flag*** PWM capture A0 flag.
***kPWM_CaptureA1Flag*** PWM capture A1 flag.
***kPWM_ReloadFlag*** PWM reload flag.
***kPWM_ReloadErrorFlag*** PWM reload error flag.
***kPWM_RegUpdatedFlag*** PWM registers updated flag.
***kPWM_Fault0Flag*** PWM fault 0 flag.
***kPWM_Fault1Flag*** PWM fault 1 flag.
***kPWM_Fault2Flag*** PWM fault 2 flag.
***kPWM_Fault3Flag*** PWM fault 3 flag.

### 34.5.16 enum pwm_mode_t

Enumerator

***kPWM_SignedCenterAligned*** Signed center-aligned.
***kPWM_CenterAligned*** Unsigned cente-aligned.
***kPWM_SignedEdgeAligned*** Signed edge-aligned.
***kPWM_EdgeAligned*** Unsigned edge-aligned.

### 34.5.17 enum pwm_level_select_t

Enumerator

***kPWM_HighTrue*** High level represents "on" or "active" state.
***kPWM_LowTrue*** Low level represents "on" or "active" state.

### 34.5.18 enum pwm_reload_source_select_t

Enumerator

***kPWM_LocalReload*** The local reload signal is used to reload registers.
***kPWM_MasterReload*** The master reload signal (from submodule 0) is used to reload.

### 34.5.19 enum pwm_fault_clear_t

Enumerator

***kPWM_Automatic*** Automatic fault clearing.
***kPWM_ManualNormal*** Manual fault clearing with no fault safety mode.
***kPWM_ManualSafety*** Manual fault clearing with fault safety mode.

### 34.5.20 enum pwm_module_control_t

Enumerator

***kPWM_Control_Module_0*** Control submodule 0's start/stop,buffer reload operation.
***kPWM_Control_Module_1*** Control submodule 1's start/stop,buffer reload operation.
***kPWM_Control_Module_2*** Control submodule 2's start/stop,buffer reload operation.
***kPWM_Control_Module_3*** Control submodule 3's start/stop,buffer reload operation.

## 34.6 Function Documentation

### 34.6.1 status_t PWM_Init ( PWM_Type * *base*, pwm_submodule_t *subModule*, const pwm_config_t * *config* )

Note

This API should be called at the beginning of the application using the PWM driver.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |

| | |
|---|---|
| *config* | Pointer to user's PWM config structure. |

Returns

kStatus_Success means success; else failed.

### 34.6.2 void PWM_Deinit ( PWM_Type * *base*, pwm_submodule_t *subModule* )

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to deinitialize |

### 34.6.3 void PWM_GetDefaultConfig ( pwm_config_t * *config* )

The default values are:

```
*   config->enableDebugMode = false;
*   config->enableWait = false;
*   config->reloadSelect = kPWM_LocalReload;
*   config->faultFilterCount = 0;
*   config->faultFilterPeriod = 0;
*   config->clockSource = kPWM_BusClock;
*   config->prescale = kPWM_Prescale_Divide_1;
*   config->initializationControl = kPWM_Initialize_LocalSync;
*   config->forceTrigger = kPWM_Force_Local;
*   config->reloadFrequency = kPWM_LoadEveryOportunity;
*   config->reloadLogic = kPWM_ReloadImmediate;
*   config->pairOperation = kPWM_Independent;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to user's PWM config structure. |

### 34.6.4 status_t PWM_SetupPwm ( PWM_Type * *base*, pwm_submodule_t *subModule*, const pwm_signal_param_t * *chnlParams*, uint8_t *numOfChnls*, pwm_mode_t *mode*, uint32_t *pwmFreq_Hz*, uint32_t *srcClock_Hz* )

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *chnlParams* | Array of PWM channel parameters to configure the channel(s) |
| *numOfChnls* | Number of channels to configure, this should be the size of the array passed in. Array size should not be more than 2 as each submodule has 2 pins to output PWM |
| *mode* | PWM operation mode, options available in enumeration pwm_mode_t |
| *pwmFreq_Hz* | PWM signal frequency in Hz |
| *srcClock_Hz* | PWM main counter clock in Hz. |

Returns

Returns kStatusFail if there was error setting up the signal; kStatusSuccess otherwise

### 34.6.5 void PWM_UpdatePwmDutycycle ( PWM_Type * *base*, pwm_submodule_t *subModule*, pwm_channels_t *pwmSignal*, pwm_mode_t *currPwmMode*, uint8_t *dutyCyclePercent* )

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *pwmSignal* | Signal (PWM A or PWM B) to update |
| *currPwmMode* | The current PWM mode set during PWM setup |
| *dutyCycle-Percent* | New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

### 34.6.6 void PWM_SetupInputCapture ( PWM_Type * *base*, pwm_submodule_t *subModule*, pwm_channels_t *pwmChannel*, const pwm_input_capture_param_t * *inputCaptureParams* )

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *pwmChannel* | Channel in the submodule to setup |
| *inputCapture-Params* | Parameters passed in to set up the input pin |

### 34.6.7 void PWM_SetupFaults ( PWM_Type * *base*, pwm_fault_input_t *faultNum*, const pwm_fault_param_t * *faultParams* )

PWM has 4 fault inputs.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *faultNum* | PWM fault to configure. |
| *faultParams* | Pointer to the PWM fault config structure |

### 34.6.8 void PWM_SetupForceSignal ( PWM_Type * *base*, pwm_submodule_t *subModule*, pwm_channels_t *pwmChannel*, pwm_force_signal_t *mode* )

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *pwmChannel* | Channel to configure |
| *mode* | Signal to output when a FORCE_OUT is triggered |

### 34.6.9 void PWM_EnableInterrupts ( PWM_Type * *base*, pwm_submodule_t *subModule*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t |

### 34.6.10 void PWM_DisableInterrupts ( PWM_Type * *base*, pwm_submodule_t *subModule*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t |

### 34.6.11 uint32_t PWM_GetEnabledInterrupts ( PWM_Type * *base*, pwm_submodule_t *subModule* )

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |

Returns

The enabled interrupts. This is the logical OR of members of the enumeration pwm_interrupt_enable_t

### 34.6.12 uint32_t PWM_GetStatusFlags ( PWM_Type * *base*, pwm_submodule_t *subModule* )

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |

Returns

The status flags. This is the logical OR of members of the enumeration pwm_status_flags_t

### 34.6.13 void PWM_ClearStatusFlags ( PWM_Type * *base*, pwm_submodule_t *subModule*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *mask* | The status flags to clear. This is a logical OR of members of the enumeration pwm_status_flags_t |

### 34.6.14 static void PWM_StartTimer ( PWM_Type * *base*, uint8_t *subModulesToStart* ) [inline], [static]

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModulesToStart* | PWM submodules to start. This is a logical OR of members of the enumeration pwm_module_control_t |

### 34.6.15 static void PWM_StopTimer ( PWM_Type * *base*, uint8_t *subModulesToStop* ) [inline], [static]

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModulesToStop* | PWM submodules to stop. This is a logical OR of members of the enumeration pwm_module_control_t |

### 34.6.16 static void PWM_OutputTriggerEnable ( PWM_Type * *base*, pwm_submodule_t *subModule*, pwm_value_register_t *valueRegister*, bool *activate* ) [inline], [static]

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *valueRegister* | Value register that will activate the trigger |
| *activate* | true: Enable the trigger; false: Disable the trigger |

### 34.6.17 static void PWM_SetupSwCtrlOut ( PWM_Type * *base*, pwm_submodule_t *subModule*, pwm_channels_t *pwmChannel*, bool *value* ) [inline], [static]

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModule* | PWM submodule to configure |
| *pwmChannel* | Channel to configure |
| *value* | true: Supply a logic 1, false: Supply a logic 0. |

### 34.6.18 static void PWM_SetPwmLdok ( PWM_Type * *base*, uint8_t *subModulesToUpdate*, bool *value* ) [inline], [static]

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if kPWM_ReloadImmediate option was choosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

Parameters

| | |
|---|---|
| *base* | PWM peripheral base address |
| *subModulesToUpdate* | PWM submodules to update with buffered values. This is a logical OR of members of the enumeration pwm_module_control_t |
| *value* | true: Set LDOK bit for the submodule list; false: Clear LDOK bit |

# Chapter 35
# PXP: Pixel Pipeline

## 35.1 Overview

The MCUXpresso SDK provides a driver for the Pixel Pipeline (PXP)

The PXP is used to process graphics buffers or composite video and graphics data before sending to an LCD display or TV encoder. The PXP driver only provides functional APIs. It does not maintain software level state, so that the APIs could be involved directly to any upper layer graphics framework easily.

To use the PXP driver, call PXP_Init first to enable and initialize the peripheral. Generally, call the PXP driver APIs the configure input buffer, output buffer, and other setting such as flip, rotate, then call PXP_Start, thus the PXP starts the processing. When finished, the flag kPXP_CompleteFlag asserts. PXP also supports operation queuing, it means that a new operation could be submited to PXP while the current PXP operation is running. When current operation finished, the new operation configurations are loaded to PXP register and new processing starts.

## 35.2 Typical use case

### 35.2.1 PXP normal operation

This example shows how to perform vertical flip to process surface and save to output buffer. The input and output buffer pixel format are RGB888.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pxp

### 35.2.2 PXP operation queue

This example shows how to perform vertical flip to process surface using operation queue. The input and output buffer pixel format are RGB888.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pxp

## Data Structures

- struct pxp_output_buffer_config_t
  *PXP output buffer configuration. More...*
- struct pxp_ps_buffer_config_t
  *PXP process surface buffer configuration. More...*
- struct pxp_as_buffer_config_t
  *PXP alphs surface buffer configuration. More...*
- struct pxp_as_blend_config_t
  *PXP alpha surface blending configuration. More...*
- struct pxp_csc2_config_t

*PXP CSC2 configuration.* More...

- struct pxp_dither_final_lut_data_t
  *PXP dither final LUT data.* More...
- struct pxp_dither_config_t
  *PXP dither configuration.* More...

## Enumerations

- enum _pxp_interrupt_enable {
  kPXP_CommandLoadInterruptEnable = PXP_CTRL_NEXT_IRQ_ENABLE_MASK,
  kPXP_CompleteInterruptEnable = PXP_CTRL_IRQ_ENABLE_MASK }
  *PXP interrupts to enable.*
- enum _pxp_flags {
  kPXP_CommandLoadFlag = PXP_STAT_NEXT_IRQ_MASK,
  kPXP_CompleteFlag = PXP_STAT_IRQ0_MASK,
  kPXP_Axi0ReadErrorFlag = PXP_STAT_AXI_READ_ERROR_0_MASK,
  kPXP_Axi0WriteErrorFlag = PXP_STAT_AXI_WRITE_ERROR_0_MASK }
  *PXP status flags.*
- enum pxp_flip_mode_t {
  kPXP_FlipDisable = 0U,
  kPXP_FlipHorizontal = 0x01U,
  kPXP_FlipVertical = 0x02U,
  kPXP_FlipBoth = 0x03U }
  *PXP output flip mode.*
- enum pxp_rotate_position_t {
  kPXP_RotateOutputBuffer = 0U,
  kPXP_RotateProcessSurface }
  *PXP rotate mode.*
- enum pxp_rotate_degree_t {
  kPXP_Rotate0 = 0U,
  kPXP_Rotate90,
  kPXP_Rotate180,
  kPXP_Rotate270 }
  *PXP rotate degree.*
- enum pxp_interlaced_output_mode_t {
  kPXP_OutputProgressive = 0U,
  kPXP_OutputField0,
  kPXP_OutputField1,
  kPXP_OutputInterlaced }
  *PXP interlaced output mode.*
- enum pxp_output_pixel_format_t {

kPXP_OutputPixelFormatARGB8888 = 0x0,
kPXP_OutputPixelFormatRGB888 = 0x4,
kPXP_OutputPixelFormatRGB888P = 0x5,
kPXP_OutputPixelFormatARGB1555 = 0x8,
kPXP_OutputPixelFormatARGB4444 = 0x9,
kPXP_OutputPixelFormatRGB555 = 0xC,
kPXP_OutputPixelFormatRGB444 = 0xD,
kPXP_OutputPixelFormatRGB565 = 0xE,
kPXP_OutputPixelFormatYUV1P444 = 0x10,
kPXP_OutputPixelFormatUYVY1P422 = 0x12,
kPXP_OutputPixelFormatVYUY1P422 = 0x13,
kPXP_OutputPixelFormatY8 = 0x14,
kPXP_OutputPixelFormatY4 = 0x15,
kPXP_OutputPixelFormatYUV2P422 = 0x18,
kPXP_OutputPixelFormatYUV2P420 = 0x19,
kPXP_OutputPixelFormatYVU2P422 = 0x1A,
kPXP_OutputPixelFormatYVU2P420 = 0x1B }

*PXP output buffer format.*

- enum pxp_ps_pixel_format_t {
  kPXP_PsPixelFormatRGB888 = 0x4,
  kPXP_PsPixelFormatRGB555 = 0xC,
  kPXP_PsPixelFormatRGB444 = 0xD,
  kPXP_PsPixelFormatRGB565 = 0xE,
  kPXP_PsPixelFormatYUV1P444 = 0x10,
  kPXP_PsPixelFormatUYVY1P422 = 0x12,
  kPXP_PsPixelFormatVYUY1P422 = 0x13,
  kPXP_PsPixelFormatY8 = 0x14,
  kPXP_PsPixelFormatY4 = 0x15,
  kPXP_PsPixelFormatYUV2P422 = 0x18,
  kPXP_PsPixelFormatYUV2P420 = 0x19,
  kPXP_PsPixelFormatYVU2P422 = 0x1A,
  kPXP_PsPixelFormatYVU2P420 = 0x1B,
  kPXP_PsPixelFormatYVU422 = 0x1E,
  kPXP_PsPixelFormatYVU420 = 0x1F }

  *PXP process surface buffer pixel format.*
- enum pxp_as_pixel_format_t {
  kPXP_AsPixelFormatARGB8888 = 0x0,
  kPXP_AsPixelFormatRGB888 = 0x4,
  kPXP_AsPixelFormatARGB1555 = 0x8,
  kPXP_AsPixelFormatARGB4444 = 0x9,
  kPXP_AsPixelFormatRGB555 = 0xC,
  kPXP_AsPixelFormatRGB444 = 0xD,
  kPXP_AsPixelFormatRGB565 = 0xE }

  *PXP alpha surface buffer pixel format.*
- enum pxp_alpha_mode_t {

kPXP_AlphaEmbedded,
kPXP_AlphaOverride,
kPXP_AlphaMultiply,
kPXP_AlphaRop }
*PXP alpha mode during blending.*

- enum pxp_rop_mode_t {
kPXP_RopMaskAs = 0x0,
kPXP_RopMaskNotAs = 0x1,
kPXP_RopMaskAsNot = 0x2,
kPXP_RopMergeAs = 0x3,
kPXP_RopMergeNotAs = 0x4,
kPXP_RopMergeAsNot = 0x5,
kPXP_RopNotCopyAs = 0x6,
kPXP_RopNot = 0x7,
kPXP_RopNotMaskAs = 0x8,
kPXP_RopNotMergeAs = 0x9,
kPXP_RopXorAs = 0xA,
kPXP_RopNotXorAs = 0xB }
*PXP ROP mode during blending.*
- enum pxp_block_size_t {
kPXP_BlockSize8 = 0U,
kPXP_BlockSize16 }
*PXP process block size.*
- enum pxp_csc1_mode_t {
kPXP_Csc1YUV2RGB = 0U,
kPXP_Csc1YCbCr2RGB }
*PXP CSC1 mode.*
- enum pxp_csc2_mode_t {
kPXP_Csc2YUV2RGB = 0U,
kPXP_Csc2YCbCr2RGB,
kPXP_Csc2RGB2YUV,
kPXP_Csc2RGB2YCbCr }
*PXP CSC2 mode.*
- enum pxp_ram_t {
kPXP_RamDither0Lut = 0U,
kPXP_RamDither1Lut = 3U,
kPXP_RamDither2Lut = 4U }
*PXP internal memory.*
- enum _pxp_dither_mode {
kPXP_DitherPassThrough = 0U,
kPXP_DitherOrdered = 3U,
kPXP_DitherQuantOnly = 4U }
*PXP dither mode.*
- enum _pxp_dither_lut_mode {
kPXP_DitherLutOff = 0U,
kPXP_DitherLutPreDither,

kPXP_DitherLutPostDither }
*PXP dither LUT mode.*
- enum _pxp_dither_matrix_size {
kPXP_DitherMatrix8 = 1,
kPXP_DitherMatrix16 }
*PXP dither matrix size.*

## Driver version

- #define FSL_PXP_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
*Version 2.0.1.*

## Initialization and deinitialization

- void PXP_Init (PXP_Type *base)
*Initialize the PXP.*
- void PXP_Deinit (PXP_Type *base)
*De-initialize the PXP.*
- void PXP_Reset (PXP_Type *base)
*Reset the PXP.*

## Global operations

- static void PXP_Start (PXP_Type *base)
*Start process.*
- static void PXP_EnableLcdHandShake (PXP_Type *base, bool enable)
*Enable or disable LCD hand shake.*
- static void PXP_EnableContinousRun (PXP_Type *base, bool enable)
*Enable or disable continous run.*
- static void PXP_SetProcessBlockSize (PXP_Type *base, pxp_block_size_t size)
*Set the PXP processing block size.*

## Status

- static uint32_t PXP_GetStatusFlags (PXP_Type *base)
*Gets PXP status flags.*
- static void PXP_ClearStatusFlags (PXP_Type *base, uint32_t statusMask)
*Clears status flags with the provided mask.*
- static uint8_t PXP_GetAxiErrorId (PXP_Type *base, uint8_t axiIndex)
*Gets the AXI ID of the failing bus operation.*

## Interrupts

- static void PXP_EnableInterrupts (PXP_Type *base, uint32_t mask)
*Enables PXP interrupts according to the provided mask.*
- static void PXP_DisableInterrupts (PXP_Type *base, uint32_t mask)
*Disables PXP interrupts according to the provided mask.*

**Typical use case**

## Alpha surface

- void PXP_SetAlphaSurfaceBufferConfig (PXP_Type *base, const pxp_as_buffer_config_t *config)
  *Set the alpha surface input buffer configuration.*
- void PXP_SetAlphaSurfaceBlendConfig (PXP_Type *base, const pxp_as_blend_config_t *config)
  *Set the alpha surface blending configuration.*
- void PXP_SetAlphaSurfaceOverlayColorKey (PXP_Type *base, uint32_t colorKeyLow, uint32_t colorKeyHigh)
  *Set the alpha surface overlay color key.*
- static void PXP_EnableAlphaSurfaceOverlayColorKey (PXP_Type *base, bool enable)
  *Enable or disable the alpha surface color key.*
- void PXP_SetAlphaSurfacePosition (PXP_Type *base, uint16_t upperLeftX, uint16_t upperLeftY, uint16_t lowerRightX, uint16_t lowerRightY)
  *Set the alpha surface position in output buffer.*

## Process surface

- static void PXP_SetProcessSurfaceBackGroundColor (PXP_Type *base, uint32_t backGroundColor)
  *Set the back ground color of PS.*
- void PXP_SetProcessSurfaceBufferConfig (PXP_Type *base, const pxp_ps_buffer_config_t *config)
  *Set the process surface input buffer configuration.*
- void PXP_SetProcessSurfaceScaler (PXP_Type *base, uint16_t inputWidth, uint16_t inputHeight, uint16_t outputWidth, uint16_t outputHeight)
  *Set the process surface scaler configuration.*
- void PXP_SetProcessSurfacePosition (PXP_Type *base, uint16_t upperLeftX, uint16_t upperLeftY, uint16_t lowerRightX, uint16_t lowerRightY)
  *Set the process surface position in output buffer.*
- void PXP_SetProcessSurfaceColorKey (PXP_Type *base, uint32_t colorKeyLow, uint32_t colorKeyHigh)
  *Set the process surface color key.*

## Output buffer

- void PXP_SetOutputBufferConfig (PXP_Type *base, const pxp_output_buffer_config_t *config)
  *Set the PXP outpt buffer configuration.*
- static void PXP_SetOverwrittenAlphaValue (PXP_Type *base, uint8_t alpha)
  *Set the global overwritten alpha value.*
- static void PXP_EnableOverWrittenAlpha (PXP_Type *base, bool enable)
  *Enable or disable the global overwritten alpha value.*
- static void PXP_SetRotateConfig (PXP_Type *base, pxp_rotate_position_t position, pxp_rotate_degree_t degree, pxp_flip_mode_t flipMode)
  *Set the rotation configuration.*

## Command queue

- static void PXP_SetNextCommand (PXP_Type *base, void *commandAddr)
  *Set the next command.*
- static bool PXP_IsNextCommandPending (PXP_Type *base)

*Check whether the next command is pending.*
- static void PXP_CancelNextCommand (PXP_Type *base)
  *Cancel command set by PXP_SetNextCommand.*

### Color space conversion

- void PXP_SetCsc1Mode (PXP_Type *base, pxp_csc1_mode_t mode)
  *Set the CSC1 mode.*
- static void PXP_EnableCsc1 (PXP_Type *base, bool enable)
  *Enable or disable the CSC1.*

## 35.3 Data Structure Documentation

### 35.3.1 struct pxp_output_buffer_config_t

### Data Fields

- pxp_output_pixel_format_t pixelFormat
  *Output buffer pixel format.*
- pxp_interlaced_output_mode_t interlacedMode
  *Interlaced output mode.*
- uint32_t buffer0Addr
  *Output buffer 0 address.*
- uint32_t buffer1Addr
  *Output buffer 1 address, used for UV data in YUV 2-plane mode, or field 1 in output interlaced mode.*
- uint16_t pitchBytes
  *Number of bytes between two vertically adjacent pixels.*
- uint16_t width
  *Pixels per line.*
- uint16_t height
  *How many lines in output buffer.*

#### 35.3.1.0.0.31 Field Documentation

#### 35.3.1.0.0.31.1 pxp_output_pixel_format_t pxp_output_buffer_config_t::pixelFormat

#### 35.3.1.0.0.31.2 pxp_interlaced_output_mode_t pxp_output_buffer_config_t::interlacedMode

#### 35.3.1.0.0.31.3 uint32_t pxp_output_buffer_config_t::buffer0Addr

#### 35.3.1.0.0.31.4 uint32_t pxp_output_buffer_config_t::buffer1Addr

#### 35.3.1.0.0.31.5 uint16_t pxp_output_buffer_config_t::pitchBytes

#### 35.3.1.0.0.31.6 uint16_t pxp_output_buffer_config_t::width

#### 35.3.1.0.0.31.7 uint16_t pxp_output_buffer_config_t::height

### 35.3.2 struct pxp_ps_buffer_config_t

## Data Fields

- pxp_ps_pixel_format_t pixelFormat
  *PS buffer pixel format.*
- bool swapByte
  *For each 16 bit word, set true to swap the two bytes.*
- uint32_t bufferAddr
  *Input buffer address for the first panel.*
- uint32_t bufferAddrU
  *Input buffer address for the second panel.*
- uint32_t bufferAddrV
  *Input buffer address for the third panel.*
- uint16_t pitchBytes
  *Number of bytes between two vertically adjacent pixels.*

#### 35.3.2.0.0.32 Field Documentation

##### 35.3.2.0.0.32.1 pxp_ps_pixel_format_t pxp_ps_buffer_config_t::pixelFormat

##### 35.3.2.0.0.32.2 bool pxp_ps_buffer_config_t::swapByte

##### 35.3.2.0.0.32.3 uint32_t pxp_ps_buffer_config_t::bufferAddr

##### 35.3.2.0.0.32.4 uint32_t pxp_ps_buffer_config_t::bufferAddrU

##### 35.3.2.0.0.32.5 uint32_t pxp_ps_buffer_config_t::bufferAddrV

##### 35.3.2.0.0.32.6 uint16_t pxp_ps_buffer_config_t::pitchBytes

### 35.3.3 struct pxp_as_buffer_config_t

## Data Fields

- pxp_as_pixel_format_t pixelFormat
  *AS buffer pixel format.*
- uint32_t bufferAddr
  *Input buffer address.*
- uint16_t pitchBytes
  *Number of bytes between two vertically adjacent pixels.*

#### 35.3.3.0.0.33 Field Documentation

##### 35.3.3.0.0.33.1 pxp_as_pixel_format_t pxp_as_buffer_config_t::pixelFormat

##### 35.3.3.0.0.33.2 uint32_t pxp_as_buffer_config_t::bufferAddr

##### 35.3.3.0.0.33.3 uint16_t pxp_as_buffer_config_t::pitchBytes

### 35.3.4 struct pxp_as_blend_config_t

## Data Fields

- uint8_t alpha
  *User defined alpha value, only used when alphaMode is kPXP_AlphaOverride or kPXP_AlphaRop.*
- bool invertAlpha
  *Set true to invert the alpha.*
- pxp_alpha_mode_t alphaMode
  *Alpha mode.*
- pxp_rop_mode_t ropMode
  *ROP mode, only valid when alphaMode is kPXP_AlphaRop.*

#### 35.3.4.0.0.34 Field Documentation

##### 35.3.4.0.0.34.1 uint8_t pxp_as_blend_config_t::alpha

##### 35.3.4.0.0.34.2 bool pxp_as_blend_config_t::invertAlpha

##### 35.3.4.0.0.34.3 pxp_alpha_mode_t pxp_as_blend_config_t::alphaMode

##### 35.3.4.0.0.34.4 pxp_rop_mode_t pxp_as_blend_config_t::ropMode

### 35.3.5 struct pxp_csc2_config_t

Converting from YUV/YCbCr color spaces to the RGB color space uses the following equation structure:

R = A1(Y+D1) + A2(U+D2) + A3(V+D3) G = B1(Y+D1) + B2(U+D2) + B3(V+D3) B = C1(Y+D1) + C2(U+D2) + C3(V+D3)

Converting from the RGB color space to YUV/YCbCr color spaces uses the following equation structure:

Y = A1∗R + A2∗G + A3∗B + D1 U = B1∗R + B2∗G + B3∗B + D2 V = C1∗R + C2∗G + C3∗B + D3

### Data Fields

- pxp_csc2_mode_t mode
  *Convertion mode.*
- float A1
  *A1.*
- float A2
  *A2.*
- float A3
  *A3.*
- float B1
  *B1.*
- float B2
  *B2.*
- float B3
  *B3.*
- float C1
  *C1.*
- float C2
  *C2.*
- float C3
  *C3.*
- int16_t D1
  *D1.*
- int16_t D2
  *D2.*
- int16_t D3
  *D3.*

#### 35.3.5.0.0.35 Field Documentation

##### 35.3.5.0.0.35.1 pxp_csc2_mode_t pxp_csc2_config_t::mode

##### 35.3.5.0.0.35.2 float pxp_csc2_config_t::A1

##### 35.3.5.0.0.35.3 float pxp_csc2_config_t::A2

##### 35.3.5.0.0.35.4 float pxp_csc2_config_t::A3

##### 35.3.5.0.0.35.5 float pxp_csc2_config_t::B1

##### 35.3.5.0.0.35.6 float pxp_csc2_config_t::B2

##### 35.3.5.0.0.35.7 float pxp_csc2_config_t::B3

##### 35.3.5.0.0.35.8 float pxp_csc2_config_t::C1

##### 35.3.5.0.0.35.9 float pxp_csc2_config_t::C2

##### 35.3.5.0.0.35.10 float pxp_csc2_config_t::C3

##### 35.3.5.0.0.35.11 int16_t pxp_csc2_config_t::D1

##### 35.3.5.0.0.35.12 int16_t pxp_csc2_config_t::D2

##### 35.3.5.0.0.35.13 int16_t pxp_csc2_config_t::D3

### 35.3.6 struct pxp_dither_final_lut_data_t

## Data Fields

- uint32_t data_3_0
  *Data 3 to data 0.*
- uint32_t data_7_4
  *Data 7 to data 4.*
- uint32_t data_11_8
  *Data 11 to data 8.*
- uint32_t data_15_12
  *Data 15 to data 12.*

#### 35.3.6.0.0.36 Field Documentation

##### 35.3.6.0.0.36.1 uint32_t pxp_dither_final_lut_data_t::data_3_0

Data 0 is the least significant byte.

##### 35.3.6.0.0.36.2 uint32_t pxp_dither_final_lut_data_t::data_7_4

Data 4 is the least significant byte.

#### 35.3.6.0.0.36.3 uint32_t pxp_dither_final_lut_data_t::data_11_8

Data 8 is the least significant byte.

#### 35.3.6.0.0.36.4 uint32_t pxp_dither_final_lut_data_t::data_15_12

Data 12 is the least significant byte.

### 35.3.7 struct pxp_dither_config_t

## Data Fields

- uint32_t enableDither0: 1
  *Enable dither engine 0 or not, set 1 to enable, 0 to disable.*
- uint32_t enableDither1: 1
  *Enable dither engine 1 or not, set 1 to enable, 0 to disable.*
- uint32_t enableDither2: 1
  *Enable dither engine 2 or not, set 1 to enable, 0 to disable.*
- uint32_t ditherMode0: 3
  *Dither mode for dither engine 0.*
- uint32_t ditherMode1: 3
  *Dither mode for dither engine 1.*
- uint32_t ditherMode2: 3
  *Dither mode for dither engine 2.*
- uint32_t quantBitNum: 3
  *Number of bits quantize down to, the valid value is 1~7.*
- uint32_t lutMode: 2
  *How to use the memory LUT, see _pxp_dither_lut_mode.*
- uint32_t idxMatrixSize0: 2
  *Size of index matrix used for dither for dither engine 0, see _pxp_dither_matrix_size.*
- uint32_t idxMatrixSize1: 2
  *Size of index matrix used for dither for dither engine 1, see _pxp_dither_matrix_size.*
- uint32_t idxMatrixSize2: 2
  *Size of index matrix used for dither for dither engine 2, see _pxp_dither_matrix_size.*
- uint32_t enableFinalLut: 1
  *Enable the final LUT, set 1 to enable, 0 to disable.*

#### 35.3.7.0.0.37 Field Documentation

#### 35.3.7.0.0.37.1 uint32_t pxp_dither_config_t::enableDither0

#### 35.3.7.0.0.37.2 uint32_t pxp_dither_config_t::enableDither1

#### 35.3.7.0.0.37.3 uint32_t pxp_dither_config_t::enableDither2

#### 35.3.7.0.0.37.4 uint32_t pxp_dither_config_t::ditherMode0

See _pxp_dither_mode.

#### 35.3.7.0.0.37.5 uint32_t pxp_dither_config_t::ditherMode1

See _pxp_dither_mode.

#### 35.3.7.0.0.37.6 uint32_t pxp_dither_config_t::ditherMode2

See _pxp_dither_mode.

#### 35.3.7.0.0.37.7 uint32_t pxp_dither_config_t::quantBitNum

#### 35.3.7.0.0.37.8 uint32_t pxp_dither_config_t::lutMode

This must be set to kPXP_DitherLutOff if any dither engine uses kPXP_DitherOrdered mode.

#### 35.3.7.0.0.37.9 uint32_t pxp_dither_config_t::idxMatrixSize0

#### 35.3.7.0.0.37.10 uint32_t pxp_dither_config_t::idxMatrixSize1

#### 35.3.7.0.0.37.11 uint32_t pxp_dither_config_t::idxMatrixSize2

#### 35.3.7.0.0.37.12 uint32_t pxp_dither_config_t::enableFinalLut

## 35.4 Enumeration Type Documentation

### 35.4.1 enum _pxp_interrupt_enable

Enumerator

***kPXP_CommandLoadInterruptEnable*** Interrupt to show that the command set by PXP_SetNextCommand has been loaded.

***kPXP_CompleteInterruptEnable*** PXP process completed.

### 35.4.2 enum _pxp_flags

Note

These enumerations are meant to be OR'd together to form a bit mask.

Enumerator

***kPXP_CommandLoadFlag*** The command set by PXP_SetNextCommand has been loaded, could set new command.

***kPXP_CompleteFlag*** PXP process completed.

***kPXP_Axi0ReadErrorFlag*** PXP encountered an AXI read error and processing has been terminated.

***kPXP_Axi0WriteErrorFlag*** PXP encountered an AXI write error and processing has been terminated.

### 35.4.3 enum pxp_flip_mode_t

Enumerator

***kPXP_FlipDisable*** Flip disable.
***kPXP_FlipHorizontal*** Horizontal flip.
***kPXP_FlipVertical*** Vertical flip.
***kPXP_FlipBoth*** Flip both directions.

### 35.4.4 enum pxp_rotate_position_t

Enumerator

***kPXP_RotateOutputBuffer*** Rotate the output buffer.
***kPXP_RotateProcessSurface*** Rotate the process surface.

### 35.4.5 enum pxp_rotate_degree_t

Enumerator

***kPXP_Rotate0*** Clock wise rotate 0 deg.
***kPXP_Rotate90*** Clock wise rotate 90 deg.
***kPXP_Rotate180*** Clock wise rotate 180 deg.
***kPXP_Rotate270*** Clock wise rotate 270 deg.

### 35.4.6 enum pxp_interlaced_output_mode_t

Enumerator

***kPXP_OutputProgressive*** All data written in progressive format to output buffer 0.
***kPXP_OutputField0*** Only write field 0 data to output buffer 0.
***kPXP_OutputField1*** Only write field 1 data to output buffer 0.
***kPXP_OutputInterlaced*** Field 0 write to buffer 0, field 1 write to buffer 1.

### 35.4.7 enum pxp_output_pixel_format_t

Enumerator

***kPXP_OutputPixelFormatARGB8888*** 32-bit pixels with alpha.
***kPXP_OutputPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)

***kPXP_OutputPixelFormatRGB888P*** 24-bit pixels without alpha (packed 24-bit format)
***kPXP_OutputPixelFormatARGB1555*** 16-bit pixels with alpha.
***kPXP_OutputPixelFormatARGB4444*** 16-bit pixels with alpha.
***kPXP_OutputPixelFormatRGB555*** 16-bit pixels without alpha.
***kPXP_OutputPixelFormatRGB444*** 16-bit pixels without alpha.
***kPXP_OutputPixelFormatRGB565*** 16-bit pixels without alpha.
***kPXP_OutputPixelFormatYUV1P444*** 32-bit pixels (1-plane XYUV unpacked).
***kPXP_OutputPixelFormatUYVY1P422*** 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)
***kPXP_OutputPixelFormatVYUY1P422*** 16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)
***kPXP_OutputPixelFormatY8*** 8-bit monochrome pixels (1-plane Y luma output)
***kPXP_OutputPixelFormatY4*** 4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)
***kPXP_OutputPixelFormatYUV2P422*** 16-bit pixels (2-plane UV interleaved bytes)
***kPXP_OutputPixelFormatYUV2P420*** 16-bit pixels (2-plane UV)
***kPXP_OutputPixelFormatYVU2P422*** 16-bit pixels (2-plane VU interleaved bytes)
***kPXP_OutputPixelFormatYVU2P420*** 16-bit pixels (2-plane VU)

### 35.4.8 enum pxp_ps_pixel_format_t

Enumerator

***kPXP_PsPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)
***kPXP_PsPixelFormatRGB555*** 16-bit pixels without alpha.
***kPXP_PsPixelFormatRGB444*** 16-bit pixels without alpha.
***kPXP_PsPixelFormatRGB565*** 16-bit pixels without alpha.
***kPXP_PsPixelFormatYUV1P444*** 32-bit pixels (1-plane XYUV unpacked).
***kPXP_PsPixelFormatUYVY1P422*** 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)
***kPXP_PsPixelFormatVYUY1P422*** 16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)
***kPXP_PsPixelFormatY8*** 8-bit monochrome pixels (1-plane Y luma output)
***kPXP_PsPixelFormatY4*** 4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)
***kPXP_PsPixelFormatYUV2P422*** 16-bit pixels (2-plane UV interleaved bytes)
***kPXP_PsPixelFormatYUV2P420*** 16-bit pixels (2-plane UV)
***kPXP_PsPixelFormatYVU2P422*** 16-bit pixels (2-plane VU interleaved bytes)
***kPXP_PsPixelFormatYVU2P420*** 16-bit pixels (2-plane VU)
***kPXP_PsPixelFormatYVU422*** 16-bit pixels (3-plane)
***kPXP_PsPixelFormatYVU420*** 16-bit pixels (3-plane)

### 35.4.9 enum pxp_as_pixel_format_t

Enumerator

***kPXP_AsPixelFormatARGB8888*** 32-bit pixels with alpha.
***kPXP_AsPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)

***kPXP_AsPixelFormatARGB1555*** 16-bit pixels with alpha.
***kPXP_AsPixelFormatARGB4444*** 16-bit pixels with alpha.
***kPXP_AsPixelFormatRGB555*** 16-bit pixels without alpha.
***kPXP_AsPixelFormatRGB444*** 16-bit pixels without alpha.
***kPXP_AsPixelFormatRGB565*** 16-bit pixels without alpha.

### 35.4.10 enum pxp_alpha_mode_t

Enumerator

***kPXP_AlphaEmbedded*** The alpha surface pixel alpha value will be used for blend.
***kPXP_AlphaOverride*** The user defined alpha value will be used for blend directly.
***kPXP_AlphaMultiply*** The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set set to 200, user defined alpha set to 100, then the reault alpha is 200 ∗ 100 / 255.
***kPXP_AlphaRop*** Raster operation.

### 35.4.11 enum pxp_rop_mode_t

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Enumerator

***kPXP_RopMaskAs*** AS AND PS.
***kPXP_RopMaskNotAs*** nAS AND PS.
***kPXP_RopMaskAsNot*** AS AND nPS.
***kPXP_RopMergeAs*** AS OR PS.
***kPXP_RopMergeNotAs*** nAS OR PS.
***kPXP_RopMergeAsNot*** AS OR nPS.
***kPXP_RopNotCopyAs*** nAS.
***kPXP_RopNot*** nPS.
***kPXP_RopNotMaskAs*** AS NAND PS.
***kPXP_RopNotMergeAs*** AS NOR PS.
***kPXP_RopXorAs*** AS XOR PS.
***kPXP_RopNotXorAs*** AS XNOR PS.

### 35.4.12 enum pxp_block_size_t

Enumerator

***kPXP_BlockSize8*** Process 8x8 pixel blocks.
***kPXP_BlockSize16*** Process 16x16 pixel blocks.

### 35.4.13 enum pxp_csc1_mode_t

Enumerator

***kPXP_Csc1YUV2RGB*** YUV to RGB.
***kPXP_Csc1YCbCr2RGB*** YCbCr to RGB.

### 35.4.14 enum pxp_csc2_mode_t

Enumerator

***kPXP_Csc2YUV2RGB*** YUV to RGB.
***kPXP_Csc2YCbCr2RGB*** YCbCr to RGB.
***kPXP_Csc2RGB2YUV*** RGB to YUV.
***kPXP_Csc2RGB2YCbCr*** RGB to YCbCr.

### 35.4.15 enum pxp_ram_t

Enumerator

***kPXP_RamDither0Lut*** Dither 0 LUT memory.
***kPXP_RamDither1Lut*** Dither 1 LUT memory.
***kPXP_RamDither2Lut*** Dither 2 LUT memory.

### 35.4.16 enum _pxp_dither_mode

Enumerator

***kPXP_DitherPassThrough*** Pass through, no dither.
***kPXP_DitherOrdered*** Ordered dither.
***kPXP_DitherQuantOnly*** No dithering, only quantization.

### 35.4.17 enum _pxp_dither_lut_mode

Enumerator

***kPXP_DitherLutOff*** The LUT memory is not used for LUT, could be used as ordered dither index matrix.

***kPXP_DitherLutPreDither*** Use LUT at the pre-dither stage, The pre-dither LUT could only be used in Floyd mode or Atkinson mode, which are not supported by current PXP module.

***kPXP_DitherLutPostDither*** Use LUT at the post-dither stage.

### 35.4.18 enum _pxp_dither_matrix_size

Enumerator

***kPXP_DitherMatrix8*** The dither index matrix is 8x8.

***kPXP_DitherMatrix16*** The dither index matrix is 16x16.

## 35.5 Function Documentation

### 35.5.1 void PXP_Init ( PXP_Type * *base* )

This function enables the PXP peripheral clock, and resets the PXP registers to default status.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |

### 35.5.2 void PXP_Deinit ( PXP_Type * *base* )

This function disables the PXP peripheral clock.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |

### 35.5.3 void PXP_Reset ( PXP_Type * *base* )

This function resets the PXP peripheral registers to default status.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |

### 35.5.4 static void PXP_Start ( PXP_Type * *base* ) [inline], [static]

Start PXP process using current configuration.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |

### 35.5.5 static void PXP_EnableLcdHandShake ( PXP_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *enable* | True to enable, false to disable. |

### 35.5.6 static void PXP_EnableContinousRun ( PXP_Type * *base*, bool *enable* ) [inline], [static]

If continous run not enabled, PXP_Start starts the PXP process. When completed, PXP enters idle mode and flag kPXP_CompleteFlag asserts.

If continous run enabled, the PXP will repeat based on the current configuration register settings.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *enable* | True to enable, false to disable. |

### 35.5.7 static void PXP_SetProcessBlockSize ( PXP_Type * *base*, pxp_block_size_t *size* ) [inline], [static]

This function chooses the pixel block size that PXP using during process. Larger block size means better performace, but be careful that when PXP is rotating, the output must be divisible by the block size

selected.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *size* | The pixel block size. |

### 35.5.8 static uint32_t PXP_GetStatusFlags ( PXP_Type * *base* ) [inline], [static]

This function gets all PXP status flags. The flags are returned as the logical OR value of the enumerators _pxp_flags. To check a specific status, compare the return value with enumerators in _pxp_flags. For example, to check whether the PXP has completed process, use like this:

```
if (kPXP_CompleteFlag & PXP_GetStatusFlags(PXP))
{
    ...
}
```

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |

Returns

PXP status flags which are OR'ed by the enumerators in the _pxp_flags.

### 35.5.9 static void PXP_ClearStatusFlags ( PXP_Type * *base*, uint32_t *statusMask* ) [inline], [static]

This function clears PXP status flags with a provided mask.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *statusMask* | The status flags to be cleared; it is logical OR value of _pxp_flags. |

### 35.5.10 static uint8_t PXP_GetAxiErrorId ( PXP_Type * *base*, uint8_t *axiIndex* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *axiIndex* | Whitch AXI to get<br>• 0: AXI0<br>• 1: AXI1 |

Returns

The AXI ID of the failing bus operation.

### 35.5.11 static void PXP_EnableInterrupts ( PXP_Type * *base*, uint32_t *mask* ) [inline], [static]

This function enables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _pxp_interrupt_enable. For example, to enable PXP process complete interrupt and command loaded interrupt, do the following.

```
PXP_EnableInterrupts(PXP, kPXP_CommandLoadInterruptEnable
      | kPXP_CompleteInterruptEnable);
```

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *mask* | The interrupts to enable. Logical OR of _pxp_interrupt_enable. |

### 35.5.12 static void PXP_DisableInterrupts ( PXP_Type * *base*, uint32_t *mask* ) [inline], [static]

This function disables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _pxp_interrupt_enable.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *mask* | The interrupts to disable. Logical OR of _pxp_interrupt_enable. |

### 35.5.13 void PXP_SetAlphaSurfaceBufferConfig ( PXP_Type * *base,* const pxp_as_buffer_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *config* | Pointer to the configuration. |

### 35.5.14 void PXP_SetAlphaSurfaceBlendConfig ( PXP_Type * *base,* const pxp_as_blend_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *config* | Pointer to the configuration structure. |

### 35.5.15 void PXP_SetAlphaSurfaceOverlayColorKey ( PXP_Type * *base,* uint32_t *colorKeyLow,* uint32_t *colorKeyHigh* )

If a pixel in the current overlay image with a color that falls in the range from the `colorKeyLow` to `colorKeyHigh` range, it will use the process surface pixel value for that location. If no PS image is present or if the PS image also matches its colorkey range, the PS background color is used.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *colorKeyLow* | Color key low range. |
| *colorKeyHigh* | Color key high range. |

Note

Colorkey operations are higher priority than alpha or ROP operations

### 35.5.16 static void PXP_EnableAlphaSurfaceOverlayColorKey ( PXP_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *enable* | True to enable, false to disable. |

### 35.5.17 void PXP_SetAlphaSurfacePosition ( PXP_Type * *base,* uint16_t *upperLeftX,* uint16_t *upperLeftY,* uint16_t *lowerRightX,* uint16_t *lowerRightY* )

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *upperLeftX* | X of the upper left corner. |
| *upperLeftY* | Y of the upper left corner. |
| *lowerRightX* | X of the lower right corner. |
| *lowerRightY* | Y of the lower right corner. |

### 35.5.18 static void PXP_SetProcessSurfaceBackGroundColor ( PXP_Type * *base,* uint32_t *backGroundColor* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *backGround-Color* | Pixel value of the background color. |

### 35.5.19 void PXP_SetProcessSurfaceBufferConfig ( PXP_Type * *base,* const pxp_ps_buffer_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *config* | Pointer to the configuration. |

### 35.5.20 void PXP_SetProcessSurfaceScaler ( PXP_Type * *base*, uint16_t *inputWidth*, uint16_t *inputHeight*, uint16_t *outputWidth*, uint16_t *outputHeight* )

The valid down scale fact is 1/(2^12) ~ 16.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *inputWidth* | Input image width. |
| *inputHeight* | Input image height. |
| *outputWidth* | Output image width. |
| *outputHeight* | Output image height. |

### 35.5.21 void PXP_SetProcessSurfacePosition ( PXP_Type * *base*, uint16_t *upperLeftX*, uint16_t *upperLeftY*, uint16_t *lowerRightX*, uint16_t *lowerRightY* )

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *upperLeftX* | X of the upper left corner. |
| *upperLeftY* | Y of the upper left corner. |
| *lowerRightX* | X of the lower right corner. |
| *lowerRightY* | Y of the lower right corner. |

### 35.5.22 void PXP_SetProcessSurfaceColorKey ( PXP_Type * *base*, uint32_t *colorKeyLow*, uint32_t *colorKeyHigh* )

If the PS image matches colorkey range, the PS background color is output. Set `colorKeyLow` to 0xFFFFFFFF and `colorKeyHigh` to 0 will disable the colorkeying.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *colorKeyLow* | Color key low range. |
| *colorKeyHigh* | Color key high range. |

### 35.5.23 void PXP_SetOutputBufferConfig ( PXP_Type * *base,* const pxp_output_buffer_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *config* | Pointer to the configuration. |

### 35.5.24 static void PXP_SetOverwrittenAlphaValue ( PXP_Type * *base,* uint8_t *alpha* ) [inline], [static]

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *alpha* | The alpha value. |

### 35.5.25 static void PXP_EnableOverWrittenAlpha ( PXP_Type * *base,* bool *enable* ) [inline], [static]

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *enable* | True to enable, false to disable. |

### 35.5.26 static void PXP_SetRotateConfig ( PXP_Type * *base*, pxp_rotate_position_t *position*, pxp_rotate_degree_t *degree*, pxp_flip_mode_t *flipMode* ) [inline], [static]

The PXP could rotate the process surface or the output buffer. There are two PXP versions:

- Version 1: Only has one rotate sub module, the output buffer and process surface share the same rotate sub module, which means the process surface and output buffer could not be rotate at the same time. When pass in kPXP_RotateOutputBuffer, the process surface could not use the rotate, Also when pass in kPXP_RotateProcessSurface, output buffer could not use the rotate.
- Version 2: Has two seperate rotate sub modules, the output buffer and process surface could configure the rotation independently.

Upper layer could use the macro PXP_SHARE_ROTATE to check which version is. PXP_SHARE_ROTATE=1 means version 1.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *position* | Rotate process surface or output buffer. |
| *degree* | Rotate degree. |
| *flipMode* | Flip mode. |

Note

This function is different depends on the macro PXP_SHARE_ROTATE.

### 35.5.27 static void PXP_SetNextCommand ( PXP_Type * *base*, void * *commandAddr* ) [inline], [static]

The PXP supports a primitive ability to queue up one operation while the current operation is running. Workflow:

1. Prepare the PXP register values except STAT, CSCCOEFn, NEXT in the memory in the order they appear in the register map.
2. Call this function sets the new operation to PXP.
3. There are two methods to check whether the PXP has loaded the new operation. The first method is using PXP_IsNextCommandPending. If there is new operation not loaded by the PXP, this function returns true. The second method is checking the flag kPXP_CommandLoadFlag, if command loaded, this flag asserts. User could enable interrupt kPXP_CommandLoadInterruptEnable to get the loaded signal in interrupt way.
4. When command loaded by PXP, a new command could be set using this function.

```
uint32_t pxp_command1[48];
```

```
uint32_t pxp_command2[48];

// Prepare the register values.
pxp_command1[0] = ...;
pxp_command1[1] = ...;
// ...
pxp_command2[0] = ...;
pxp_command2[1] = ...;
// ...

// Make sure no new command pending.
while (PXP_IsNextCommandPending(PXP))
{
}

// Set new operation.
PXP_SetNextCommand(PXP, pxp_command1);

// Wait for new command loaded. Here could check @ref kPXP_CommandLoadFlag too.
while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command2);
```

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *commandAddr* | Address of the new command. |

### 35.5.28 static bool PXP_IsNextCommandPending ( PXP_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | UART peripheral base address. |

Returns

True is pending, false is not.

### 35.5.29 static void PXP_CancelNextCommand ( PXP_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | UART peripheral base address. |

### 35.5.30 void PXP_SetCsc1Mode ( PXP_Type * *base*, pxp_csc1_mode_t *mode* )

The CSC1 module receives scaled YUV/YCbCr444 pixels from the scale engine and converts the pixels to the RGB888 color space. It could only be used by process surface.

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *mode* | The conversion mode. |

### 35.5.31 static void PXP_EnableCsc1 ( PXP_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | PXP peripheral base address. |
| *enable* | True to enable, false to disable. |

# Chapter 36
# RTWDOG: 32-bit Watchdog Timer

## 36.1 Overview

The MCUXpresso SDK provides a peripheral driver for the RTWDOG module of MCUXpresso SDK devices.

## 36.2 Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtwdog

## Data Structures

- struct rtwdog_work_mode_t
  *Defines RTWDOG work mode.* *More...*
- struct rtwdog_config_t
  *Describes RTWDOG configuration structure.* *More...*

## Enumerations

- enum rtwdog_clock_source_t {
  kRTWDOG_ClockSource0 = 0U,
  kRTWDOG_ClockSource1 = 1U,
  kRTWDOG_ClockSource2 = 2U,
  kRTWDOG_ClockSource3 = 3U }
  *Describes RTWDOG clock source.*
- enum rtwdog_clock_prescaler_t {
  kRTWDOG_ClockPrescalerDivide1 = 0x0U,
  kRTWDOG_ClockPrescalerDivide256 = 0x1U }
  *Describes the selection of the clock prescaler.*
- enum rtwdog_test_mode_t {
  kRTWDOG_TestModeDisabled = 0U,
  kRTWDOG_UserModeEnabled = 1U,
  kRTWDOG_LowByteTest = 2U,
  kRTWDOG_HighByteTest = 3U }
  *Describes RTWDOG test mode.*
- enum _rtwdog_interrupt_enable_t { kRTWDOG_InterruptEnable = RTWDOG_CS_INT_MASK }
  *RTWDOG interrupt configuration structure.*
- enum _rtwdog_status_flags_t {
  kRTWDOG_RunningFlag = RTWDOG_CS_EN_MASK,
  kRTWDOG_InterruptFlag = RTWDOG_CS_FLG_MASK }
  *RTWDOG status flags.*

**Typical use case**

## Unlock sequence

- #define WDOG_FIRST_WORD_OF_UNLOCK (RTWDOG_UPDATE_KEY & 0xFFFFU)
  *First word of unlock sequence.*
- #define WDOG_SECOND_WORD_OF_UNLOCK ((RTWDOG_UPDATE_KEY >> 16U) & 0xFFFFU)
  *Second word of unlock sequence.*

## Refresh sequence

- #define WDOG_FIRST_WORD_OF_REFRESH (RTWDOG_REFRESH_KEY & 0xFFFFU)
  *First word of refresh sequence.*
- #define WDOG_SECOND_WORD_OF_REFRESH ((RTWDOG_REFRESH_KEY >> 16U) & 0xFFFFU)
  *Second word of refresh sequence.*

## Driver version

- #define FSL_RTWDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  *RTWDOG driver version 2.1.0.*

## RTWDOG Initialization and De-initialization

- void RTWDOG_GetDefaultConfig (rtwdog_config_t *config)
  *Initializes the RTWDOG configuration structure.*
- AT_QUICKACCESS_SECTION_CODE (void RTWDOG_Init(RTWDOG_Type *base, const rtwdog_config_t *config))
  *Initializes the RTWDOG module.*
- void RTWDOG_Deinit (RTWDOG_Type *base)
  *De-initializes the RTWDOG module.*

## RTWDOG functional Operation

- static void RTWDOG_Enable (RTWDOG_Type *base)
  *Enables the RTWDOG module.*
- static void RTWDOG_Disable (RTWDOG_Type *base)
  *Disables the RTWDOG module.*
- static void RTWDOG_EnableInterrupts (RTWDOG_Type *base, uint32_t mask)
  *Enables the RTWDOG interrupt.*
- static void RTWDOG_DisableInterrupts (RTWDOG_Type *base, uint32_t mask)
  *Disables the RTWDOG interrupt.*
- static uint32_t RTWDOG_GetStatusFlags (RTWDOG_Type *base)
  *Gets the RTWDOG all status flags.*
- static void RTWDOG_EnableWindowMode (RTWDOG_Type *base, bool enable)
  *Enables/disables the window mode.*
- static uint32_t RTWDOG_CountToMesec (RTWDOG_Type *base, uint32_t count, uint32_t clockFreqInHz)
  *Converts raw count value to millisecond.*
- void RTWDOG_ClearStatusFlags (RTWDOG_Type *base, uint32_t mask)
  *Clears the RTWDOG flag.*

- static void RTWDOG_SetTimeoutValue (RTWDOG_Type *base, uint16_t timeoutCount)
  *Sets the RTWDOG timeout value.*
- static void RTWDOG_SetWindowValue (RTWDOG_Type *base, uint16_t windowValue)
  *Sets the RTWDOG window value.*
- static void RTWDOG_Unlock (RTWDOG_Type *base)
  *Unlocks the RTWDOG register written.*
- static void RTWDOG_Refresh (RTWDOG_Type *base)
  *Refreshes the RTWDOG timer.*
- static uint16_t RTWDOG_GetCounterValue (RTWDOG_Type *base)
  *Gets the RTWDOG counter value.*

## 36.3 Data Structure Documentation

### 36.3.1 struct rtwdog_work_mode_t

#### Data Fields

- bool enableWait
  *Enables or disables RTWDOG in wait mode.*
- bool enableStop
  *Enables or disables RTWDOG in stop mode.*
- bool enableDebug
  *Enables or disables RTWDOG in debug mode.*

### 36.3.2 struct rtwdog_config_t

#### Data Fields

- bool enableRtwdog
  *Enables or disables RTWDOG.*
- rtwdog_clock_source_t clockSource
  *Clock source select.*
- rtwdog_clock_prescaler_t prescaler
  *Clock prescaler value.*
- rtwdog_work_mode_t workMode
  *Configures RTWDOG work mode in debug stop and wait mode.*
- rtwdog_test_mode_t testMode
  *Configures RTWDOG test mode.*
- bool enableUpdate
  *Update write-once register enable.*
- bool enableInterrupt
  *Enables or disables RTWDOG interrupt.*
- bool enableWindowMode
  *Enables or disables RTWDOG window mode.*
- uint16_t windowValue
  *Window value.*
- uint16_t timeoutValue
  *Timeout value.*

## 36.4 Macro Definition Documentation

### 36.4.1 #define FSL_RTWDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

## 36.5 Enumeration Type Documentation

### 36.5.1 enum rtwdog_clock_source_t

Enumerator

***kRTWDOG_ClockSource0*** Clock source 0.
***kRTWDOG_ClockSource1*** Clock source 1.
***kRTWDOG_ClockSource2*** Clock source 2.
***kRTWDOG_ClockSource3*** Clock source 3.

### 36.5.2 enum rtwdog_clock_prescaler_t

Enumerator

***kRTWDOG_ClockPrescalerDivide1*** Divided by 1.
***kRTWDOG_ClockPrescalerDivide256*** Divided by 256.

### 36.5.3 enum rtwdog_test_mode_t

Enumerator

***kRTWDOG_TestModeDisabled*** Test Mode disabled.
***kRTWDOG_UserModeEnabled*** User Mode enabled.
***kRTWDOG_LowByteTest*** Test Mode enabled, only low byte is used.
***kRTWDOG_HighByteTest*** Test Mode enabled, only high byte is used.

### 36.5.4 enum _rtwdog_interrupt_enable_t

This structure contains the settings for all of the RTWDOG interrupt configurations.

Enumerator

***kRTWDOG_InterruptEnable*** Interrupt is generated before forcing a reset.

### 36.5.5 enum _rtwdog_status_flags_t

This structure contains the RTWDOG status flags for use in the RTWDOG functions.

Enumerator

***kRTWDOG_RunningFlag*** Running flag, set when RTWDOG is enabled.
***kRTWDOG_InterruptFlag*** Interrupt flag, set when interrupt occurs.

## 36.6 Function Documentation

### 36.6.1 void RTWDOG_GetDefaultConfig ( rtwdog_config_t * *config* )

This function initializes the RTWDOG configuration structure to default values. The default values are:

```
*   rtwdogConfig->enableRtwdog = true;
*   rtwdogConfig->clockSource = kRTWDOG_ClockSource1;
*   rtwdogConfig->prescaler = kRTWDOG_ClockPrescalerDivide1;
*   rtwdogConfig->workMode.enableWait = true;
*   rtwdogConfig->workMode.enableStop = false;
*   rtwdogConfig->workMode.enableDebug = false;
*   rtwdogConfig->testMode = kRTWDOG_TestModeDisabled;
*   rtwdogConfig->enableUpdate = true;
*   rtwdogConfig->enableInterrupt = false;
*   rtwdogConfig->enableWindowMode = false;
*   rtwdogConfig->windowValue = 0U;
*   rtwdogConfig->timeoutValue = 0xFFFFU;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to the RTWDOG configuration structure. |

See Also

rtwdog_config_t

### 36.6.2 AT_QUICKACCESS_SECTION_CODE ( void *RTWDOG_InitRTWDOG_Type* **base, const rtwdog_config_t* **config* )

This function initializes the RTWDOG. To reconfigure the RTWDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
*   rtwdog_config_t config;
*   RTWDOG_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   config.enableUpdate = true;
*   RTWDOG_Init(wdog_base,&config);
*
```

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address. |
| *config* | The configuration of the RTWDOG. |

### 36.6.3 void RTWDOG_Deinit ( RTWDOG_Type * *base* )

This function shuts down the RTWDOG. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address. |

### 36.6.4 static void RTWDOG_Enable ( RTWDOG_Type * *base* ) [inline], [static]

This function writes a value into the WDOG_CS register to enable the RTWDOG. The WDOG_CS register is a write-once register. 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* | RTWDOG peripheral base address. |

### 36.6.5 static void RTWDOG_Disable ( RTWDOG_Type * *base* ) [inline], [static]

This function writes a value into the WDOG_CS register to disable the RTWDOG. The WDOG_CS register is a write-once register. 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* | RTWDOG peripheral base address |

### 36.6.6 static void RTWDOG_EnableInterrupts ( RTWDOG_Type * *base*, uint32_t *mask* ) [inline], [static]

This function writes a value into the WDOG_CS register to enable the RTWDOG interrupt. The WDOG-_CS register is a write-once register. 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* | RTWDOG peripheral base address. |
| *mask* | The interrupts to enable. The parameter can be a combination of the following source if defined:<br>• kRTWDOG_InterruptEnable |

### 36.6.7 static void RTWDOG_DisableInterrupts ( RTWDOG_Type * *base*, uint32_t *mask* ) [inline], [static]

This function writes a value into the WDOG_CS register to disable the RTWDOG interrupt. The WDO-G_CS register is a write-once register. 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* | RTWDOG peripheral base address. |
| *mask* | The interrupts to disabled. The parameter can be a combination of the following source if defined:<br>• kRTWDOG_InterruptEnable |

### 36.6.8 static uint32_t RTWDOG_GetStatusFlags ( RTWDOG_Type * *base* ) [inline], [static]

This function gets all status flags.

Example to get the running flag:

```
*   uint32_t status;
*   status = RTWDOG_GetStatusFlags(wdog_base) &
      kRTWDOG_RunningFlag;
*
```

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address |

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

_rtwdog_status_flags_t
- true: related status flag has been set.
- false: related status flag is not set.

### 36.6.9 static void RTWDOG_EnableWindowMode ( RTWDOG_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address. |
| *enable* | Enables(true) or disables(false) the feature. |

### 36.6.10 static uint32_t RTWDOG_CountToMesec ( RTWDOG_Type * *base,* uint32_t *count,* uint32_t *clockFreqInHz* ) [inline], [static]

Note that if the clock frequency is too high the timeout period can be less than 1 ms. In this case this api will return 0 value.

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address. |
| *count* | Raw count value.<br>**clockFreqInHz The frequency of the clock source RTWDOG uses.** |

### 36.6.11 void RTWDOG_ClearStatusFlags ( RTWDOG_Type * *base,* uint32_t *mask* )

This function clears the RTWDOG status flag.

Example to clear an interrupt flag:

```
*   RTWDOG_ClearStatusFlags(wdog_base,
      kRTWDOG_InterruptFlag);
*
```

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address. |
| *mask* | The status flags to clear. The parameter can be any combination of the following values:<br>• kRTWDOG_InterruptFlag |

### 36.6.12 static void RTWDOG_SetTimeoutValue ( RTWDOG_Type * *base*, uint16_t *timeoutCount* ) [inline], [static]

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. 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* | RTWDOG peripheral base address |
| *timeoutCount* | RTWDOG timeout value, count of RTWDOG clock ticks. |

### 36.6.13 static void RTWDOG_SetWindowValue ( RTWDOG_Type * *base*, uint16_t *windowValue* ) [inline], [static]

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. 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* | RTWDOG peripheral base address. |

| | |
|---|---|
| *windowValue* | RTWDOG window value. |

### 36.6.14 static void RTWDOG_Unlock ( RTWDOG_Type * *base* ) [inline], [static]

This function unlocks the RTWDOG 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* | RTWDOG peripheral base address |

### 36.6.15 static void RTWDOG_Refresh ( RTWDOG_Type * *base* ) [inline], [static]

This function feeds the RTWDOG. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address |

### 36.6.16 static uint16_t RTWDOG_GetCounterValue ( RTWDOG_Type * *base* ) [inline], [static]

This function gets the RTWDOG counter value.

Parameters

| | |
|---|---|
| *base* | RTWDOG peripheral base address. |

Returns

Current RTWDOG counter value.

# Chapter 37
# QTMR: Quad Timer Driver

## 37.1 Overview

The MCUXpresso SDK provides a driver for the QTMR module of MCUXpresso SDK devices.

## Data Structures

- struct qtmr_config_t
  *Quad Timer config structure. More...*

## Enumerations

- enum qtmr_primary_count_source_t {
  kQTMR_ClockCounter0InputPin = 0,
  kQTMR_ClockCounter1InputPin,
  kQTMR_ClockCounter2InputPin,
  kQTMR_ClockCounter3InputPin,
  kQTMR_ClockCounter0Output,
  kQTMR_ClockCounter1Output,
  kQTMR_ClockCounter2Output,
  kQTMR_ClockCounter3Output,
  kQTMR_ClockDivide_1,
  kQTMR_ClockDivide_2,
  kQTMR_ClockDivide_4,
  kQTMR_ClockDivide_8,
  kQTMR_ClockDivide_16,
  kQTMR_ClockDivide_32,
  kQTMR_ClockDivide_64,
  kQTMR_ClockDivide_128 }
  *Quad Timer primary clock source selection.*
- enum qtmr_input_source_t {
  kQTMR_Counter0InputPin = 0,
  kQTMR_Counter1InputPin,
  kQTMR_Counter2InputPin,
  kQTMR_Counter3InputPin }
  *Quad Timer input sources selection.*
- enum qtmr_counting_mode_t {

kQTMR_NoOperation = 0,
kQTMR_PriSrcRiseEdge,
kQTMR_PriSrcRiseAndFallEdge,
kQTMR_PriSrcRiseEdgeSecInpHigh,
kQTMR_QuadCountMode,
kQTMR_PriSrcRiseEdgeSecDir,
kQTMR_SecSrcTrigPriCnt,
kQTMR_CascadeCount }
*Quad Timer counting mode selection.*

- enum qtmr_output_mode_t {
kQTMR_AssertWhenCountActive = 0,
kQTMR_ClearOnCompare,
kQTMR_SetOnCompare,
kQTMR_ToggleOnCompare,
kQTMR_ToggleOnAltCompareReg,
kQTMR_SetOnCompareClearOnSecSrcInp,
kQTMR_SetOnCompareClearOnCountRoll,
kQTMR_EnableGateClock }
*Quad Timer output mode selection.*
- enum qtmr_input_capture_edge_t {
kQTMR_NoCapture = 0,
kQTMR_RisingEdge,
kQTMR_FallingEdge,
kQTMR_RisingAndFallingEdge }
*Quad Timer input capture edge mode, rising edge, or falling edge.*
- enum qtmr_preload_control_t {
kQTMR_NoPreload = 0,
kQTMR_LoadOnComp1,
kQTMR_LoadOnComp2 }
*Quad Timer input capture edge mode, rising edge, or falling edge.*
- enum qtmr_debug_action_t {
kQTMR_RunNormalInDebug = 0U,
kQTMR_HaltCounter,
kQTMR_ForceOutToZero,
kQTMR_HaltCountForceOutZero }
*List of Quad Timer run options when in Debug mode.*
- enum qtmr_interrupt_enable_t {
kQTMR_CompareInterruptEnable = (1U << 0),
kQTMR_Compare1InterruptEnable = (1U << 1),
kQTMR_Compare2InterruptEnable = (1U << 2),
kQTMR_OverflowInterruptEnable = (1U << 3),
kQTMR_EdgeInterruptEnable = (1U << 4) }
*List of Quad Timer interrupts.*
- enum qtmr_status_flags_t {

kQTMR_CompareFlag = (1U << 0),
kQTMR_Compare1Flag = (1U << 1),
kQTMR_Compare2Flag = (1U << 2),
kQTMR_OverflowFlag = (1U << 3),
kQTMR_EdgeFlag = (1U << 4) }
*List of Quad Timer flags.*

## Functions

- status_t QTMR_SetupPwm (TMR_Type *base, uint32_t pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity, uint32_t srcClock_Hz)
  *Sets up Quad timer module for PWM signal output.*
- void QTMR_SetupInputCapture (TMR_Type *base, qtmr_input_source_t capturePin, bool inputPolarity, bool reloadOnCapture, qtmr_input_capture_edge_t captureMode)
  *Allows the user to count the source clock cycles until a capture event arrives.*

## Driver version

- #define FSL_QTMR_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *Version 2.0.0.*

## Initialization and deinitialization

- void QTMR_Init (TMR_Type *base, const qtmr_config_t *config)
  *Ungates the Quad Timer clock and configures the peripheral for basic operation.*
- void QTMR_Deinit (TMR_Type *base)
  *Stops the counter and gates the Quad Timer clock.*
- void QTMR_GetDefaultConfig (qtmr_config_t *config)
  *Fill in the Quad Timer config struct with the default settings.*

## Interrupt Interface

- void QTMR_EnableInterrupts (TMR_Type *base, uint32_t mask)
  *Enables the selected Quad Timer interrupts.*
- void QTMR_DisableInterrupts (TMR_Type *base, uint32_t mask)
  *Disables the selected Quad Timer interrupts.*
- uint32_t QTMR_GetEnabledInterrupts (TMR_Type *base)
  *Gets the enabled Quad Timer interrupts.*

## Status Interface

- uint32_t QTMR_GetStatus (TMR_Type *base)
  *Gets the Quad Timer status flags.*
- void QTMR_ClearStatusFlags (TMR_Type *base, uint32_t mask)
  *Clears the Quad Timer status flags.*

## Read and Write the timer period

- void QTMR_SetTimerPeriod (TMR_Type *base, uint16_t ticks)

*Sets the timer period in ticks.*

- static uint16_t QTMR_GetCurrentTimerCount (TMR_Type *base)
  *Reads the current timer counting value.*

### Timer Start and Stop

- static void QTMR_StartTimer (TMR_Type *base, qtmr_counting_mode_t clockSource)
  *Starts the Quad Timer counter.*
- static void QTMR_StopTimer (TMR_Type *base)
  *Stops the Quad Timer counter.*

## 37.2 Data Structure Documentation

### 37.2.1 struct qtmr_config_t

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the QTMR_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

### Data Fields

- qtmr_primary_count_source_t primarySource
  *Specify the primary count source.*
- qtmr_input_source_t secondarySource
  *Specify the secondary count source.*
- bool enableMasterMode
  *true: Broadcast compare function output to other counters; false no broadcast*
- bool enableExternalForce
  *true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter*
- uint8_t faultFilterCount
  *Fault filter count.*
- uint8_t faultFilterPeriod
  *Fault filter period;value of 0 will bypass the filter.*
- qtmr_debug_action_t debugMode
  *Operation in Debug mode.*

## 37.3 Enumeration Type Documentation

### 37.3.1 enum qtmr_primary_count_source_t

Enumerator

| | |
|---|---|
| ***kQTMR_ClockCounter0InputPin*** | Use counter 0 input pin. |
| ***kQTMR_ClockCounter1InputPin*** | Use counter 1 input pin. |
| ***kQTMR_ClockCounter2InputPin*** | Use counter 2 input pin. |

***kQTMR_ClockCounter3InputPin*** Use counter 3 input pin.
***kQTMR_ClockCounter0Output*** Use counter 0 output.
***kQTMR_ClockCounter1Output*** Use counter 1 output.
***kQTMR_ClockCounter2Output*** Use counter 2 output.
***kQTMR_ClockCounter3Output*** Use counter 3 output.
***kQTMR_ClockDivide_1*** IP bus clock divide by 1 prescaler.
***kQTMR_ClockDivide_2*** IP bus clock divide by 2 prescaler.
***kQTMR_ClockDivide_4*** IP bus clock divide by 4 prescaler.
***kQTMR_ClockDivide_8*** IP bus clock divide by 8 prescaler.
***kQTMR_ClockDivide_16*** IP bus clock divide by 16 prescaler.
***kQTMR_ClockDivide_32*** IP bus clock divide by 32 prescaler.
***kQTMR_ClockDivide_64*** IP bus clock divide by 64 prescaler.
***kQTMR_ClockDivide_128*** IP bus clock divide by 128 prescaler.

### 37.3.2 enum qtmr_input_source_t

Enumerator

***kQTMR_Counter0InputPin*** Use counter 0 input pin.
***kQTMR_Counter1InputPin*** Use counter 1 input pin.
***kQTMR_Counter2InputPin*** Use counter 2 input pin.
***kQTMR_Counter3InputPin*** Use counter 3 input pin.

### 37.3.3 enum qtmr_counting_mode_t

Enumerator

***kQTMR_NoOperation*** No operation.
***kQTMR_PriSrcRiseEdge*** Count rising edges or primary source.
***kQTMR_PriSrcRiseAndFallEdge*** Count rising and falling edges of primary source.
***kQTMR_PriSrcRiseEdgeSecInpHigh*** Count rise edges of pri SRC while sec inp high active.
***kQTMR_QuadCountMode*** Quadrature count mode, uses pri and sec sources.
***kQTMR_PriSrcRiseEdgeSecDir*** Count rising edges of pri SRC; sec SRC specifies dir.
***kQTMR_SecSrcTrigPriCnt*** Edge of sec SRC trigger primary count until compare.
***kQTMR_CascadeCount*** Cascaded count mode (up/down)

### 37.3.4 enum qtmr_output_mode_t

Enumerator

***kQTMR_AssertWhenCountActive*** Assert OFLAG while counter is active.

***kQTMR_ClearOnCompare*** Clear OFLAG on successful compare.
***kQTMR_SetOnCompare*** Set OFLAG on successful compare.
***kQTMR_ToggleOnCompare*** Toggle OFLAG on successful compare.
***kQTMR_ToggleOnAltCompareReg*** Toggle OFLAG using alternating compare registers.
***kQTMR_SetOnCompareClearOnSecSrcInp*** Set OFLAG on compare, clear on sec SRC input edge.

***kQTMR_SetOnCompareClearOnCountRoll*** Set OFLAG on compare, clear on counter rollover.
***kQTMR_EnableGateClock*** Enable gated clock output while count is active.

### 37.3.5 enum qtmr_input_capture_edge_t

Enumerator

***kQTMR_NoCapture*** Capture is disabled.
***kQTMR_RisingEdge*** Capture on rising edge (IPS=0) or falling edge (IPS=1)
***kQTMR_FallingEdge*** Capture on falling edge (IPS=0) or rising edge (IPS=1)
***kQTMR_RisingAndFallingEdge*** Capture on both edges.

### 37.3.6 enum qtmr_preload_control_t

Enumerator

***kQTMR_NoPreload*** Never preload.
***kQTMR_LoadOnComp1*** Load upon successful compare with value in COMP1.
***kQTMR_LoadOnComp2*** Load upon successful compare with value in COMP2.

### 37.3.7 enum qtmr_debug_action_t

Enumerator

***kQTMR_RunNormalInDebug*** Continue with normal operation.
***kQTMR_HaltCounter*** Halt counter.
***kQTMR_ForceOutToZero*** Force output to logic 0.
***kQTMR_HaltCountForceOutZero*** Halt counter and force output to logic 0.

### 37.3.8 enum qtmr_interrupt_enable_t

Enumerator

***kQTMR_CompareInterruptEnable*** Compare interrupt.

***kQTMR_Compare1InterruptEnable*** Compare 1 interrupt.
***kQTMR_Compare2InterruptEnable*** Compare 2 interrupt.
***kQTMR_OverflowInterruptEnable*** Timer overflow interrupt.
***kQTMR_EdgeInterruptEnable*** Input edge interrupt.

### 37.3.9 enum qtmr_status_flags_t

Enumerator

***kQTMR_CompareFlag*** Compare flag.
***kQTMR_Compare1Flag*** Compare 1 flag.
***kQTMR_Compare2Flag*** Compare 2 flag.
***kQTMR_OverflowFlag*** Timer overflow flag.
***kQTMR_EdgeFlag*** Input edge flag.

## 37.4 Function Documentation

### 37.4.1 void QTMR_Init ( TMR_Type * *base*, const qtmr_config_t * *config* )

Note

This API should be called at the beginning of the application using the Quad Timer driver.

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *config* | Pointer to user's Quad Timer config structure |

### 37.4.2 void QTMR_Deinit ( TMR_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |

### 37.4.3 void QTMR_GetDefaultConfig ( qtmr_config_t * *config* )

The default values are:

```
*     config->debugMode = kQTMR_RunNormalInDebug;
*     config->enableExternalForce = false;
*     config->enableMasterMode = false;
```

```
*     config->faultFilterCount = 0;
*     config->faultFilterPeriod = 0;
*     config->primarySource = kQTMR_ClockDivide_2;
*     config->secondarySource = kQTMR_Counter0InputPin;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to user's Quad Timer config structure. |

### 37.4.4 status_t QTMR_SetupPwm ( TMR_Type * *base*, uint32_t *pwmFreqHz*, uint8_t *dutyCyclePercent*, bool *outputPolarity*, uint32_t *srcClock_Hz* )

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *pwmFreqHz* | PWM signal frequency in Hz |
| *dutyCycle-Percent* | PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |
| *outputPolarity* | true: invert polarity of the output signal, false: no inversion |
| *srcClock_Hz* | Main counter clock in Hz. |

Returns

Returns an error if there was error setting up the signal.

### 37.4.5 void QTMR_SetupInputCapture ( TMR_Type * *base*, qtmr_input-_source_t *capturePin*, bool *inputPolarity*, bool *reloadOnCapture*, qtmr_input_capture_edge_t *captureMode* )

The count is stored in the capture register.

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *capturePin* | Pin through which we receive the input signal to trigger the capture |
| *inputPolarity* | true: invert polarity of the input signal, false: no inversion |
| *reloadOn-Capture* | true: reload the counter when an input capture occurs, false: no reload |
| *captureMode* | Specifies which edge of the input signal triggers a capture |

### 37.4.6 void QTMR_EnableInterrupts ( TMR_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration qtmr_-interrupt_enable_t |

### 37.4.7 void QTMR_DisableInterrupts ( TMR_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration qtmr_-interrupt_enable_t |

### 37.4.8 uint32_t QTMR_GetEnabledInterrupts ( TMR_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |

Returns

The enabled interrupts. This is the logical OR of members of the enumeration qtmr_interrupt_-enable_t

### 37.4.9 uint32_t QTMR_GetStatus ( TMR_Type * *base* )

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |

Returns

The status flags. This is the logical OR of members of the enumeration qtmr_status_flags_t

### 37.4.10 void QTMR_ClearStatusFlags ( TMR_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *mask* | The status flags to clear. This is a logical OR of members of the enumeration qtmr_status_flags_t |

### 37.4.11 void QTMR_SetTimerPeriod ( TMR_Type * *base*, uint16_t *ticks* )

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in fsl_common.h to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *ticks* | Timer period in units of ticks |

### 37.4.12 static uint16_t QTMR_GetCurrentTimerCount ( TMR_Type * *base* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in fsl_common.h to convert ticks to usec or msec

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |

Returns

Current counter value in ticks

### 37.4.13 static void QTMR_StartTimer ( TMR_Type * *base*, qtmr_counting_mode_t *clockSource* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |
| *clockSource* | Quad Timer clock source |

### 37.4.14 static void QTMR_StopTimer ( TMR_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | Quad Timer peripheral base address |

# Chapter 38
# SAI: Serial Audio Interface

## 38.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Audio Interface (SAI) module of MCUXpresso SDK devices.

SAI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SAI initialization, configuration and operation, and for optimization and customization purposes. Using the functional API requires the knowledge of the SAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SAI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the sai_handle_t as the first parameter. Initialize the handle by calling the SAI_TransferTxCreateHandle() or SAI_TransferRxCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SAI_TransferSendNonBlocking() and SAI_TransferReceiveNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_SAI_TxIdle and kStatus_SAI_RxIdle status.

## 38.2 Typical use case

### 38.2.1 SAI Send/receive using an interrupt method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/sai

### 38.2.2 SAI Send/receive using a DMA method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/sai

## Modules

- SAI DMA Driver
- SAI Driver
- SAI EDMA Driver
- SAI SDMA Driver

## 38.3 SAI Driver

### 38.3.1 Overview

#### Data Structures

- struct sai_config_t
  *SAI user configuration structure.* More...
- struct sai_transfer_format_t
  *sai transfer format* More...
- struct sai_fifo_t
  *sai fifo configurations* More...
- struct sai_bit_clock_t
  *sai bit clock configurations* More...
- struct sai_frame_sync_t
  *sai frame sync configurations* More...
- struct sai_serial_data_t
  *sai serial data configurations* More...
- struct sai_transceiver_t
  *sai transceiver configurations* More...
- struct sai_transfer_t
  *SAI transfer structure.* More...
- struct sai_handle_t
  *SAI handle structure.* More...

#### Macros

- #define SAI_XFER_QUEUE_SIZE (4)
  *SAI transfer queue size, user can refine it according to use case.*

#### Typedefs

- typedef void(∗ sai_transfer_callback_t )(I2S_Type ∗base, sai_handle_t ∗handle, status_t status, void ∗userData)
  *SAI transfer callback prototype.*

#### Enumerations

- enum _sai_status_t {
  kStatus_SAI_TxBusy = MAKE_STATUS(kStatusGroup_SAI, 0),
  kStatus_SAI_RxBusy = MAKE_STATUS(kStatusGroup_SAI, 1),
  kStatus_SAI_TxError = MAKE_STATUS(kStatusGroup_SAI, 2),
  kStatus_SAI_RxError = MAKE_STATUS(kStatusGroup_SAI, 3),
  kStatus_SAI_QueueFull = MAKE_STATUS(kStatusGroup_SAI, 4),
  kStatus_SAI_TxIdle = MAKE_STATUS(kStatusGroup_SAI, 5),
  kStatus_SAI_RxIdle = MAKE_STATUS(kStatusGroup_SAI, 6) }

*SAI return status.*

- enum _sai_channel_mask {
  kSAI_Channel0Mask = 1 << 0U,
  kSAI_Channel1Mask = 1 << 1U,
  kSAI_Channel2Mask = 1 << 2U,
  kSAI_Channel3Mask = 1 << 3U,
  kSAI_Channel4Mask = 1 << 4U,
  kSAI_Channel5Mask = 1 << 5U,
  kSAI_Channel6Mask = 1 << 6U,
  kSAI_Channel7Mask = 1 << 7U }
- enum sai_protocol_t {
  kSAI_BusLeftJustified = 0x0U,
  kSAI_BusRightJustified,
  kSAI_BusI2S,
  kSAI_BusPCMA,
  kSAI_BusPCMB }

  *Define the SAI bus type.*
- enum sai_master_slave_t {
  kSAI_Master = 0x0U,
  kSAI_Slave = 0x1U,
  kSAI_Bclk_Master_FrameSync_Slave = 0x2U,
  kSAI_Bclk_Slave_FrameSync_Master = 0x3U }

  *Master or slave mode.*
- enum sai_mono_stereo_t {
  kSAI_Stereo = 0x0U,
  kSAI_MonoRight,
  kSAI_MonoLeft }

  *Mono or stereo audio format.*
- enum sai_data_order_t {
  kSAI_DataLSB = 0x0U,
  kSAI_DataMSB }

  *SAI data order, MSB or LSB.*
- enum sai_clock_polarity_t {
  kSAI_PolarityActiveHigh = 0x0U,
  kSAI_PolarityActiveLow,
  kSAI_SampleOnFallingEdge = 0x0U,
  kSAI_SampleOnRisingEdge }

  *SAI clock polarity, active high or low.*
- enum sai_sync_mode_t {
  kSAI_ModeAsync = 0x0U,
  kSAI_ModeSync }

  *Synchronous or asynchronous mode.*
- enum sai_mclk_source_t {
  kSAI_MclkSourceSysclk = 0x0U,
  kSAI_MclkSourceSelect1,
  kSAI_MclkSourceSelect2,
  kSAI_MclkSourceSelect3 }

*Mater clock source.*

- enum sai_bclk_source_t {
  kSAI_BclkSourceBusclk = 0x0U,
  kSAI_BclkSourceMclkOption1 = 0x1U,
  kSAI_BclkSourceMclkOption2 = 0x2U,
  kSAI_BclkSourceMclkOption3 = 0x3U,
  kSAI_BclkSourceMclkDiv = 0x1U,
  kSAI_BclkSourceOtherSai0 = 0x2U,
  kSAI_BclkSourceOtherSai1 = 0x3U }
  *Bit clock source.*
- enum _sai_interrupt_enable_t {
  kSAI_WordStartInterruptEnable,
  kSAI_SyncErrorInterruptEnable = I2S_TCSR_SEIE_MASK,
  kSAI_FIFOWarningInterruptEnable = I2S_TCSR_FWIE_MASK,
  kSAI_FIFOErrorInterruptEnable = I2S_TCSR_FEIE_MASK,
  kSAI_FIFORequestInterruptEnable = I2S_TCSR_FRIE_MASK }
  *The SAI interrupt enable flag.*
- enum _sai_dma_enable_t {
  kSAI_FIFOWarningDMAEnable = I2S_TCSR_FWDE_MASK,
  kSAI_FIFORequestDMAEnable = I2S_TCSR_FRDE_MASK }
  *The DMA request sources.*
- enum _sai_flags {
  kSAI_WordStartFlag = I2S_TCSR_WSF_MASK,
  kSAI_SyncErrorFlag = I2S_TCSR_SEF_MASK,
  kSAI_FIFOErrorFlag = I2S_TCSR_FEF_MASK,
  kSAI_FIFORequestFlag = I2S_TCSR_FRF_MASK,
  kSAI_FIFOWarningFlag = I2S_TCSR_FWF_MASK }
  *The SAI status flag.*
- enum sai_reset_type_t {
  kSAI_ResetTypeSoftware = I2S_TCSR_SR_MASK,
  kSAI_ResetTypeFIFO = I2S_TCSR_FR_MASK,
  kSAI_ResetAll = I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK }
  *The reset type.*
- enum sai_fifo_packing_t {
  kSAI_FifoPackingDisabled = 0x0U,
  kSAI_FifoPacking8bit = 0x2U,
  kSAI_FifoPacking16bit = 0x3U }
  *The SAI packing mode The mode includes 8 bit and 16 bit packing.*
- enum sai_sample_rate_t {

kSAI_SampleRate8KHz = 8000U,
kSAI_SampleRate11025Hz = 11025U,
kSAI_SampleRate12KHz = 12000U,
kSAI_SampleRate16KHz = 16000U,
kSAI_SampleRate22050Hz = 22050U,
kSAI_SampleRate24KHz = 24000U,
kSAI_SampleRate32KHz = 32000U,
kSAI_SampleRate44100Hz = 44100U,
kSAI_SampleRate48KHz = 48000U,
kSAI_SampleRate96KHz = 96000U,
kSAI_SampleRate192KHz = 192000U,
kSAI_SampleRate384KHz = 384000U }
*Audio sample rate.*

- enum sai_word_width_t {
kSAI_WordWidth8bits = 8U,
kSAI_WordWidth16bits = 16U,
kSAI_WordWidth24bits = 24U,
kSAI_WordWidth32bits = 32U }
*Audio word width.*
- enum sai_transceiver_type_t {
kSAI_Transmitter = 0U,
kSAI_Receiver = 1U }
*sai transceiver type*
- enum sai_frame_sync_len_t {
kSAI_FrameSyncLenOneBitClk = 0U,
kSAI_FrameSyncLenPerWordWidth = 1U }
*sai frame sync len*

## Driver version

- #define FSL_SAI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
*Version 2.2.1.*

## Initialization and deinitialization

- void SAI_TxInit (I2S_Type *base, const sai_config_t *config)
*Initializes the SAI Tx peripheral.*
- void SAI_RxInit (I2S_Type *base, const sai_config_t *config)
*Initializes the SAI Rx peripheral.*
- void SAI_TxGetDefaultConfig (sai_config_t *config)
*Sets the SAI Tx configuration structure to default values.*
- void SAI_RxGetDefaultConfig (sai_config_t *config)
*Sets the SAI Rx configuration structure to default values.*
- void SAI_Init (I2S_Type *base)
*Initializes the SAI peripheral.*
- void SAI_Deinit (I2S_Type *base)

*De-initializes the SAI peripheral.*
- void SAI_TxReset (I2S_Type *base)
  *Resets the SAI Tx.*
- void SAI_RxReset (I2S_Type *base)
  *Resets the SAI Rx.*
- void SAI_TxEnable (I2S_Type *base, bool enable)
  *Enables/disables the SAI Tx.*
- void SAI_RxEnable (I2S_Type *base, bool enable)
  *Enables/disables the SAI Rx.*
- static void SAI_TxSetBitClockDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  *Set Rx bit clock direction.*
- static void SAI_RxSetBitClockDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  *Set Rx bit clock direction.*
- static void SAI_RxSetFrameSyncDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  *Set Rx frame sync direction.*
- static void SAI_TxSetFrameSyncDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  *Set Tx frame sync direction.*
- void SAI_TxSetBitClockRate (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
  *Transmitter bit clock rate configurations.*
- void SAI_RxSetBitClockRate (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
  *Receiver bit clock rate configurations.*
- void SAI_TxSetBitclockConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
  *Transmitter Bit clock configurations.*
- void SAI_RxSetBitclockConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
  *Receiver Bit clock configurations.*
- void SAI_TxSetFifoConfig (I2S_Type *base, sai_fifo_t *config)
  *SAI transmitter fifo configurations.*
- void SAI_RxSetFifoConfig (I2S_Type *base, sai_fifo_t *config)
  *SAI receiver fifo configurations.*
- void SAI_TxSetFrameSyncConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
  *SAI transmitter Frame sync configurations.*
- void SAI_RxSetFrameSyncConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
  *SAI receiver Frame sync configurations.*
- void SAI_TxSetSerialDataConfig (I2S_Type *base, sai_serial_data_t *config)
  *SAI transmitter Serial data configurations.*
- void SAI_RxSetSerialDataConfig (I2S_Type *base, sai_serial_data_t *config)
  *SAI receiver Serial data configurations.*
- void SAI_TxSetConfig (I2S_Type *base, sai_transceiver_t *config)
  *SAI transmitter configurations.*
- void SAI_RxSetConfig (I2S_Type *base, sai_transceiver_t *config)
  *SAI receiver configurations.*
- void SAI_GetClassicI2SConfig (sai_transceiver_t *config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  *Get classic I2S mode configurations.*

- void SAI_GetLeftJustifiedConfig (sai_transceiver_t *config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  *Get left justified mode configurations.*
- void SAI_GetRightJustifiedConfig (sai_transceiver_t *config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  *Get right justified mode configurations.*
- void SAI_GetTDMConfig (sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth, sai_word_width_t bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)
  *Get TDM mode configurations.*
- void SAI_GetDSPConfig (sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  *Get DSP mode configurations.*

## Status

- static uint32_t SAI_TxGetStatusFlag (I2S_Type *base)
  *Gets the SAI Tx status flag state.*
- static void SAI_TxClearStatusFlags (I2S_Type *base, uint32_t mask)
  *Clears the SAI Tx status flag state.*
- static uint32_t SAI_RxGetStatusFlag (I2S_Type *base)
  *Gets the SAI Tx status flag state.*
- static void SAI_RxClearStatusFlags (I2S_Type *base, uint32_t mask)
  *Clears the SAI Rx status flag state.*
- void SAI_TxSoftwareReset (I2S_Type *base, sai_reset_type_t type)
  *Do software reset or FIFO reset .*
- void SAI_RxSoftwareReset (I2S_Type *base, sai_reset_type_t type)
  *Do software reset or FIFO reset .*
- void SAI_TxSetChannelFIFOMask (I2S_Type *base, uint8_t mask)
  *Set the Tx channel FIFO enable mask.*
- void SAI_RxSetChannelFIFOMask (I2S_Type *base, uint8_t mask)
  *Set the Rx channel FIFO enable mask.*
- void SAI_TxSetDataOrder (I2S_Type *base, sai_data_order_t order)
  *Set the Tx data order.*
- void SAI_RxSetDataOrder (I2S_Type *base, sai_data_order_t order)
  *Set the Rx data order.*
- void SAI_TxSetBitClockPolarity (I2S_Type *base, sai_clock_polarity_t polarity)
  *Set the Tx data order.*
- void SAI_RxSetBitClockPolarity (I2S_Type *base, sai_clock_polarity_t polarity)
  *Set the Rx data order.*
- void SAI_TxSetFrameSyncPolarity (I2S_Type *base, sai_clock_polarity_t polarity)
  *Set the Tx data order.*
- void SAI_RxSetFrameSyncPolarity (I2S_Type *base, sai_clock_polarity_t polarity)
  *Set the Rx data order.*
- void SAI_TxSetFIFOPacking (I2S_Type *base, sai_fifo_packing_t pack)
  *Set Tx FIFO packing feature.*
- void SAI_RxSetFIFOPacking (I2S_Type *base, sai_fifo_packing_t pack)
  *Set Rx FIFO packing feature.*
- static void SAI_TxSetFIFOErrorContinue (I2S_Type *base, bool isEnabled)
  *Set Tx FIFO error continue.*

- static void SAI_RxSetFIFOErrorContinue (I2S_Type *base, bool isEnabled)
  *Set Rx FIFO error continue.*

## Interrupts

- static void SAI_TxEnableInterrupts (I2S_Type *base, uint32_t mask)
  *Enables the SAI Tx interrupt requests.*
- static void SAI_RxEnableInterrupts (I2S_Type *base, uint32_t mask)
  *Enables the SAI Rx interrupt requests.*
- static void SAI_TxDisableInterrupts (I2S_Type *base, uint32_t mask)
  *Disables the SAI Tx interrupt requests.*
- static void SAI_RxDisableInterrupts (I2S_Type *base, uint32_t mask)
  *Disables the SAI Rx interrupt requests.*

## DMA Control

- static void SAI_TxEnableDMA (I2S_Type *base, uint32_t mask, bool enable)
  *Enables/disables the SAI Tx DMA requests.*
- static void SAI_RxEnableDMA (I2S_Type *base, uint32_t mask, bool enable)
  *Enables/disables the SAI Rx DMA requests.*
- static uint32_t SAI_TxGetDataRegisterAddress (I2S_Type *base, uint32_t channel)
  *Gets the SAI Tx data register address.*
- static uint32_t SAI_RxGetDataRegisterAddress (I2S_Type *base, uint32_t channel)
  *Gets the SAI Rx data register address.*

## Bus Operations

- void SAI_TxSetFormat (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Tx audio format.*
- void SAI_RxSetFormat (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Rx audio format.*
- void SAI_WriteBlocking (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
  *Sends data using a blocking method.*
- void SAI_WriteMultiChannelBlocking (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
  *Sends data to multi channel using a blocking method.*
- static void SAI_WriteData (I2S_Type *base, uint32_t channel, uint32_t data)
  *Writes data into SAI FIFO.*
- void SAI_ReadBlocking (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
  *Receives data using a blocking method.*
- void SAI_ReadMultiChannelBlocking (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

*Receives multi channel data using a blocking method.*
- static uint32_t SAI_ReadData (I2S_Type *base, uint32_t channel)
  *Reads data from the SAI FIFO.*

## Transactional

- void SAI_TransferTxCreateHandle (I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t callback, void *userData)
  *Initializes the SAI Tx handle.*
- void SAI_TransferRxCreateHandle (I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t callback, void *userData)
  *Initializes the SAI Rx handle.*
- void SAI_TransferTxSetConfig (I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)
  *SAI transmitter transfer configurations.*
- void SAI_TransferRxSetConfig (I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)
  *SAI receiver transfer configurations.*
- status_t SAI_TransferTxSetFormat (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Tx audio format.*
- status_t SAI_TransferRxSetFormat (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Rx audio format.*
- status_t SAI_TransferSendNonBlocking (I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)
  *Performs an interrupt non-blocking send transfer on SAI.*
- status_t SAI_TransferReceiveNonBlocking (I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)
  *Performs an interrupt non-blocking receive transfer on SAI.*
- status_t SAI_TransferGetSendCount (I2S_Type *base, sai_handle_t *handle, size_t *count)
  *Gets a set byte count.*
- status_t SAI_TransferGetReceiveCount (I2S_Type *base, sai_handle_t *handle, size_t *count)
  *Gets a received byte count.*
- void SAI_TransferAbortSend (I2S_Type *base, sai_handle_t *handle)
  *Aborts the current send.*
- void SAI_TransferAbortReceive (I2S_Type *base, sai_handle_t *handle)
  *Aborts the current IRQ receive.*
- void SAI_TransferTerminateSend (I2S_Type *base, sai_handle_t *handle)
  *Terminate all SAI send.*
- void SAI_TransferTerminateReceive (I2S_Type *base, sai_handle_t *handle)
  *Terminate all SAI receive.*
- void SAI_TransferTxHandleIRQ (I2S_Type *base, sai_handle_t *handle)
  *Tx interrupt handler.*
- void SAI_TransferRxHandleIRQ (I2S_Type *base, sai_handle_t *handle)
  *Tx interrupt handler.*

## 38.3.2 Data Structure Documentation

### 38.3.2.1 struct sai_config_t

#### Data Fields

- sai_protocol_t protocol
  *Audio bus protocol in SAI.*
- sai_sync_mode_t syncMode
  *SAI sync mode, control Tx/Rx clock sync.*
- sai_bclk_source_t bclkSource
  *Bit Clock source.*
- sai_master_slave_t masterSlave
  *Master or slave.*

### 38.3.2.2 struct sai_transfer_format_t

#### Data Fields

- uint32_t sampleRate_Hz
  *Sample rate of audio data.*
- uint32_t bitWidth
  *Data length of audio data, usually 8/16/24/32 bits.*
- sai_mono_stereo_t stereo
  *Mono or stereo.*
- uint8_t watermark
  *Watermark value.*
- uint8_t channel
  *Transfer start channel.*
- uint8_t channelMask
  *enabled channel mask value, reference _sai_channel_mask*
- uint8_t endChannel
  *end channel number*
- uint8_t channelNums
  *Total enabled channel numbers.*
- sai_protocol_t protocol
  *Which audio protocol used.*
- bool isFrameSyncCompact
  *True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.*

#### 38.3.2.2.0.38 Field Documentation

##### 38.3.2.2.0.38.1 bool sai_transfer_format_t::isFrameSyncCompact

### 38.3.2.3 struct sai_fifo_t

### Data Fields

- bool fifoContinueOneError
  *fifo continues when error occur*
- sai_fifo_packing_t fifoPacking
  *fifo packing mode*
- uint8_t fifoWatermark
  *fifo watermark*

### 38.3.2.4 struct sai_bit_clock_t

### Data Fields

- bool bclkSrcSwap
  *bit clock source swap*
- bool bclkInputDelay
  *bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .*
- sai_clock_polarity_t bclkPolarity
  *bit clock polarity*
- sai_bclk_source_t bclkSource
  *bit Clock source*

#### 38.3.2.4.0.39 Field Documentation

##### 38.3.2.4.0.39.1 bool sai_bit_clock_t::bclkInputDelay

### 38.3.2.5 struct sai_frame_sync_t

### Data Fields

- uint8_t frameSyncWidth
  *frame sync width in number of bit clocks*
- bool frameSyncEarly
  *TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.*
- sai_clock_polarity_t frameSyncPolarity
  *frame sync polarity*

#### 38.3.2.6 struct sai_serial_data_t

### Data Fields

- sai_data_order_t dataOrder
  *configure whether the LSB or MSB is transmitted first*
- uint8_t dataWord0Length
  *configure the number of bits in the first word in each frame*
- uint8_t dataWordNLength
  *configure the number of bits in the each word in each frame, except the first word*
- uint8_t dataWordLength
  *used to record the data length for dma transfer*
- uint8_t dataFirstBitShifted
  *Configure the bit index for the first bit transmitted for each word in the frame.*
- uint8_t dataWordNum
  *configure the number of words in each frame*
- uint32_t dataMaskedWord
  *configure whether the transmit word is masked*

#### 38.3.2.7 struct sai_transceiver_t

### Data Fields

- sai_serial_data_t serialData
  *serial data configurations*
- sai_frame_sync_t frameSync
  *ws configurations*
- sai_bit_clock_t bitClock
  *bit clock configurations*
- sai_fifo_t fifo
  *fifo configurations*
- sai_master_slave_t masterSlave
  *transceiver is master or slave*
- sai_sync_mode_t syncMode
  *transceiver sync mode*
- uint8_t startChannel
  *Transfer start channel.*
- uint8_t channelMask
  *enabled channel mask value, reference _sai_channel_mask*
- uint8_t endChannel
  *end channel number*
- uint8_t channelNums
  *Total enabled channel numbers.*

#### 38.3.2.8 struct sai_transfer_t

### Data Fields

- uint8_t * data

*Data start address to transfer.*
- size_t dataSize
  *Transfer size.*

#### 38.3.2.8.0.40 Field Documentation

#### 38.3.2.8.0.40.1 uint8_t* sai_transfer_t::data

#### 38.3.2.8.0.40.2 size_t sai_transfer_t::dataSize

### 38.3.2.9 struct _sai_handle

### Data Fields

- I2S_Type * base
  *base address*
- uint32_t state
  *Transfer status.*
- sai_transfer_callback_t callback
  *Callback function called at transfer event.*
- void * userData
  *Callback parameter passed to callback function.*
- uint8_t bitWidth
  *Bit width for transfer, 8/16/24/32 bits.*
- uint8_t channel
  *Transfer start channel.*
- uint8_t channelMask
  *enabled channel mask value, refernece _sai_channel_mask*
- uint8_t endChannel
  *end channel number*
- uint8_t channelNums
  *Total enabled channel numbers.*
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*
- uint8_t watermark
  *Watermark value.*

### 38.3.3 Macro Definition Documentation

#### 38.3.3.1 #define SAI_XFER_QUEUE_SIZE (4)

### 38.3.4 Enumeration Type Documentation

#### 38.3.4.1 enum _sai_status_t

Enumerator

***kStatus_SAI_TxBusy*** SAI Tx is busy.
***kStatus_SAI_RxBusy*** SAI Rx is busy.
***kStatus_SAI_TxError*** SAI Tx FIFO error.
***kStatus_SAI_RxError*** SAI Rx FIFO error.
***kStatus_SAI_QueueFull*** SAI transfer queue is full.
***kStatus_SAI_TxIdle*** SAI Tx is idle.
***kStatus_SAI_RxIdle*** SAI Rx is idle.

#### 38.3.4.2 enum _sai_channel_mask

Enumerator

***kSAI_Channel0Mask*** channel 0 mask value
***kSAI_Channel1Mask*** channel 1 mask value
***kSAI_Channel2Mask*** channel 2 mask value
***kSAI_Channel3Mask*** channel 3 mask value
***kSAI_Channel4Mask*** channel 4 mask value
***kSAI_Channel5Mask*** channel 5 mask value
***kSAI_Channel6Mask*** channel 6 mask value
***kSAI_Channel7Mask*** channel 7 mask value

#### 38.3.4.3 enum sai_protocol_t

Enumerator

***kSAI_BusLeftJustified*** Uses left justified format.
***kSAI_BusRightJustified*** Uses right justified format.
***kSAI_BusI2S*** Uses I2S format.
***kSAI_BusPCMA*** Uses I2S PCM A format.
***kSAI_BusPCMB*** Uses I2S PCM B format.

#### 38.3.4.4 enum sai_master_slave_t

Enumerator

***kSAI_Master*** Master mode include bclk and frame sync.
***kSAI_Slave*** Slave mode include bclk and frame sync.
***kSAI_Bclk_Master_FrameSync_Slave*** bclk in master mode, frame sync in slave mode
***kSAI_Bclk_Slave_FrameSync_Master*** bclk in slave mode, frame sync in master mode

#### 38.3.4.5 enum sai_mono_stereo_t

Enumerator

***kSAI_Stereo*** Stereo sound.
***kSAI_MonoRight*** Only Right channel have sound.
***kSAI_MonoLeft*** Only left channel have sound.

#### 38.3.4.6 enum sai_data_order_t

Enumerator

***kSAI_DataLSB*** LSB bit transferred first.
***kSAI_DataMSB*** MSB bit transferred first.

#### 38.3.4.7 enum sai_clock_polarity_t

Enumerator

***kSAI_PolarityActiveHigh*** Drive outputs on rising edge.
***kSAI_PolarityActiveLow*** Drive outputs on falling edge.
***kSAI_SampleOnFallingEdge*** Sample inputs on falling edge.
***kSAI_SampleOnRisingEdge*** Sample inputs on rising edge.

#### 38.3.4.8 enum sai_sync_mode_t

Enumerator

***kSAI_ModeAsync*** Asynchronous mode.
***kSAI_ModeSync*** Synchronous mode (with receiver or transmit)

#### 38.3.4.9 enum sai_mclk_source_t

Enumerator

***kSAI_MclkSourceSysclk*** Master clock from the system clock.
***kSAI_MclkSourceSelect1*** Master clock from source 1.
***kSAI_MclkSourceSelect2*** Master clock from source 2.
***kSAI_MclkSourceSelect3*** Master clock from source 3.

#### 38.3.4.10 enum sai_bclk_source_t

Enumerator

***kSAI_BclkSourceBusclk*** Bit clock using bus clock.
***kSAI_BclkSourceMclkOption1*** Bit clock MCLK option 1.
***kSAI_BclkSourceMclkOption2*** Bit clock MCLK option2.
***kSAI_BclkSourceMclkOption3*** Bit clock MCLK option3.
***kSAI_BclkSourceMclkDiv*** Bit clock using master clock divider.
***kSAI_BclkSourceOtherSai0*** Bit clock from other SAI device.
***kSAI_BclkSourceOtherSai1*** Bit clock from other SAI device.

#### 38.3.4.11 enum _sai_interrupt_enable_t

Enumerator

***kSAI_WordStartInterruptEnable*** Word start flag, means the first word in a frame detected.
***kSAI_SyncErrorInterruptEnable*** Sync error flag, means the sync error is detected.
***kSAI_FIFOWarningInterruptEnable*** FIFO warning flag, means the FIFO is empty.
***kSAI_FIFOErrorInterruptEnable*** FIFO error flag.
***kSAI_FIFORequestInterruptEnable*** FIFO request, means reached watermark.

#### 38.3.4.12 enum _sai_dma_enable_t

Enumerator

***kSAI_FIFOWarningDMAEnable*** FIFO warning caused by the DMA request.
***kSAI_FIFORequestDMAEnable*** FIFO request caused by the DMA request.

#### 38.3.4.13 enum _sai_flags

Enumerator

***kSAI_WordStartFlag*** Word start flag, means the first word in a frame detected.

*kSAI_SyncErrorFlag* Sync error flag, means the sync error is detected.
*kSAI_FIFOErrorFlag* FIFO error flag.
*kSAI_FIFORequestFlag* FIFO request flag.
*kSAI_FIFOWarningFlag* FIFO warning flag.

#### 38.3.4.14 enum sai_reset_type_t

Enumerator

*kSAI_ResetTypeSoftware* Software reset, reset the logic state.
*kSAI_ResetTypeFIFO* FIFO reset, reset the FIFO read and write pointer.
*kSAI_ResetAll* All reset.

#### 38.3.4.15 enum sai_fifo_packing_t

Enumerator

*kSAI_FifoPackingDisabled* Packing disabled.
*kSAI_FifoPacking8bit* 8 bit packing enabled
*kSAI_FifoPacking16bit* 16bit packing enabled

#### 38.3.4.16 enum sai_sample_rate_t

Enumerator

*kSAI_SampleRate8KHz* Sample rate 8000 Hz.
*kSAI_SampleRate11025Hz* Sample rate 11025 Hz.
*kSAI_SampleRate12KHz* Sample rate 12000 Hz.
*kSAI_SampleRate16KHz* Sample rate 16000 Hz.
*kSAI_SampleRate22050Hz* Sample rate 22050 Hz.
*kSAI_SampleRate24KHz* Sample rate 24000 Hz.
*kSAI_SampleRate32KHz* Sample rate 32000 Hz.
*kSAI_SampleRate44100Hz* Sample rate 44100 Hz.
*kSAI_SampleRate48KHz* Sample rate 48000 Hz.
*kSAI_SampleRate96KHz* Sample rate 96000 Hz.
*kSAI_SampleRate192KHz* Sample rate 192000 Hz.
*kSAI_SampleRate384KHz* Sample rate 384000 Hz.

#### 38.3.4.17 enum sai_word_width_t

Enumerator

*kSAI_WordWidth8bits* Audio data width 8 bits.

***kSAI_WordWidth16bits*** Audio data width 16 bits.
***kSAI_WordWidth24bits*** Audio data width 24 bits.
***kSAI_WordWidth32bits*** Audio data width 32 bits.

#### 38.3.4.18 enum sai_transceiver_type_t

Enumerator

***kSAI_Transmitter*** sai transmitter
***kSAI_Receiver*** sai receiver

#### 38.3.4.19 enum sai_frame_sync_len_t

Enumerator

***kSAI_FrameSyncLenOneBitClk*** 1 bit clock frame sync len for DSP mode
***kSAI_FrameSyncLenPerWordWidth*** Frame sync length decided by word width.

### 38.3.5 Function Documentation

#### 38.3.5.1 void SAI_TxInit ( I2S_Type * *base*, const sai_config_t * *config* )

Ungates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by SAI_TxGetDefaultConfig().

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *config* | SAI configuration structure. |

#### 38.3.5.2 void SAI_RxInit ( I2S_Type * *base*, const sai_config_t * *config* )

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by SAI_RxGetDefaultConfig().

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *config* | SAI configuration structure. |

#### 38.3.5.3 void SAI_TxGetDefaultConfig ( sai_config_t * *config* )

This API initializes the configuration structure for use in SAI_TxConfig(). The initialized structure can remain unchanged in SAI_TxConfig(), or it can be modified before calling SAI_TxConfig(). This is an example.

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

Parameters

| | |
|---|---|
| *config* | pointer to master configuration structure |

#### 38.3.5.4 void SAI_RxGetDefaultConfig ( sai_config_t * *config* )

This API initializes the configuration structure for use in SAI_RxConfig(). The initialized structure can remain unchanged in SAI_RxConfig() or it can be modified before calling SAI_RxConfig(). This is an example.

```
sai_config_t config;
SAI_RxGetDefaultConfig(&config);
```

Parameters

| | |
|---|---|
| *config* | pointer to master configuration structure |

#### 38.3.5.5 void SAI_Init ( I2S_Type * *base* )

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

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |

#### 38.3.5.6 void SAI_Deinit ( I2S_Type * *base* )

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |

#### 38.3.5.7 void SAI_TxReset ( I2S_Type * *base* )

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |

#### 38.3.5.8 void SAI_RxReset ( I2S_Type * *base* )

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |

#### 38.3.5.9 void SAI_TxEnable ( I2S_Type * *base*, bool *enable* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *enable* | True means enable SAI Tx, false means disable. |

#### 38.3.5.10 void SAI_RxEnable ( I2S_Type * *base*, bool *enable* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *enable* | True means enable SAI Rx, false means disable. |

#### 38.3.5.11 static void SAI_TxSetBitClockDirection ( I2S_Type * *base*, sai_master_slave_t *masterSlave* ) [inline], [static]

Select bit clock direction, master or slave.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | reference sai_master_slave_t. |

#### 38.3.5.12 static void SAI_RxSetBitClockDirection ( I2S_Type * *base*, sai_master_slave_t *masterSlave* ) [inline], [static]

Select bit clock direction, master or slave.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | reference sai_master_slave_t. |

#### 38.3.5.13 static void SAI_RxSetFrameSyncDirection ( I2S_Type * *base*, sai_master_slave_t *masterSlave* ) [inline], [static]

Select frame sync direction, master or slave.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | reference sai_master_slave_t. |

#### 38.3.5.14 static void SAI_TxSetFrameSyncDirection ( I2S_Type * *base*, sai_master_slave_t *masterSlave* ) [inline], [static]

Select frame sync direction, master or slave.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | reference sai_master_slave_t. |

#### 38.3.5.15 void SAI_TxSetBitClockRate ( I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *sourceClock-Hz,bit* | clock source frequency. |
| *sampleRate* | audio data sample rate. |
| *bitWidth,audio* | data bitWidth. |
| *channel-Numbers,audio* | channel numbers. |

#### 38.3.5.16 void SAI_RxSetBitClockRate ( I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *sourceClock-Hz,bit* | clock source frequency. |
| *sampleRate* | audio data sample rate. |
| *bitWidth,audio* | data bitWidth. |
| *channel-Numbers,audio* | channel numbers. |

#### 38.3.5.17 void SAI_TxSetBitclockConfig ( I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | master or slave. |
| *config* | bit clock other configurations, can be NULL in slave mode. |

#### 38.3.5.18 void SAI_RxSetBitclockConfig ( I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | master or slave. |
| *config* | bit clock other configurations, can be NULL in slave mode. |

#### 38.3.5.19 void SAI_TxSetFifoConfig ( I2S_Type * *base*, sai_fifo_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *config* | fifo configurations. |

#### 38.3.5.20 void SAI_RxSetFifoConfig ( I2S_Type * *base*, sai_fifo_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *config* | fifo configurations. |

#### 38.3.5.21 void SAI_TxSetFrameSyncConfig ( I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | master or slave. |
| *config* | frame sync configurations, can be NULL in slave mode. |

#### 38.3.5.22 void SAI_RxSetFrameSyncConfig ( I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *masterSlave* | master or slave. |
| *config* | frame sync configurations, can be NULL in slave mode. |

#### 38.3.5.23 void SAI_TxSetSerialDataConfig ( I2S_Type * *base*, sai_serial_data_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *config* | serial data configurations. |

#### 38.3.5.24 void SAI_RxSetSerialDataConfig ( I2S_Type * *base*, sai_serial_data_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *config* | serial data configurations. |

#### 38.3.5.25 void SAI_TxSetConfig ( I2S_Type * *base*, sai_transceiver_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *config* | transmitter configurations. |

#### 38.3.5.26 void SAI_RxSetConfig ( I2S_Type * *base*, sai_transceiver_t * *config* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *config* | receiver configurations. |

#### 38.3.5.27 void SAI_GetClassicI2SConfig ( sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask* )

Parameters

| | |
|---|---|
| *config* | transceiver configurations. |
| *bitWidth* | audio data bitWidth. |
| *mode* | audio data channel. |
| *saiChannel-Mask* | mask value of the channel to be enable. |

#### 38.3.5.28 void SAI_GetLeftJustifiedConfig ( sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask* )

Parameters

| | |
|---|---|
| *config* | transceiver configurations. |
| *bitWidth* | audio data bitWidth. |
| *mode* | audio data channel. |
| *saiChannel-Mask* | mask value of the channel to be enable. |

#### 38.3.5.29 void SAI_GetRightJustifiedConfig ( sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask* )

Parameters

| | |
|---|---|
| *config* | transceiver configurations. |
| *bitWidth* | audio data bitWidth. |
| *mode* | audio data channel. |
| *saiChannel-Mask* | mask value of the channel to be enable. |

#### 38.3.5.30 void SAI_GetTDMConfig ( sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, uint32_t *dataWordNum*, uint32_t *saiChannelMask* )

Parameters

| | |
|---|---|
| *config* | transceiver configurations. |
| *frameSync-Width* | length of frame sync. |
| *bitWidth* | audio data word width. |
| *dataWordNum* | word number in one frame. |
| *saiChannel-Mask* | mask value of the channel to be enable. |

#### 38.3.5.31 void SAI_GetDSPConfig ( sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask* )

Parameters

| | |
|---|---|
| *config* | transceiver configurations. |
| *frameSync-Width* | length of frame sync. |
| *bitWidth* | audio data bitWidth. |
| *mode* | audio data channel. |

| | |
|---|---|
| *saiChannel-Mask* | mask value of the channel to enable. |

#### 38.3.5.32 static uint32_t SAI_TxGetStatusFlag ( I2S_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |

Returns

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

#### 38.3.5.33 static void SAI_TxClearStatusFlags ( I2S_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | State mask. It can be a combination of the following source if defined:<br>• kSAI_WordStartFlag<br>• kSAI_SyncErrorFlag<br>• kSAI_FIFOErrorFlag |

#### 38.3.5.34 static uint32_t SAI_RxGetStatusFlag ( I2S_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

#### 38.3.5.35 static void SAI_RxClearStatusFlags ( I2S_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | State mask. It can be a combination of the following sources if defined.<br>• kSAI_WordStartFlag<br>• kSAI_SyncErrorFlag<br>• kSAI_FIFOErrorFlag |

#### 38.3.5.36 void SAI_TxSoftwareReset ( I2S_Type * *base*, sai_reset_type_t *type* )

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1∼TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *type* | Reset type, FIFO reset or software reset |

#### 38.3.5.37 void SAI_RxSoftwareReset ( I2S_Type * *base*, sai_reset_type_t *type* )

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1∼RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *type* | Reset type, FIFO reset or software reset |

#### 38.3.5.38 void SAI_TxSetChannelFIFOMask ( I2S_Type * *base*, uint8_t *mask* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled. |

#### 38.3.5.39 void SAI_RxSetChannelFIFOMask ( I2S_Type * *base*, uint8_t *mask* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled. |

#### 38.3.5.40 void SAI_TxSetDataOrder ( I2S_Type * *base*, sai_data_order_t *order* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *order* | Data order MSB or LSB |

#### 38.3.5.41 void SAI_RxSetDataOrder ( I2S_Type * *base*, sai_data_order_t *order* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *order* | Data order MSB or LSB |

#### 38.3.5.42 void SAI_TxSetBitClockPolarity ( I2S_Type * *base*, sai_clock_polarity_t *polarity* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *order* | Data order MSB or LSB |

#### 38.3.5.43 void SAI_RxSetBitClockPolarity ( I2S_Type * *base*, sai_clock_polarity_t *polarity* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *order* | Data order MSB or LSB |

#### 38.3.5.44 void SAI_TxSetFrameSyncPolarity ( I2S_Type * *base*, sai_clock_polarity_t *polarity* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *order* | Data order MSB or LSB |

#### 38.3.5.45 void SAI_RxSetFrameSyncPolarity ( I2S_Type * *base*, sai_clock_polarity_t *polarity* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *order* | Data order MSB or LSB |

#### 38.3.5.46 void SAI_TxSetFIFOPacking ( I2S_Type * *base*, sai_fifo_packing_t *pack* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *pack* | FIFO pack type. It is element of sai_fifo_packing_t. |

#### 38.3.5.47 void SAI_RxSetFIFOPacking ( I2S_Type * *base*, sai_fifo_packing_t *pack* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *pack* | FIFO pack type. It is element of sai_fifo_packing_t. |

#### 38.3.5.48 static void SAI_TxSetFIFOErrorContinue ( I2S_Type * *base,* bool *isEnabled* ) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *isEnabled* | Is FIFO error continue enabled, true means enable, false means disable. |

#### 38.3.5.49 static void SAI_RxSetFIFOErrorContinue ( I2S_Type * *base,* bool *isEnabled* ) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *isEnabled* | Is FIFO error continue enabled, true means enable, false means disable. |

#### 38.3.5.50 static void SAI_TxEnableInterrupts ( I2S_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | interrupt source The parameter can be a combination of the following sources if defined.<br>• kSAI_WordStartInterruptEnable<br>• kSAI_SyncErrorInterruptEnable<br>• kSAI_FIFOWarningInterruptEnable<br>• kSAI_FIFORequestInterruptEnable<br>• kSAI_FIFOErrorInterruptEnable |

#### 38.3.5.51 static void SAI_RxEnableInterrupts ( I2S_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | interrupt source The parameter can be a combination of the following sources if defined.<br>• kSAI_WordStartInterruptEnable<br>• kSAI_SyncErrorInterruptEnable<br>• kSAI_FIFOWarningInterruptEnable<br>• kSAI_FIFORequestInterruptEnable<br>• kSAI_FIFOErrorInterruptEnable |

#### 38.3.5.52 static void SAI_TxDisableInterrupts ( I2S_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | interrupt source The parameter can be a combination of the following sources if defined.<br>• kSAI_WordStartInterruptEnable<br>• kSAI_SyncErrorInterruptEnable<br>• kSAI_FIFOWarningInterruptEnable<br>• kSAI_FIFORequestInterruptEnable<br>• kSAI_FIFOErrorInterruptEnable |

#### 38.3.5.53 static void SAI_RxDisableInterrupts ( I2S_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | interrupt source The parameter can be a combination of the following sources if defined.<br>• kSAI_WordStartInterruptEnable<br>• kSAI_SyncErrorInterruptEnable<br>• kSAI_FIFOWarningInterruptEnable<br>• kSAI_FIFORequestInterruptEnable<br>• kSAI_FIFOErrorInterruptEnable |

#### 38.3.5.54 static void SAI_TxEnableDMA ( I2S_Type * *base,* uint32_t *mask,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | DMA source The parameter can be combination of the following sources if defined.<br>• kSAI_FIFOWarningDMAEnable<br>• kSAI_FIFORequestDMAEnable |
| *enable* | True means enable DMA, false means disable DMA. |

#### 38.3.5.55 static void SAI_RxEnableDMA ( I2S_Type * *base*, uint32_t *mask*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *mask* | DMA source The parameter can be a combination of the following sources if defined.<br>• kSAI_FIFOWarningDMAEnable<br>• kSAI_FIFORequestDMAEnable |
| *enable* | True means enable DMA, false means disable DMA. |

#### 38.3.5.56 static uint32_t SAI_TxGetDataRegisterAddress ( I2S_Type * *base*, uint32_t *channel* ) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Which data channel used. |

Returns

data register address.

#### 38.3.5.57 static uint32_t SAI_RxGetDataRegisterAddress ( I2S_Type * *base*, uint32_t *channel* ) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Which data channel used. |

Returns

data register address.

#### 38.3.5.58 void SAI_TxSetFormat ( I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *format* | Pointer to the SAI audio data format structure. |
| *mclkSourceClockHz* | SAI master clock source frequency in Hz. |
| *bclkSourceClockHz* | SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz. |

#### 38.3.5.59 void SAI_RxSetFormat ( I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *format* | Pointer to the SAI audio data format structure. |
| *mclkSourceClockHz* | SAI master clock source frequency in Hz. |
| *bclkSourceClockHz* | SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz. |

#### 38.3.5.60 void SAI_WriteBlocking ( I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size* )

Note

This function blocks by polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Data channel used. |
| *bitWidth* | How many bits in an audio word; usually 8/16/24/32 bits. |
| *buffer* | Pointer to the data to be written. |
| *size* | Bytes to be written. |

#### 38.3.5.61 void SAI_WriteMultiChannelBlocking ( I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size* )

Note

This function blocks by polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Data channel used. |
| *channelMask* | channel mask. |
| *bitWidth* | How many bits in an audio word; usually 8/16/24/32 bits. |
| *buffer* | Pointer to the data to be written. |
| *size* | Bytes to be written. |

#### 38.3.5.62 static void SAI_WriteData ( I2S_Type * *base*, uint32_t *channel*, uint32_t *data* ) `[inline], [static]`

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Data channel used. |
| *data* | Data needs to be written. |

#### 38.3.5.63 void SAI_ReadBlocking ( I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size* )

Note

This function blocks by polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Data channel used. |
| *bitWidth* | How many bits in an audio word; usually 8/16/24/32 bits. |
| *buffer* | Pointer to the data to be read. |
| *size* | Bytes to be read. |

#### 38.3.5.64 void SAI_ReadMultiChannelBlocking ( I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size* )

Note

This function blocks by polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Data channel used. |
| *channelMask* | channel mask. |
| *bitWidth* | How many bits in an audio word; usually 8/16/24/32 bits. |
| *buffer* | Pointer to the data to be read. |
| *size* | Bytes to be read. |

#### 38.3.5.65 static uint32_t SAI_ReadData ( I2S_Type * *base*, uint32_t *channel* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *channel* | Data channel used. |

Returns

Data in SAI FIFO.

#### 38.3.5.66 void SAI_TransferTxCreateHandle ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData* )

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | SAI handle pointer. |
| *callback* | Pointer to the user callback function. |
| *userData* | User parameter passed to the callback function |

#### 38.3.5.67 void SAI_TransferRxCreateHandle ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData* )

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI handle pointer. |
| *callback* | Pointer to the user callback function. |
| *userData* | User parameter passed to the callback function. |

#### 38.3.5.68 void SAI_TransferTxSetConfig ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config* )

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI handle pointer. |
| *config* | tranmitter configurations. |

#### 38.3.5.69 void SAI_TransferRxSetConfig ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config* )

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI handle pointer. |
| *config* | receiver configurations. |

#### 38.3.5.70 status_t SAI_TransferTxSetFormat ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI handle pointer. |
| *format* | Pointer to the SAI audio data format structure. |
| *mclkSourceClockHz* | SAI master clock source frequency in Hz. |
| *bclkSourceClockHz* | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format. |

Returns

Status of this function. Return value is the status_t.

#### 38.3.5.71 status_t SAI_TransferRxSetFormat ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI handle pointer. |
| *format* | Pointer to the SAI audio data format structure. |
| *mclkSource-ClockHz* | SAI master clock source frequency in Hz. |
| *bclkSource-ClockHz* | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format. |

Returns

Status of this function. Return value is one of status_t.

### 38.3.5.72 status_t SAI_TransferSendNonBlocking ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_-SAI_Busy, the transfer is finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | Pointer to the sai_handle_t structure which stores the transfer state. |
| *xfer* | Pointer to the sai_transfer_t structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully started the data receive. |
| *kStatus_SAI_TxBusy* | Previous receive still not finished. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

### 38.3.5.73 status_t SAI_TransferReceiveNonBlocking ( I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_-SAI_Busy, the transfer is finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | Pointer to the sai_handle_t structure which stores the transfer state. |
| *xfer* | Pointer to the sai_transfer_t structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully started the data receive. |
| *kStatus_SAI_RxBusy* | Previous receive still not finished. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

#### 38.3.5.74 status_t SAI_TransferGetSendCount ( I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | Pointer to the sai_handle_t structure which stores the transfer state. |
| *count* | Bytes count sent. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferIn-Progress* | There is not a non-blocking transaction currently in progress. |

#### 38.3.5.75 status_t SAI_TransferGetReceiveCount ( I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | Pointer to the sai_handle_t structure which stores the transfer state. |
| *count* | Bytes count received. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

#### 38.3.5.76 void SAI_TransferAbortSend ( I2S_Type * *base*, sai_handle_t * *handle* )

Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | Pointer to the sai_handle_t structure which stores the transfer state. |

#### 38.3.5.77 void SAI_TransferAbortReceive ( I2S_Type * *base*, sai_handle_t * *handle* )

Note

This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | Pointer to the sai_handle_t structure which stores the transfer state. |

#### 38.3.5.78 void SAI_TransferTerminateSend ( I2S_Type * *base*, sai_handle_t * *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |

#### 38.3.5.79 void SAI_TransferTerminateReceive ( I2S_Type * *base*, sai_handle_t * *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |

#### 38.3.5.80 void SAI_TransferTxHandleIRQ ( I2S_Type * *base*, sai_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | Pointer to the sai_handle_t structure. |

#### 38.3.5.81 void SAI_TransferRxHandleIRQ ( I2S_Type * *base*, sai_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | Pointer to the sai_handle_t structure. |

## 38.4 SAI DMA Driver

### 38.4.1 Overview

#### Data Structures

- struct sai_dma_handle_t
  *SAI DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef void(∗ sai_dma_callback_t )(I2S_Type ∗base, sai_dma_handle_t ∗handle, status_t status, void ∗userData)
  *Define SAI DMA callback.*

#### Driver version

- #define FSL_SAI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  *Version 2.2.0.*

#### DMA Transactional

- void SAI_TransferTxCreateHandleDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle, sai_dma_callback_t callback, void ∗userData, dma_handle_t ∗dmaHandle)
  *Initializes the SAI master DMA handle.*
- void SAI_TransferRxCreateHandleDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle, sai_dma_callback_t callback, void ∗userData, dma_handle_t ∗dmaHandle)
  *Initializes the SAI slave DMA handle.*
- void SAI_TransferTxSetFormatDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Tx audio format.*
- void SAI_TransferRxSetFormatDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Rx audio format.*
- status_t SAI_TransferSendDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle, sai_transfer_t ∗xfer)
  *Performs a non-blocking SAI transfer using DMA.*
- status_t SAI_TransferReceiveDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle, sai_transfer_t ∗xfer)
  *Performs a non-blocking SAI transfer using DMA.*
- void SAI_TransferAbortSendDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle)
  *Aborts a SAI transfer using DMA.*
- void SAI_TransferAbortReceiveDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle)
  *Aborts a SAI transfer using DMA.*
- status_t SAI_TransferGetSendCountDMA (I2S_Type ∗base, sai_dma_handle_t ∗handle, size_t ∗count)

*Gets byte count sent by SAI.*

- status_t SAI_TransferGetReceiveCountDMA (I2S_Type *base, sai_dma_handle_t *handle, size_t *count)
  *Gets byte count received by SAI.*
- void SAI_TransferTxSetConfigDMA (I2S_Type *base, sai_dma_handle_t *handle, sai_transceiver_t *saiConfig)
  *Configures the SAI Tx.*
- void SAI_TransferRxSetConfigDMA (I2S_Type *base, sai_dma_handle_t *handle, sai_transceiver_t *saiConfig)
  *Configures the SAI Rx.*

### 38.4.2 Data Structure Documentation

#### 38.4.2.1 struct _sai_dma_handle

#### Data Fields

- dma_handle_t * dmaHandle
  *DMA handler for SAI send.*
- uint8_t bytesPerFrame
  *Bytes in a frame.*
- uint8_t channel
  *Which Data channel SAI use.*
- uint32_t state
  *SAI DMA transfer internal state.*
- sai_dma_callback_t callback
  *Callback for users while transfer finish or error occurred.*
- void * userData
  *User callback parameter.*
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*

#### 38.4.2.1.0.41 Field Documentation

##### 38.4.2.1.0.41.1 sai_transfer_t sai_dma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]

##### 38.4.2.1.0.41.2 volatile uint8_t sai_dma_handle_t::queueUser

### 38.4.3 Function Documentation

#### 38.4.3.1 void SAI_TransferTxCreateHandleDMA ( I2S_Type * *base,* sai_dma_handle_t * *handle,* sai_dma_callback_t *callback,* void * *userData,* dma_handle_t * *dmaHandle* )

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

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |
| *base* | SAI peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaHandle* | DMA handle pointer, this handle shall be static allocated by users. |

#### 38.4.3.2 void SAI_TransferRxCreateHandleDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, sai_dma_callback_t *callback*, void * *userData*, dma_handle_t * *dmaHandle* )

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |
| *base* | SAI peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaHandle* | DMA handle pointer, this handle shall be static allocated by users. |

#### 38.4.3.3 void SAI_TransferTxSetFormatDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to the format.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |

| | |
|---|---|
| *handle* | SAI DMA handle pointer. |
| *format* | Pointer to SAI audio data format structure. |
| *mclkSource-ClockHz* | SAI master clock source frequency in Hz. |
| *bclkSource-ClockHz* | SAI bit clock source frequency in Hz. If bit clock source is master. clock, this value should equals to masterClockHz in format. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |

#### 38.4.3.4 void SAI_TransferRxSetFormatDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets eDMA parameter according to format.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |
| *format* | Pointer to SAI audio data format structure. |
| *mclkSource-ClockHz* | SAI master clock source frequency in Hz. |
| *bclkSource-ClockHz* | SAI bit clock source frequency in Hz. If bit clock source is master. clock, this value should equals to masterClockHz in format. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input arguments is invalid. |

#### 38.4.3.5 status_t SAI_TransferSendDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetTransferStatus to poll the transfer status to check whether the SAI transfer finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start the data receive. |
| *kStatus_SAI_TxBusy* | Previous receive still not finished. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

#### 38.4.3.6 status_t SAI_TransferReceiveDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This interface returns immediately after transfer initiates. Call SAI_GetTransferStatus to poll the transfer status to check whether the SAI transfer is finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | SAI DMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully start the data receive. |
| *kStatus_SAI_RxBusy* | Previous receive still not finished. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

#### 38.4.3.7 void SAI_TransferAbortSendDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |

#### 38.4.3.8 void SAI_TransferAbortReceiveDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |

#### 38.4.3.9 status_t SAI_TransferGetSendCountDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |
| *count* | Bytes count sent by SAI. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

#### 38.4.3.10 status_t SAI_TransferGetReceiveCountDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |

| | |
|---|---|
| *handle* | SAI DMA handle pointer. |
| *count* | Bytes count received by SAI. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

#### 38.4.3.11 void SAI_TransferTxSetConfigDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, sai_transceiver_t * *saiConfig* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |
| *saiConfig* | sai configurations. |

#### 38.4.3.12 void SAI_TransferRxSetConfigDMA ( I2S_Type * *base*, sai_dma_handle_t * *handle*, sai_transceiver_t * *saiConfig* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI DMA handle pointer. |
| *saiConfig* | sai configurations. |

## 38.5 SAI EDMA Driver

### 38.5.1 Overview

#### Data Structures

- struct sai_edma_handle_t
  *SAI DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef void(∗ sai_edma_callback_t )(I2S_Type ∗base, sai_edma_handle_t ∗handle, status_t status, void ∗userData)
  *SAI eDMA transfer callback function for finish and error.*

#### Driver version

- #define FSL_SAI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  *Version 2.2.0.*

#### eDMA Transactional

- void SAI_TransferTxCreateHandleEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_edma_callback_t callback, void ∗userData, edma_handle_t ∗dmaHandle)
  *Initializes the SAI eDMA handle.*
- void SAI_TransferRxCreateHandleEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_edma_callback_t callback, void ∗userData, edma_handle_t ∗dmaHandle)
  *Initializes the SAI Rx eDMA handle.*
- void SAI_TransferTxSetFormatEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Tx audio format.*
- void SAI_TransferRxSetFormatEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Rx audio format.*
- void SAI_TransferTxSetConfigEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_transceiver_t ∗saiConfig)
  *Configures the SAI Tx.*
- void SAI_TransferRxSetConfigEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_transceiver_t ∗saiConfig)
  *Configures the SAI Rx.*
- status_t SAI_TransferSendEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_transfer_t ∗xfer)
  *Performs a non-blocking SAI transfer using DMA.*
- status_t SAI_TransferReceiveEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, sai_transfer_t ∗xfer)

*Performs a non-blocking SAI receive using eDMA.*
- void SAI_TransferTerminateSendEDMA (I2S_Type *base, sai_edma_handle_t *handle)
  *Terminate all SAI send.*
- void SAI_TransferTerminateReceiveEDMA (I2S_Type *base, sai_edma_handle_t *handle)
  *Terminate all SAI receive.*
- void SAI_TransferAbortSendEDMA (I2S_Type *base, sai_edma_handle_t *handle)
  *Aborts a SAI transfer using eDMA.*
- void SAI_TransferAbortReceiveEDMA (I2S_Type *base, sai_edma_handle_t *handle)
  *Aborts a SAI receive using eDMA.*
- status_t SAI_TransferGetSendCountEDMA (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
  *Gets byte count sent by SAI.*
- status_t SAI_TransferGetReceiveCountEDMA (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
  *Gets byte count received by SAI.*

## 38.5.2 Data Structure Documentation

### 38.5.2.1 struct _sai_edma_handle

### Data Fields

- edma_handle_t * dmaHandle
  *DMA handler for SAI send.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- uint8_t bytesPerFrame
  *Bytes in a frame.*
- uint8_t channel
  *Which data channel.*
- uint8_t count
  *The transfer data count in a DMA request.*
- uint32_t state
  *Internal state for SAI eDMA transfer.*
- sai_edma_callback_t callback
  *Callback for users while transfer finish or error occurs.*
- void * userData
  *User callback parameter.*
- uint8_t tcd [(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]
  *TCD pool for eDMA transfer.*
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*

#### 38.5.2.1.0.42 Field Documentation

#### 38.5.2.1.0.42.1 uint8_t sai_edma_handle_t::nbytes

#### 38.5.2.1.0.42.2 uint8_t sai_edma_handle_t::tcd[(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]

#### 38.5.2.1.0.42.3 sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]

#### 38.5.2.1.0.42.4 volatile uint8_t sai_edma_handle_t::queueUser

### 38.5.3 Function Documentation

#### 38.5.3.1 void SAI_TransferTxCreateHandleEDMA ( I2S_Type * *base,* sai_edma_handle_t * *handle,* sai_edma_callback_t *callback,* void * *userData,* edma_handle_t * *dmaHandle* )

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

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |
| *base* | SAI peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaHandle* | eDMA handle pointer, this handle shall be static allocated by users. |

#### 38.5.3.2 void SAI_TransferRxCreateHandleEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_edma_callback_t *callback*, void * *userData*, edma_handle_t * *dmaHandle* )

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |
| *base* | SAI peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaHandle* | eDMA handle pointer, this handle shall be static allocated by users. |

#### 38.5.3.3 void SAI_TransferTxSetFormatEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |

| | |
|---|---|
| *handle* | SAI eDMA handle pointer. |
| *format* | Pointer to SAI audio data format structure. |
| *mclkSource-ClockHz* | SAI master clock source frequency in Hz. |
| *bclkSource-ClockHz* | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |

#### 38.5.3.4 void SAI_TransferRxSetFormatEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |
| *format* | Pointer to SAI audio data format structure. |
| *mclkSource-ClockHz* | SAI master clock source frequency in Hz. |
| *bclkSource-ClockHz* | SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |

#### 38.5.3.5 void SAI_TransferTxSetConfigEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |
| *saiConfig* | sai configurations. |

#### 38.5.3.6 void SAI_TransferRxSetConfigEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |
| *saiConfig* | sai configurations. |

#### 38.5.3.7 status_t SAI_TransferSendEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |
| *xfer* | Pointer to the DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a SAI eDMA send successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |
| *kStatus_TxBusy* | SAI is busy sending data. |

#### 38.5.3.8 status_t SAI_TransferReceiveEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | SAI eDMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a SAI eDMA receive successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |
| *kStatus_RxBusy* | SAI is busy receiving data. |

#### 38.5.3.9 void SAI_TransferTerminateSendEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |

#### 38.5.3.10 void SAI_TransferTerminateReceiveEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |

| | |
|---|---|
| *handle* | SAI eDMA handle pointer. |

#### 38.5.3.11 void SAI_TransferAbortSendEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle* )

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |

#### 38.5.3.12 void SAI_TransferAbortReceiveEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle* )

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | SAI eDMA handle pointer. |

#### 38.5.3.13 status_t SAI_TransferGetSendCountEDMA ( I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI eDMA handle pointer. |
| *count* | Bytes count sent by SAI. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is no non-blocking transaction in progress. |

#### 38.5.3.14 status_t SAI_TransferGetReceiveCountEDMA ( I2S_Type * *base,* sai_edma_handle_t * *handle,* size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | SAI eDMA handle pointer. |
| *count* | Bytes count received by SAI. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is no non-blocking transaction in progress. |

## 38.6 SAI SDMA Driver

### 38.6.1 Overview

#### Data Structures

- struct sai_sdma_handle_t
  *SAI DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef void(∗ sai_sdma_callback_t )(I2S_Type ∗base, sai_sdma_handle_t ∗handle, status_t status, void ∗userData)
  *SAI SDMA transfer callback function for finish and error.*

#### Driver version

- #define FSL_SAI_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  *Version 2.2.0.*

#### SDMA Transactional

- void SAI_TransferTxCreateHandleSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_sdma_callback_t callback, void ∗userData, sdma_handle_t ∗dmaHandle, uint32_t eventSource)
  *Initializes the SAI SDMA handle.*
- void SAI_TransferRxCreateHandleSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_sdma_callback_t callback, void ∗userData, sdma_handle_t ∗dmaHandle, uint32_t eventSource)
  *Initializes the SAI Rx SDMA handle.*
- void SAI_TransferTxSetFormatSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Tx audio format.*
- void SAI_TransferRxSetFormatSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  *Configures the SAI Rx audio format.*
- status_t SAI_TransferSendSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_t ∗xfer)
  *Performs a non-blocking SAI transfer using DMA.*
- status_t SAI_TransferReceiveSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_t ∗xfer)
  *Performs a non-blocking SAI receive using SDMA.*
- void SAI_TransferAbortSendSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle)
  *Aborts a SAI transfer using SDMA.*
- void SAI_TransferAbortReceiveSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle)
  *Aborts a SAI receive using SDMA.*

- void SAI_TransferRxSetConfigSDMA (I2S_Type *base, sai_sdma_handle_t *handle, sai_transceiver_t *saiConfig)
  *brief Configures the SAI RX.*
- void SAI_TransferTxSetConfigSDMA (I2S_Type *base, sai_sdma_handle_t *handle, sai_transceiver_t *saiConfig)
  *brief Configures the SAI Tx.*

## 38.6.2 Data Structure Documentation

### 38.6.2.1 struct _sai_sdma_handle

#### Data Fields

- sdma_handle_t * dmaHandle
  *DMA handler for SAI send.*
- uint8_t bytesPerFrame
  *Bytes in a frame.*
- uint8_t channel
  *start data channel*
- uint8_t channelNums
  *total transfer channel numbers, used for multififo*
- uint8_t channelMask
  *enabled channel mask value, refernece _sai_channel_mask*
- uint8_t fifoOffset
  *fifo address offset between multifo*
- uint8_t count
  *The transfer data count in a DMA request.*
- uint32_t state
  *Internal state for SAI SDMA transfer.*
- uint32_t eventSource
  *SAI event source number.*
- sai_sdma_callback_t callback
  *Callback for users while transfer finish or error occurs.*
- void * userData
  *User callback parameter.*
- sdma_buffer_descriptor_t bdPool [SAI_XFER_QUEUE_SIZE]
  *BD pool for SDMA transfer.*
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*

#### 38.6.2.1.0.43 Field Documentation

##### 38.6.2.1.0.43.1 sdma_buffer_descriptor_t sai_sdma_handle_t::bdPool[SAI_XFER_QUEUE_SIZE]

##### 38.6.2.1.0.43.2 sai_transfer_t sai_sdma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]

##### 38.6.2.1.0.43.3 volatile uint8_t sai_sdma_handle_t::queueUser

### 38.6.3 Function Documentation

#### 38.6.3.1 void SAI_TransferTxCreateHandleSDMA ( I2S_Type * *base,* sai_sdma_handle_t * *handle,* sai_sdma_callback_t *callback,* void * *userData,* sdma_handle_t * *dmaHandle,* uint32_t *eventSource* )

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

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI SDMA handle pointer. |
| *base* | SAI peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaHandle* | SDMA handle pointer, this handle shall be static allocated by users. |

#### 38.6.3.2 void SAI_TransferRxCreateHandleSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle*, sai_sdma_callback_t *callback*, void * *userData*, sdma_handle_t * *dmaHandle*, uint32_t *eventSource* )

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI SDMA handle pointer. |
| *base* | SAI peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaHandle* | SDMA handle pointer, this handle shall be static allocated by users. |

#### 38.6.3.3 void SAI_TransferTxSetFormatSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |

| | |
|---|---|
| *handle* | SAI SDMA handle pointer. |
| *format* | Pointer to SAI audio data format structure. |
| *mclkSource-ClockHz* | SAI master clock source frequency in Hz. |
| *bclkSource-ClockHz* | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |

#### 38.6.3.4 void SAI_TransferRxSetFormatSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI SDMA handle pointer. |
| *format* | Pointer to SAI audio data format structure. |
| *mclkSource-ClockHz* | SAI master clock source frequency in Hz. |
| *bclkSource-ClockHz* | SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format. |

Return values

| | |
|---|---|
| *kStatus_Success* | Audio format set successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |

#### 38.6.3.5 status_t SAI_TransferSendSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI SDMA handle pointer. |
| *xfer* | Pointer to the DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a SAI SDMA send successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |
| *kStatus_TxBusy* | SAI is busy sending data. |

#### 38.6.3.6 status_t SAI_TransferReceiveSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle*, sai_transfer_t * *xfer* )

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | SAI SDMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a SAI SDMA receive successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |
| *kStatus_RxBusy* | SAI is busy receiving data. |

#### 38.6.3.7 void SAI_TransferAbortSendSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer. |
| *handle* | SAI SDMA handle pointer. |

#### 38.6.3.8 void SAI_TransferAbortReceiveSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SAI base pointer |
| *handle* | SAI SDMA handle pointer. |

#### 38.6.3.9 void SAI_TransferRxSetConfigSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle*, sai_transceiver_t * *saiConfig* )

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

#### 38.6.3.10 void SAI_TransferTxSetConfigSDMA ( I2S_Type * *base*, sai_sdma_handle_t * *handle*, sai_transceiver_t * *saiConfig* )

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

# Chapter 39
# SEMC: Smart External DRAM Controller Driver

## 39.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Smart External DRAM Controller block of MCUXpresso SDK devices.

The SEMC Initialize is to initialize for common configure: gate the SEMC clock, configure IOMUX, and queue weight setting. The SEMC Deinitialize is to ungate the clock and disable SEMC module.

The interrupt and disable operation for SEMC.

This group is mainly provide NAND/NOR memory access API which is through IP bus/ IP command access. Since the AXI access is directly read/write is so easy, so the AXI read/write part is not provided in SEMC.

## 39.2 Typical use case

## Data Structures

- struct semc_sdram_config_t
  *SEMC SDRAM configuration structure. More...*
- struct semc_nand_timing_config_t
  *SEMC NAND device timing configuration structure. More...*
- struct semc_nand_config_t
  *SEMC NAND configuration structure. More...*
- struct semc_nor_config_t
  *SEMC NOR configuration structure. More...*
- struct semc_sram_config_t
  *SEMC SRAM configuration structure. More...*
- struct semc_dbi_config_t
  *SEMC DBI configuration structure. More...*
- struct semc_queuea_weight_struct_t
  *SEMC AXI queue a weight setting structure. More...*
- union semc_queuea_weight_t
  *SEMC AXI queue a weight setting union. More...*
- struct semc_queueb_weight_struct_t
  *SEMC AXI queue b weight setting structure. More...*
- union semc_queueb_weight_t
  *SEMC AXI queue b weight setting union. More...*
- struct semc_axi_queueweight_t
  *SEMC AXI queue weight setting. More...*
- struct semc_config_t
  *SEMC configuration structure. More...*

## Enumerations

- enum _semc_status

*SEMC status.*

- enum semc_mem_type_t {
  kSEMC_MemType_SDRAM = 0,
  kSEMC_MemType_SRAM,
  kSEMC_MemType_NOR,
  kSEMC_MemType_NAND,
  kSEMC_MemType_8080 }
  *SEMC memory device type.*
- enum semc_waitready_polarity_t {
  kSEMC_LowActive = 0,
  kSEMC_HighActive }
  *SEMC WAIT/RDY polarity.*
- enum semc_sdram_cs_t {
  kSEMC_SDRAM_CS0 = 0,
  kSEMC_SDRAM_CS1,
  kSEMC_SDRAM_CS2,
  kSEMC_SDRAM_CS3 }
  *SEMC SDRAM Chip selection .*
- enum semc_nand_access_type_t
  *SEMC NAND device type.*
- enum semc_interrupt_enable_t {
  kSEMC_IPCmdDoneInterrupt = SEMC_INTEN_IPCMDDONEEN_MASK,
  kSEMC_IPCmdErrInterrupt = SEMC_INTEN_IPCMDERREN_MASK,
  kSEMC_AXICmdErrInterrupt = SEMC_INTEN_AXICMDERREN_MASK,
  kSEMC_AXIBusErrInterrupt = SEMC_INTEN_AXIBUSERREN_MASK }
  *SEMC interrupts .*
- enum semc_ipcmd_datasize_t {
  kSEMC_IPcmdDataSize_1bytes = 1,
  kSEMC_IPcmdDataSize_2bytes,
  kSEMC_IPcmdDataSize_3bytes,
  kSEMC_IPcmdDataSize_4bytes }
  *SEMC IP command data size in bytes.*
- enum semc_refresh_time_t {
  kSEMC_RefreshThreeClocks = 0x0U,
  kSEMC_RefreshSixClocks,
  kSEMC_RefreshNineClocks }
  *SEMC auto-refresh timing.*
- enum semc_caslatency_t {
  kSEMC_LatencyOne = 1,
  kSEMC_LatencyTwo,
  kSEMC_LatencyThree }
  *CAS latency.*
- enum semc_sdram_column_bit_num_t {
  kSEMC_SdramColunm_12bit = 0x0U,
  kSEMC_SdramColunm_11bit,
  kSEMC_SdramColunm_10bit,
  kSEMC_SdramColunm_9bit }

*SEMC sdram column address bit number.*

- enum sem_sdram_burst_len_t {
  kSEMC_Sdram_BurstLen1 = 0,
  kSEMC_Sdram_BurstLen2,
  kSEMC_Sdram_BurstLen4,
  kSEMC_Sdram_BurstLen8 }
  *SEMC sdram burst length.*
- enum semc_nand_column_bit_num_t {
  kSEMC_NandColum_16bit = 0x0U,
  kSEMC_NandColum_15bit,
  kSEMC_NandColum_14bit,
  kSEMC_NandColum_13bit,
  kSEMC_NandColum_12bit,
  kSEMC_NandColum_11bit,
  kSEMC_NandColum_10bit,
  kSEMC_NandColum_9bit }
  *SEMC nand column address bit number.*
- enum sem_nand_burst_len_t {
  kSEMC_Nand_BurstLen1 = 0,
  kSEMC_Nand_BurstLen2,
  kSEMC_Nand_BurstLen4,
  kSEMC_Nand_BurstLen8,
  kSEMC_Nand_BurstLen16,
  kSEMC_Nand_BurstLen32,
  kSEMC_Nand_BurstLen64 }
  *SEMC nand burst length.*
- enum semc_norsram_column_bit_num_t {
  kSEMC_NorColum_12bit = 0x0U,
  kSEMC_NorColum_11bit,
  kSEMC_NorColum_10bit,
  kSEMC_NorColum_9bit,
  kSEMC_NorColum_8bit,
  kSEMC_NorColum_7bit,
  kSEMC_NorColum_6bit,
  kSEMC_NorColum_5bit,
  kSEMC_NorColum_4bit,
  kSEMC_NorColum_3bit,
  kSEMC_NorColum_2bit }
  *SEMC nor/sram column address bit number.*
- enum sem_norsram_burst_len_t {
  kSEMC_Nor_BurstLen1 = 0,
  kSEMC_Nor_BurstLen2,
  kSEMC_Nor_BurstLen4,
  kSEMC_Nor_BurstLen8,
  kSEMC_Nor_BurstLen16,
  kSEMC_Nor_BurstLen32,

kSEMC_Nor_BurstLen64 }
*SEMC nor/sram burst length.*

- enum semc_dbi_column_bit_num_t {
  kSEMC_Dbi_Colum_12bit = 0x0U,
  kSEMC_Dbi_Colum_11bit,
  kSEMC_Dbi_Colum_10bit,
  kSEMC_Dbi_Colum_9bit,
  kSEMC_Dbi_Colum_8bit,
  kSEMC_Dbi_Colum_7bit,
  kSEMC_Dbi_Colum_6bit,
  kSEMC_Dbi_Colum_5bit,
  kSEMC_Dbi_Colum_4bit,
  kSEMC_Dbi_Colum_3bit,
  kSEMC_Dbi_Colum_2bit }
  *SEMC dbi column address bit number.*
- enum sem_dbi_burst_len_t {
  kSEMC_Dbi_BurstLen1 = 0,
  kSEMC_Dbi_BurstLen2,
  kSEMC_Dbi_Dbi_BurstLen4,
  kSEMC_Dbi_BurstLen8,
  kSEMC_Dbi_BurstLen16,
  kSEMC_Dbi_BurstLen32,
  kSEMC_Dbi_BurstLen64 }
  *SEMC dbi burst length.*
- enum semc_iomux_pin {
  kSEMC_MUXA8 = SEMC_IOCR_MUX_A8_SHIFT,
  kSEMC_MUXCSX0 = SEMC_IOCR_MUX_CSX0_SHIFT,
  kSEMC_MUXCSX1 = SEMC_IOCR_MUX_CSX1_SHIFT,
  kSEMC_MUXCSX2 = SEMC_IOCR_MUX_CSX2_SHIFT,
  kSEMC_MUXCSX3 = SEMC_IOCR_MUX_CSX3_SHIFT,
  kSEMC_MUXRDY = SEMC_IOCR_MUX_RDY_SHIFT }
  *SEMC IOMUXC.*
- enum semc_iomux_nora27_pin {
  kSEMC_MORA27_NONE = 0,
  kSEMC_NORA27_MUXCSX3 = SEMC_IOCR_MUX_CSX3_SHIFT,
  kSEMC_NORA27_MUXRDY = SEMC_IOCR_MUX_RDY_SHIFT }
  *SEMC NOR/PSRAM Address bit 27 A27.*
- enum smec_port_size_t {
  kSEMC_PortSize8Bit = 0,
  kSEMC_PortSize16Bit }
  *SEMC port size.*
- enum semc_addr_mode_t {
  kSEMC_AddrDataMux = 0,
  kSEMC_AdvAddrdataMux,
  kSEMC_AddrDataNonMux }
  *SEMC address mode.*

- enum semc_dqs_mode_t {
  kSEMC_Loopbackinternal = 0,
  kSEMC_Loopbackdqspad }
  *SEMC DQS read strobe mode.*
- enum semc_adv_polarity_t {
  kSEMC_AdvActiveLow = 0,
  kSEMC_AdvActivehigh }
  *SEMC ADV signal active polarity.*
- enum semc_rdy_polarity_t {
  kSEMC_RdyActiveLow = 0,
  kSEMC_RdyActivehigh }
  *SEMC RDY signal active polarity.*
- enum semc_ipcmd_nand_addrmode_t {
  kSEMC_NANDAM_ColumnRow = 0x0U,
  kSEMC_NANDAM_ColumnCA0,
  kSEMC_NANDAM_ColumnCA0CA1,
  kSEMC_NANDAM_RawRA0,
  kSEMC_NANDAM_RawRA0RA1,
  kSEMC_NANDAM_RawRA0RA1RA2 }
  *SEMC IP command for NAND: address mode.*
- enum semc_ipcmd_nand_cmdmode_t {
  kSEMC_NANDCM_Command = 0x2U,
  kSEMC_NANDCM_CommandHold,
  kSEMC_NANDCM_CommandAddress,
  kSEMC_NANDCM_CommandAddressHold,
  kSEMC_NANDCM_CommandAddressRead,
  kSEMC_NANDCM_CommandAddressWrite,
  kSEMC_NANDCM_CommandRead,
  kSEMC_NANDCM_CommandWrite,
  kSEMC_NANDCM_Read,
  kSEMC_NANDCM_Write }
  *SEMC IP command for NAND command mode.*
- enum semc_nand_address_option_t {
  kSEMC_NandAddrOption_5byte_CA2RA3 = 0U,
  kSEMC_NandAddrOption_4byte_CA2RA2 = 2U,
  kSEMC_NandAddrOption_3byte_CA2RA1 = 4U,
  kSEMC_NandAddrOption_4byte_CA1RA3 = 1U,
  kSEMC_NandAddrOption_3byte_CA1RA2 = 3U,
  kSEMC_NandAddrOption_2byte_CA1RA1 = 7U }
  *SEMC NAND address option.*
- enum semc_ipcmd_nor_dbi_t {
  kSEMC_NORDBICM_Read = 0x2U,
  kSEMC_NORDBICM_Write }
  *SEMC IP command for NOR.*
- enum semc_ipcmd_sram_t {

kSEMC_SRAMCM_ArrayRead = 0x2U,
kSEMC_SRAMCM_ArrayWrite,
kSEMC_SRAMCM_RegRead,
kSEMC_SRAMCM_RegWrite }
*SEMC IP command for SRAM.*

- enum semc_ipcmd_sdram_t {
  kSEMC_SDRAMCM_Read = 0x8U,
  kSEMC_SDRAMCM_Write,
  kSEMC_SDRAMCM_Modeset,
  kSEMC_SDRAMCM_Active,
  kSEMC_SDRAMCM_AutoRefresh,
  kSEMC_SDRAMCM_SelfRefresh,
  kSEMC_SDRAMCM_Precharge,
  kSEMC_SDRAMCM_Prechargeall }
  *SEMC IP command for SDARM.*

## Driver version

- #define FSL_SEMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  *SEMC driver version 2.0.4.*

## SEMC Initialization and De-initialization

- void SEMC_GetDefaultConfig (semc_config_t *config)
  *Gets the SEMC default basic configuration structure.*
- void SEMC_Init (SEMC_Type *base, semc_config_t *configure)
  *Initializes SEMC.*
- void SEMC_Deinit (SEMC_Type *base)
  *Deinitializes the SEMC module and gates the clock.*

## SEMC Configuration Operation For Each Memory Type

- status_t SEMC_ConfigureSDRAM (SEMC_Type *base, semc_sdram_cs_t cs, semc_sdram_config_t *config, uint32_t clkSrc_Hz)
  *Configures SDRAM controller in SEMC.*
- status_t SEMC_ConfigureNAND (SEMC_Type *base, semc_nand_config_t *config, uint32_t clkSrc_Hz)
  *Configures NAND controller in SEMC.*
- status_t SEMC_ConfigureNOR (SEMC_Type *base, semc_nor_config_t *config, uint32_t clkSrc_Hz)
  *Configures NOR controller in SEMC.*
- status_t SEMC_ConfigureSRAM (SEMC_Type *base, semc_sram_config_t *config, uint32_t clkSrc_Hz)
  *Configures SRAM controller in SEMC.*
- status_t SEMC_ConfigureDBI (SEMC_Type *base, semc_dbi_config_t *config, uint32_t clkSrc_Hz)
  *Configures DBI controller in SEMC.*

## SEMC Interrupt Operation

- static void SEMC_EnableInterrupts (SEMC_Type ∗base, uint32_t mask)
  *Enables the SEMC interrupt.*
- static void SEMC_DisableInterrupts (SEMC_Type ∗base, uint32_t mask)
  *Disables the SEMC interrupt.*
- static bool SEMC_GetStatusFlag (SEMC_Type ∗base)
  *Gets the SEMC status.*
- static void SEMC_ClearStatusFlags (SEMC_Type ∗base, uint32_t mask)
  *Clears the SEMC status flag state.*

## SEMC Memory Access Operation

- static bool SEMC_IsInIdle (SEMC_Type ∗base)
  *Check if SEMC is in idle.*
- status_t SEMC_SendIPCommand (SEMC_Type ∗base, semc_mem_type_t type, uint32_t address, uint16_t command, uint32_t write, uint32_t ∗read)
  *SEMC IP command access.*
- static uint16_t SEMC_BuildNandIPCommand (uint8_t userCommand, semc_ipcmd_nand_addrmode_t addrMode, semc_ipcmd_nand_cmdmode_t cmdMode)
  *Build SEMC IP command for NAND.*
- static bool SEMC_IsNandReady (SEMC_Type ∗base)
  *Check if the NAND device is ready.*
- status_t SEMC_IPCommandNandWrite (SEMC_Type ∗base, uint32_t address, uint8_t ∗data, uint32_t size_bytes)
  *SEMC NAND device memory write through IP command.*
- status_t SEMC_IPCommandNandRead (SEMC_Type ∗base, uint32_t address, uint8_t ∗data, uint32_t size_bytes)
  *SEMC NAND device memory read through IP command.*
- status_t SEMC_IPCommandNorWrite (SEMC_Type ∗base, uint32_t address, uint8_t ∗data, uint32_t size_bytes)
  *SEMC NOR device memory write through IP command.*
- status_t SEMC_IPCommandNorRead (SEMC_Type ∗base, uint32_t address, uint8_t ∗data, uint32_t size_bytes)
  *SEMC NOR device memory read through IP command.*

# 39.3 Data Structure Documentation

## 39.3.1 struct semc_sdram_config_t

1. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2, 2^3, 2^4 .etc which is base 2KB exponential function. Take refer to BR0∼BR3 register in RM for details.
2. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer_n16cycle = 0, it means the prescaler timer period is 256 ∗ 16 clock cycles. For precalerIf precalerTimer_n16cycle not equal to 0, The prescaler timer period is prescalePeriod_N16Cycle ∗ 16 clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

## Data Fields

- semc_iomux_pin csxPinMux
  *CS pin mux.*
- uint32_t address
  *The base address.*
- uint32_t memsize_kbytes
  *The memory size in unit of kbytes.*
- smec_port_size_t portSize
  *Port size.*
- sem_sdram_burst_len_t burstLen
  *Burst length.*
- semc_sdram_column_bit_num_t columnAddrBitNum
  *Column address bit number.*
- semc_caslatency_t casLatency
  *CAS latency.*
- uint8_t tPrecharge2Act_Ns
  *Precharge to active wait time in unit of nanosecond.*
- uint8_t tAct2ReadWrite_Ns
  *Act to read/write wait time in unit of nanosecond.*
- uint8_t tRefreshRecovery_Ns
  *Refresh recovery time in unit of nanosecond.*
- uint8_t tWriteRecovery_Ns
  *write recovery time in unit of nanosecond.*
- uint8_t tCkeOff_Ns
  *CKE off minimum time in unit of nanosecond.*
- uint8_t tAct2Prechage_Ns
  *Active to precharge in unit of nanosecond.*
- uint8_t tSelfRefRecovery_Ns
  *Self refresh recovery time in unit of nanosecond.*
- uint8_t tRefresh2Refresh_Ns
  *Refresh to refresh wait time in unit of nanosecond.*
- uint8_t tAct2Act_Ns
  *Active to active wait time in unit of nanosecond.*
- uint32_t tPrescalePeriod_Ns
  *Prescaler timer period should not be larger than 256 ∗ 16 ∗ clock cycle.*
- uint32_t tIdleTimeout_Ns
  *Idle timeout in unit of prescale time period.*
- uint32_t refreshPeriod_nsPerRow
  *Refresh timer period like 64ms ∗ 1000000/8192 .*
- uint32_t refreshUrgThreshold
  *Refresh urgent threshold.*
- uint8_t refreshBurstLen
  *Refresh burst length.*

#### 39.3.1.0.0.44 Field Documentation

##### 39.3.1.0.0.44.1 semc_iomux_pin semc_sdram_config_t::csxPinMux

The kSEMC_MUXA8 is not valid in sdram pin mux setting.

#### 39.3.1.0.0.44.2 uint32_t semc_sdram_config_t::address

#### 39.3.1.0.0.44.3 uint32_t semc_sdram_config_t::memsize_kbytes

#### 39.3.1.0.0.44.4 smec_port_size_t semc_sdram_config_t::portSize

#### 39.3.1.0.0.44.5 sem_sdram_burst_len_t semc_sdram_config_t::burstLen

#### 39.3.1.0.0.44.6 semc_sdram_column_bit_num_t semc_sdram_config_t::columnAddrBitNum

#### 39.3.1.0.0.44.7 semc_caslatency_t semc_sdram_config_t::casLatency

#### 39.3.1.0.0.44.8 uint8_t semc_sdram_config_t::tPrecharge2Act_Ns

#### 39.3.1.0.0.44.9 uint8_t semc_sdram_config_t::tAct2ReadWrite_Ns

#### 39.3.1.0.0.44.10 uint8_t semc_sdram_config_t::tRefreshRecovery_Ns

#### 39.3.1.0.0.44.11 uint8_t semc_sdram_config_t::tWriteRecovery_Ns

#### 39.3.1.0.0.44.12 uint8_t semc_sdram_config_t::tCkeOff_Ns

#### 39.3.1.0.0.44.13 uint8_t semc_sdram_config_t::tAct2Prechage_Ns

#### 39.3.1.0.0.44.14 uint8_t semc_sdram_config_t::tSelfRefRecovery_Ns

#### 39.3.1.0.0.44.15 uint8_t semc_sdram_config_t::tRefresh2Refresh_Ns

#### 39.3.1.0.0.44.16 uint8_t semc_sdram_config_t::tAct2Act_Ns

#### 39.3.1.0.0.44.17 uint32_t semc_sdram_config_t::tPrescalePeriod_Ns

#### 39.3.1.0.0.44.18 uint32_t semc_sdram_config_t::tIdleTimeout_Ns

#### 39.3.1.0.0.44.19 uint32_t semc_sdram_config_t::refreshPeriod_nsPerRow

#### 39.3.1.0.0.44.20 uint32_t semc_sdram_config_t::refreshUrgThreshold

#### 39.3.1.0.0.44.21 uint8_t semc_sdram_config_t::refreshBurstLen

### 39.3.2 struct semc_nand_timing_config_t

## Data Fields

- uint8_t tCeSetup_Ns
  *CE setup time: tCS.*
- uint8_t tCeHold_Ns
  *CE hold time: tCH.*
- uint8_t tCeInterval_Ns
  *CE interval time:tCEITV.*

- uint8_t tWeLow_Ns
  *WE low time: tWP.*
- uint8_t tWeHigh_Ns
  *WE high time: tWH.*
- uint8_t tReLow_Ns
  *RE low time: tRP.*
- uint8_t tReHigh_Ns
  *RE high time: tREH.*
- uint8_t tTurnAround_Ns
  *Turnaround time for async mode: tTA.*
- uint8_t tWehigh2Relow_Ns
  *WE# high to RE# wait time: tWHR.*
- uint8_t tRehigh2Welow_Ns
  *RE# high to WE# low wait time: tRHW.*
- uint8_t tAle2WriteStart_Ns
  *ALE to write start wait time: tADL.*
- uint8_t tReady2Relow_Ns
  *Ready to RE# low min wait time: tRR.*
- uint8_t tWehigh2Busy_Ns
  *WE# high to busy wait time: tWB.*

#### 39.3.2.0.0.45 Field Documentation

##### 39.3.2.0.0.45.1 uint8_t semc_nand_timing_config_t::tCeSetup_Ns

##### 39.3.2.0.0.45.2 uint8_t semc_nand_timing_config_t::tCeHold_Ns

##### 39.3.2.0.0.45.3 uint8_t semc_nand_timing_config_t::tCeInterval_Ns

##### 39.3.2.0.0.45.4 uint8_t semc_nand_timing_config_t::tWeLow_Ns

##### 39.3.2.0.0.45.5 uint8_t semc_nand_timing_config_t::tWeHigh_Ns

##### 39.3.2.0.0.45.6 uint8_t semc_nand_timing_config_t::tReLow_Ns

##### 39.3.2.0.0.45.7 uint8_t semc_nand_timing_config_t::tReHigh_Ns

##### 39.3.2.0.0.45.8 uint8_t semc_nand_timing_config_t::tTurnAround_Ns

##### 39.3.2.0.0.45.9 uint8_t semc_nand_timing_config_t::tWehigh2Relow_Ns

##### 39.3.2.0.0.45.10 uint8_t semc_nand_timing_config_t::tRehigh2Welow_Ns

##### 39.3.2.0.0.45.11 uint8_t semc_nand_timing_config_t::tAle2WriteStart_Ns

##### 39.3.2.0.0.45.12 uint8_t semc_nand_timing_config_t::tReady2Relow_Ns

##### 39.3.2.0.0.45.13 uint8_t semc_nand_timing_config_t::tWehigh2Busy_Ns

### 39.3.3 struct semc_nand_config_t

## Data Fields

- semc_iomux_pin cePinMux
  *The CE pin mux setting.*
- uint32_t axiAddress
  *The base address for AXI nand.*
- uint32_t axiMemsize_kbytes
  *The memory size in unit of kbytes for AXI nand.*
- uint32_t ipgAddress
  *The base address for IPG nand .*
- uint32_t ipgMemsize_kbytes
  *The memory size in unit of kbytes for IPG nand.*
- semc_rdy_polarity_t rdyactivePolarity
  *Wait ready polarity.*
- bool edoModeEnabled
  *EDO mode enabled.*
- semc_nand_column_bit_num_t columnAddrBitNum
  *Column address bit number.*
- semc_nand_address_option_t arrayAddrOption
  *Address option.*

- sem_nand_burst_len_t burstLen
  *Burst length.*
- smec_port_size_t portSize
  *Port size.*
- semc_nand_timing_config_t * timingConfig
  *SEMC nand timing configuration.*

#### 39.3.3.0.0.46 Field Documentation

##### 39.3.3.0.0.46.1 semc_iomux_pin semc_nand_config_t::cePinMux

The kSEMC_MUXRDY is not valid for CE pin setting.

##### 39.3.3.0.0.46.2 uint32_t semc_nand_config_t::axiAddress

##### 39.3.3.0.0.46.3 uint32_t semc_nand_config_t::axiMemsize_kbytes

##### 39.3.3.0.0.46.4 uint32_t semc_nand_config_t::ipgAddress

##### 39.3.3.0.0.46.5 uint32_t semc_nand_config_t::ipgMemsize_kbytes

##### 39.3.3.0.0.46.6 semc_rdy_polarity_t semc_nand_config_t::rdyactivePolarity

##### 39.3.3.0.0.46.7 bool semc_nand_config_t::edoModeEnabled

##### 39.3.3.0.0.46.8 semc_nand_column_bit_num_t semc_nand_config_t::columnAddrBitNum

##### 39.3.3.0.0.46.9 semc_nand_address_option_t semc_nand_config_t::arrayAddrOption

##### 39.3.3.0.0.46.10 sem_nand_burst_len_t semc_nand_config_t::burstLen

##### 39.3.3.0.0.46.11 smec_port_size_t semc_nand_config_t::portSize

##### 39.3.3.0.0.46.12 semc_nand_timing_config_t* semc_nand_config_t::timingConfig

### 39.3.4 struct semc_nor_config_t

## Data Fields

- semc_iomux_pin cePinMux
  *The CE# pin mux setting.*
- semc_iomux_nora27_pin addr27
  *The Addr bit 27 pin mux setting.*
- uint32_t address
  *The base address.*
- uint32_t memsize_kbytes
  *The memory size in unit of kbytes.*
- uint8_t addrPortWidth
  *The address port width.*
- semc_rdy_polarity_t rdyactivePolarity

*Wait ready polarity.*
- semc_adv_polarity_t advActivePolarity
  *ADV# polarity.*
- semc_norsram_column_bit_num_t columnAddrBitNum
  *Column address bit number.*
- semc_addr_mode_t addrMode
  *Address mode.*
- sem_norsram_burst_len_t burstLen
  *Burst length.*
- smec_port_size_t portSize
  *Port size.*
- uint8_t tCeSetup_Ns
  *The CE setup time.*
- uint8_t tCeHold_Ns
  *The CE hold time.*
- uint8_t tCeInterval_Ns
  *CE interval minimum time.*
- uint8_t tAddrSetup_Ns
  *The address setup time.*
- uint8_t tAddrHold_Ns
  *The address hold time.*
- uint8_t tWeLow_Ns
  *WE low time for async mode.*
- uint8_t tWeHigh_Ns
  *WE high time for async mode.*
- uint8_t tReLow_Ns
  *RE low time for async mode.*
- uint8_t tReHigh_Ns
  *RE high time for async mode.*
- uint8_t tTurnAround_Ns
  *Turnaround time for async mode.*
- uint8_t tAddr2WriteHold_Ns
  *Address to write data hold time for async mode.*
- uint8_t latencyCount
  *Latency count for sync mode.*
- uint8_t readCycle
  *Read cycle time for sync mode.*

#### 39.3.4.0.0.47 Field Documentation

##### 39.3.4.0.0.47.1 semc_iomux_pin semc_nor_config_t::cePinMux

##### 39.3.4.0.0.47.2 semc_iomux_nora27_pin semc_nor_config_t::addr27

##### 39.3.4.0.0.47.3 uint32_t semc_nor_config_t::address

##### 39.3.4.0.0.47.4 uint32_t semc_nor_config_t::memsize_kbytes

##### 39.3.4.0.0.47.5 uint8_t semc_nor_config_t::addrPortWidth

##### 39.3.4.0.0.47.6 semc_rdy_polarity_t semc_nor_config_t::rdyactivePolarity

##### 39.3.4.0.0.47.7 semc_adv_polarity_t semc_nor_config_t::advActivePolarity

##### 39.3.4.0.0.47.8 semc_norsram_column_bit_num_t semc_nor_config_t::columnAddrBitNum

##### 39.3.4.0.0.47.9 semc_addr_mode_t semc_nor_config_t::addrMode

##### 39.3.4.0.0.47.10 sem_norsram_burst_len_t semc_nor_config_t::burstLen

##### 39.3.4.0.0.47.11 smec_port_size_t semc_nor_config_t::portSize

##### 39.3.4.0.0.47.12 uint8_t semc_nor_config_t::tCeSetup_Ns

##### 39.3.4.0.0.47.13 uint8_t semc_nor_config_t::tCeHold_Ns

##### 39.3.4.0.0.47.14 uint8_t semc_nor_config_t::tCeInterval_Ns

##### 39.3.4.0.0.47.15 uint8_t semc_nor_config_t::tAddrSetup_Ns

##### 39.3.4.0.0.47.16 uint8_t semc_nor_config_t::tAddrHold_Ns

##### 39.3.4.0.0.47.17 uint8_t semc_nor_config_t::tWeLow_Ns

##### 39.3.4.0.0.47.18 uint8_t semc_nor_config_t::tWeHigh_Ns

##### 39.3.4.0.0.47.19 uint8_t semc_nor_config_t::tReLow_Ns

##### 39.3.4.0.0.47.20 uint8_t semc_nor_config_t::tReHigh_Ns

##### 39.3.4.0.0.47.21 uint8_t semc_nor_config_t::tTurnAround_Ns

##### 39.3.4.0.0.47.22 uint8_t semc_nor_config_t::tAddr2WriteHold_Ns

##### 39.3.4.0.0.47.23 uint8_t semc_nor_config_t::latencyCount

##### 39.3.4.0.0.47.24 uint8_t semc_nor_config_t::readCycle

### 39.3.5 struct semc_sram_config_t

**Data Fields**

- semc_iomux_pin cePinMux

*The CE# pin mux setting.*

- semc_iomux_nora27_pin addr27
  *The Addr bit 27 pin mux setting.*
- uint32_t address
  *The base address.*
- uint32_t memsize_kbytes
  *The memory size in unit of kbytes.*
- uint8_t addrPortWidth
  *The address port width.*
- semc_adv_polarity_t advActivePolarity
  *ADV# polarity 1: active high, 0: active low.*
- semc_addr_mode_t addrMode
  *Address mode.*
- sem_norsram_burst_len_t burstLen
  *Burst length.*
- smec_port_size_t portSize
  *Port size.*
- uint8_t tCeSetup_Ns
  *The CE setup time.*
- uint8_t tCeHold_Ns
  *The CE hold time.*
- uint8_t tCeInterval_Ns
  *CE interval minimum time.*
- uint8_t tAddrSetup_Ns
  *The address setup time.*
- uint8_t tAddrHold_Ns
  *The address hold time.*
- uint8_t tWeLow_Ns
  *WE low time for async mode.*
- uint8_t tWeHigh_Ns
  *WE high time for async mode.*
- uint8_t tReLow_Ns
  *RE low time for async mode.*
- uint8_t tReHigh_Ns
  *RE high time for async mode.*
- uint8_t tTurnAround_Ns
  *Turnaround time for async mode.*
- uint8_t tAddr2WriteHold_Ns
  *Address to write data hold time for async mode.*
- uint8_t tWriteSetup_Ns
  *Write data setup time for sync mode.*
- uint8_t tWriteHold_Ns
  *Write hold time for sync mode.*
- uint8_t latencyCount
  *Latency count for sync mode.*
- uint8_t readCycle
  *Read cycle time for sync mode.*

#### 39.3.5.0.0.48 Field Documentation

##### 39.3.5.0.0.48.1 semc_iomux_pin semc_sram_config_t::cePinMux

##### 39.3.5.0.0.48.2 semc_iomux_nora27_pin semc_sram_config_t::addr27

##### 39.3.5.0.0.48.3 uint32_t semc_sram_config_t::address

##### 39.3.5.0.0.48.4 uint32_t semc_sram_config_t::memsize_kbytes

##### 39.3.5.0.0.48.5 uint8_t semc_sram_config_t::addrPortWidth

##### 39.3.5.0.0.48.6 semc_adv_polarity_t semc_sram_config_t::advActivePolarity

##### 39.3.5.0.0.48.7 semc_addr_mode_t semc_sram_config_t::addrMode

##### 39.3.5.0.0.48.8 sem_norsram_burst_len_t semc_sram_config_t::burstLen

##### 39.3.5.0.0.48.9 smec_port_size_t semc_sram_config_t::portSize

##### 39.3.5.0.0.48.10 uint8_t semc_sram_config_t::tCeSetup_Ns

##### 39.3.5.0.0.48.11 uint8_t semc_sram_config_t::tCeHold_Ns

##### 39.3.5.0.0.48.12 uint8_t semc_sram_config_t::tCeInterval_Ns

##### 39.3.5.0.0.48.13 uint8_t semc_sram_config_t::tAddrSetup_Ns

##### 39.3.5.0.0.48.14 uint8_t semc_sram_config_t::tAddrHold_Ns

##### 39.3.5.0.0.48.15 uint8_t semc_sram_config_t::tWeLow_Ns

##### 39.3.5.0.0.48.16 uint8_t semc_sram_config_t::tWeHigh_Ns

##### 39.3.5.0.0.48.17 uint8_t semc_sram_config_t::tReLow_Ns

##### 39.3.5.0.0.48.18 uint8_t semc_sram_config_t::tReHigh_Ns

##### 39.3.5.0.0.48.19 uint8_t semc_sram_config_t::tTurnAround_Ns

##### 39.3.5.0.0.48.20 uint8_t semc_sram_config_t::tAddr2WriteHold_Ns

##### 39.3.5.0.0.48.21 uint8_t semc_sram_config_t::tWriteSetup_Ns

##### 39.3.5.0.0.48.22 uint8_t semc_sram_config_t::tWriteHold_Ns

##### 39.3.5.0.0.48.23 uint8_t semc_sram_config_t::latencyCount

##### 39.3.5.0.0.48.24 uint8_t semc_sram_config_t::readCycle

### 39.3.6 struct semc_dbi_config_t

**Data Fields**



- semc_iomux_pin csxPinMux

*The CE# pin mux.*
- uint32_t address
  *The base address.*
- uint32_t memsize_kbytes
  *The memory size in unit of 4kbytes.*
- semc_dbi_column_bit_num_t columnAddrBitNum
  *Column address bit number.*
- sem_dbi_burst_len_t burstLen
  *Burst length.*
- smec_port_size_t portSize
  *Port size.*
- uint8_t tCsxSetup_Ns
  *The CSX setup time.*
- uint8_t tCsxHold_Ns
  *The CSX hold time.*
- uint8_t tWexLow_Ns
  *WEX low time.*
- uint8_t tWexHigh_Ns
  *WEX high time.*
- uint8_t tRdxLow_Ns
  *RDX low time.*
- uint8_t tRdxHigh_Ns
  *RDX high time.*
- uint8_t tCsxInterval_Ns
  *Write data setup time.*

#### 39.3.6.0.0.49 Field Documentation

#### 39.3.6.0.0.49.1 semc_iomux_pin semc_dbi_config_t::csxPinMux

#### 39.3.6.0.0.49.2 uint32_t semc_dbi_config_t::address

#### 39.3.6.0.0.49.3 uint32_t semc_dbi_config_t::memsize_kbytes

#### 39.3.6.0.0.49.4 semc_dbi_column_bit_num_t semc_dbi_config_t::columnAddrBitNum

#### 39.3.6.0.0.49.5 sem_dbi_burst_len_t semc_dbi_config_t::burstLen

#### 39.3.6.0.0.49.6 smec_port_size_t semc_dbi_config_t::portSize

#### 39.3.6.0.0.49.7 uint8_t semc_dbi_config_t::tCsxSetup_Ns

#### 39.3.6.0.0.49.8 uint8_t semc_dbi_config_t::tCsxHold_Ns

#### 39.3.6.0.0.49.9 uint8_t semc_dbi_config_t::tWexLow_Ns

#### 39.3.6.0.0.49.10 uint8_t semc_dbi_config_t::tWexHigh_Ns

#### 39.3.6.0.0.49.11 uint8_t semc_dbi_config_t::tRdxLow_Ns

#### 39.3.6.0.0.49.12 uint8_t semc_dbi_config_t::tRdxHigh_Ns

#### 39.3.6.0.0.49.13 uint8_t semc_dbi_config_t::tCsxInterval_Ns

### 39.3.7 struct semc_queuea_weight_struct_t

## Data Fields

- uint32_t qos: 4
  *weight of qos for queue 0 .*
- uint32_t aging: 4
  *weight of aging for queue 0.*
- uint32_t slaveHitSwith: 8
  *weight of read/write switch for queue 0.*
- uint32_t slaveHitNoswitch: 8
  *weight of read/write no switch for queue 0 .*

#### 39.3.7.0.0.50 Field Documentation

##### 39.3.7.0.0.50.1 uint32_t semc_queuea_weight_struct_t::qos

##### 39.3.7.0.0.50.2 uint32_t semc_queuea_weight_struct_t::aging

##### 39.3.7.0.0.50.3 uint32_t semc_queuea_weight_struct_t::slaveHitSwith

##### 39.3.7.0.0.50.4 uint32_t semc_queuea_weight_struct_t::slaveHitNoswitch

### 39.3.8 union semc_queuea_weight_t

## Data Fields

- semc_queuea_weight_struct_t queueaConfig
  *Structure configuration for queueA.*
- uint32_t queueaValue
  *Configuration value for queueA which could directly write to the reg.*

#### 39.3.8.0.0.51 Field Documentation

##### 39.3.8.0.0.51.1 semc_queuea_weight_struct_t semc_queuea_weight_t::queueaConfig

##### 39.3.8.0.0.51.2 uint32_t semc_queuea_weight_t::queueaValue

### 39.3.9 struct semc_queueb_weight_struct_t

## Data Fields

- uint32_t qos: 4
  *weight of qos for queue 1.*
- uint32_t aging: 4
  *weight of aging for queue 1.*
- uint32_t slaveHitSwith: 8
  *weight of read/write switch for queue 1.*
- uint32_t weightPagehit: 8
  *weight of page hit for queue 1 only .*
- uint32_t bankRotation: 8
  *weight of bank rotation for queue 1 only .*

#### 39.3.9.0.0.52 Field Documentation

##### 39.3.9.0.0.52.1 uint32_t semc_queueb_weight_struct_t::qos

##### 39.3.9.0.0.52.2 uint32_t semc_queueb_weight_struct_t::aging

##### 39.3.9.0.0.52.3 uint32_t semc_queueb_weight_struct_t::slaveHitSwith

##### 39.3.9.0.0.52.4 uint32_t semc_queueb_weight_struct_t::weightPagehit

##### 39.3.9.0.0.52.5 uint32_t semc_queueb_weight_struct_t::bankRotation

### 39.3.10 union semc_queueb_weight_t

## Data Fields

- semc_queueb_weight_struct_t queuebConfig
  *Structure configuration for queueB.*
- uint32_t queuebValue
  *Configuration value for queueB which could directly write to the reg.*

#### 39.3.10.0.0.53 Field Documentation

##### 39.3.10.0.0.53.1 semc_queueb_weight_struct_t semc_queueb_weight_t::queuebConfig

##### 39.3.10.0.0.53.2 uint32_t semc_queueb_weight_t::queuebValue

### 39.3.11 struct semc_axi_queueweight_t

## Data Fields

- semc_queuea_weight_t queueaWeight
  *Weight settings for queue a.*
- semc_queueb_weight_t queuebWeight
  *Weight settings for queue b.*

#### 39.3.11.0.0.54 Field Documentation

##### 39.3.11.0.0.54.1 semc_queuea_weight_t semc_axi_queueweight_t::queueaWeight

##### 39.3.11.0.0.54.2 semc_queueb_weight_t semc_axi_queueweight_t::queuebWeight

### 39.3.12 struct semc_config_t

busTimeoutCycles: when busTimeoutCycles is zero, the bus timeout cycle is 255∗1024. otherwise the bus timeout cycles is busTimeoutCycles∗1024. cmdTimeoutCycles: is used for command execution timeout cycles. it's similar to the busTimeoutCycles.

### Data Fields

- semc_dqs_mode_t dqsMode
  *Dummy read strobe mode: use enum in "semc_dqs_mode_t".*
- uint8_t cmdTimeoutCycles
  *Command execution timeout cycles.*
- uint8_t busTimeoutCycles
  *Bus timeout cycles.*
- semc_axi_queueweight_t queueWeight
  *AXI queue weight.*

#### 39.3.12.0.0.55 Field Documentation

#### 39.3.12.0.0.55.1 semc_dqs_mode_t semc_config_t::dqsMode

#### 39.3.12.0.0.55.2 uint8_t semc_config_t::cmdTimeoutCycles

#### 39.3.12.0.0.55.3 uint8_t semc_config_t::busTimeoutCycles

#### 39.3.12.0.0.55.4 semc_axi_queueweight_t semc_config_t::queueWeight

## 39.4 Macro Definition Documentation

### 39.4.1 #define FSL_SEMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

## 39.5 Enumeration Type Documentation

### 39.5.1 enum _semc_status

### 39.5.2 enum semc_mem_type_t

Enumerator

***kSEMC_MemType_SDRAM*** SDRAM.
***kSEMC_MemType_SRAM*** SRAM.
***kSEMC_MemType_NOR*** NOR.
***kSEMC_MemType_NAND*** NAND.
***kSEMC_MemType_8080*** 1.

### 39.5.3 enum semc_waitready_polarity_t

Enumerator

***kSEMC_LowActive*** Low active.
***kSEMC_HighActive*** High active.

### 39.5.4 enum semc_sdram_cs_t

Enumerator

***kSEMC_SDRAM_CS0*** SEMC SDRAM CS0.
***kSEMC_SDRAM_CS1*** SEMC SDRAM CS1.
***kSEMC_SDRAM_CS2*** SEMC SDRAM CS2.
***kSEMC_SDRAM_CS3*** SEMC SDRAM CS3.

### 39.5.5 enum semc_nand_access_type_t

### 39.5.6 enum semc_interrupt_enable_t

Enumerator

***kSEMC_IPCmdDoneInterrupt*** Ip command done interrupt.
***kSEMC_IPCmdErrInterrupt*** Ip command error interrupt.
***kSEMC_AXICmdErrInterrupt*** AXI command error interrupt.
***kSEMC_AXIBusErrInterrupt*** AXI bus error interrupt.

### 39.5.7 enum semc_ipcmd_datasize_t

Enumerator

***kSEMC_IPcmdDataSize_1bytes*** The IP command data size 1 byte.
***kSEMC_IPcmdDataSize_2bytes*** The IP command data size 2 byte.
***kSEMC_IPcmdDataSize_3bytes*** The IP command data size 3 byte.
***kSEMC_IPcmdDataSize_4bytes*** The IP command data size 4 byte.

### 39.5.8 enum semc_refresh_time_t

Enumerator

***kSEMC_RefreshThreeClocks*** The refresh timing with three bus clocks.
***kSEMC_RefreshSixClocks*** The refresh timing with six bus clocks.
***kSEMC_RefreshNineClocks*** The refresh timing with nine bus clocks.

### 39.5.9 enum semc_caslatency_t

Enumerator

***kSEMC_LatencyOne*** Latency 1.
***kSEMC_LatencyTwo*** Latency 2.
***kSEMC_LatencyThree*** Latency 3.

### 39.5.10 enum semc_sdram_column_bit_num_t

Enumerator

***kSEMC_SdramColunm_12bit*** 12 bit.
***kSEMC_SdramColunm_11bit*** 11 bit.
***kSEMC_SdramColunm_10bit*** 10 bit.
***kSEMC_SdramColunm_9bit*** 9 bit.

### 39.5.11 enum sem_sdram_burst_len_t

Enumerator

***kSEMC_Sdram_BurstLen1*** Burst length 1.
***kSEMC_Sdram_BurstLen2*** Burst length 2.
***kSEMC_Sdram_BurstLen4*** Burst length 4.
***kSEMC_Sdram_BurstLen8*** Burst length 8.

### 39.5.12 enum semc_nand_column_bit_num_t

Enumerator

***kSEMC_NandColum_16bit*** 16 bit.
***kSEMC_NandColum_15bit*** 15 bit.
***kSEMC_NandColum_14bit*** 14 bit.
***kSEMC_NandColum_13bit*** 13 bit.
***kSEMC_NandColum_12bit*** 12 bit.
***kSEMC_NandColum_11bit*** 11 bit.
***kSEMC_NandColum_10bit*** 10 bit.
***kSEMC_NandColum_9bit*** 9 bit.

### 39.5.13 enum sem_nand_burst_len_t

Enumerator

***kSEMC_Nand_BurstLen1*** Burst length 1.
***kSEMC_Nand_BurstLen2*** Burst length 2.
***kSEMC_Nand_BurstLen4*** Burst length 4.
***kSEMC_Nand_BurstLen8*** Burst length 8.
***kSEMC_Nand_BurstLen16*** Burst length 16.
***kSEMC_Nand_BurstLen32*** Burst length 32.
***kSEMC_Nand_BurstLen64*** Burst length 64.

### 39.5.14 enum semc_norsram_column_bit_num_t

Enumerator

***kSEMC_NorColum_12bit*** 12 bit.
***kSEMC_NorColum_11bit*** 11 bit.
***kSEMC_NorColum_10bit*** 10 bit.
***kSEMC_NorColum_9bit*** 9 bit.
***kSEMC_NorColum_8bit*** 8 bit.
***kSEMC_NorColum_7bit*** 7 bit.
***kSEMC_NorColum_6bit*** 6 bit.
***kSEMC_NorColum_5bit*** 5 bit.
***kSEMC_NorColum_4bit*** 4 bit.
***kSEMC_NorColum_3bit*** 3 bit.
***kSEMC_NorColum_2bit*** 2 bit.

### 39.5.15 enum sem_norsram_burst_len_t

Enumerator

***kSEMC_Nor_BurstLen1*** Burst length 1.
***kSEMC_Nor_BurstLen2*** Burst length 2.
***kSEMC_Nor_BurstLen4*** Burst length 4.
***kSEMC_Nor_BurstLen8*** Burst length 8.
***kSEMC_Nor_BurstLen16*** Burst length 16.
***kSEMC_Nor_BurstLen32*** Burst length 32.
***kSEMC_Nor_BurstLen64*** Burst length 64.

### 39.5.16 enum semc_dbi_column_bit_num_t

Enumerator

***kSEMC_Dbi_Colum_12bit*** 12 bit.
***kSEMC_Dbi_Colum_11bit*** 11 bit.
***kSEMC_Dbi_Colum_10bit*** 10 bit.
***kSEMC_Dbi_Colum_9bit*** 9 bit.
***kSEMC_Dbi_Colum_8bit*** 8 bit.
***kSEMC_Dbi_Colum_7bit*** 7 bit.
***kSEMC_Dbi_Colum_6bit*** 6 bit.
***kSEMC_Dbi_Colum_5bit*** 5 bit.
***kSEMC_Dbi_Colum_4bit*** 4 bit.
***kSEMC_Dbi_Colum_3bit*** 3 bit.
***kSEMC_Dbi_Colum_2bit*** 2 bit.

### 39.5.17 enum sem_dbi_burst_len_t

Enumerator

***kSEMC_Dbi_BurstLen1*** Burst length 1.
***kSEMC_Dbi_BurstLen2*** Burst length 2.
***kSEMC_Dbi_Dbi_BurstLen4*** Burst length 4.
***kSEMC_Dbi_BurstLen8*** Burst length 8.
***kSEMC_Dbi_BurstLen16*** Burst length 16.
***kSEMC_Dbi_BurstLen32*** Burst length 32.
***kSEMC_Dbi_BurstLen64*** Burst length 64.

### 39.5.18 enum semc_iomux_pin

Enumerator

***kSEMC_MUXA8*** MUX A8 pin.
***kSEMC_MUXCSX0*** MUX CSX0 pin.
***kSEMC_MUXCSX1*** MUX CSX1 Pin.
***kSEMC_MUXCSX2*** MUX CSX2 Pin.
***kSEMC_MUXCSX3*** MUX CSX3 Pin.
***kSEMC_MUXRDY*** MUX RDY pin.

### 39.5.19 enum semc_iomux_nora27_pin

Enumerator

***kSEMC_MORA27_NONE*** No NOR/SRAM A27 pin.
***kSEMC_NORA27_MUXCSX3*** MUX CSX3 Pin.
***kSEMC_NORA27_MUXRDY*** MUX RDY pin.

### 39.5.20 enum smec_port_size_t

Enumerator

***kSEMC_PortSize8Bit*** 8-Bit port size.
***kSEMC_PortSize16Bit*** 16-Bit port size.

### 39.5.21 enum semc_addr_mode_t

Enumerator

***kSEMC_AddrDataMux*** SEMC address/data mux mode.
***kSEMC_AdvAddrdataMux*** Advanced address/data mux mode.
***kSEMC_AddrDataNonMux*** Address/data non-mux mode.

### 39.5.22 enum semc_dqs_mode_t

Enumerator

***kSEMC_Loopbackinternal*** Dummy read strobe loopbacked internally.
***kSEMC_Loopbackdqspad*** Dummy read strobe loopbacked from DQS pad.

### 39.5.23 enum semc_adv_polarity_t

Enumerator

***kSEMC_AdvActiveLow*** Adv active low.
***kSEMC_AdvActivehigh*** Adv active low.

### 39.5.24 enum semc_rdy_polarity_t

Enumerator

***kSEMC_RdyActiveLow*** Adv active low.
***kSEMC_RdyActivehigh*** Adv active low.

### 39.5.25 enum semc_ipcmd_nand_addrmode_t

Enumerator

***kSEMC_NANDAM_ColumnRow*** Address mode: column and row address(5Byte-CA0/CA1/RA0/-RA1/RA2).
***kSEMC_NANDAM_ColumnCA0*** Address mode: column address only(1 Byte-CA0).
***kSEMC_NANDAM_ColumnCA0CA1*** Address mode: column address only(2 Byte-CA0/CA1).
***kSEMC_NANDAM_RawRA0*** Address mode: row address only(1 Byte-RA0).
***kSEMC_NANDAM_RawRA0RA1*** Address mode: row address only(2 Byte-RA0/RA1).
***kSEMC_NANDAM_RawRA0RA1RA2*** Address mode: row address only(3 Byte-RA0).

### 39.5.26 enum semc_ipcmd_nand_cmdmode_t

Enumerator

***kSEMC_NANDCM_Command*** command.
***kSEMC_NANDCM_CommandHold*** Command hold.
***kSEMC_NANDCM_CommandAddress*** Command address.
***kSEMC_NANDCM_CommandAddressHold*** Command address hold.
***kSEMC_NANDCM_CommandAddressRead*** Command address read.
***kSEMC_NANDCM_CommandAddressWrite*** Command address write.
***kSEMC_NANDCM_CommandRead*** Command read.
***kSEMC_NANDCM_CommandWrite*** Command write.
***kSEMC_NANDCM_Read*** Read.
***kSEMC_NANDCM_Write*** Write.

### 39.5.27 enum semc_nand_address_option_t

Enumerator

***kSEMC_NandAddrOption_5byte_CA2RA3*** CA0+CA1+RA0+RA1+RA2.
***kSEMC_NandAddrOption_4byte_CA2RA2*** CA0+CA1+RA0+RA1.
***kSEMC_NandAddrOption_3byte_CA2RA1*** CA0+CA1+RA0.
***kSEMC_NandAddrOption_4byte_CA1RA3*** CA0+RA0+RA1+RA2.

*kSEMC_NandAddrOption_3byte_CA1RA2* CA0+RA0+RA1.
*kSEMC_NandAddrOption_2byte_CA1RA1* CA0+RA0.

### 39.5.28 enum semc_ipcmd_nor_dbi_t

Enumerator

*kSEMC_NORDBICM_Read* NOR read.
*kSEMC_NORDBICM_Write* NOR write.

### 39.5.29 enum semc_ipcmd_sram_t

Enumerator

*kSEMC_SRAMCM_ArrayRead* SRAM memory array read.
*kSEMC_SRAMCM_ArrayWrite* SRAM memory array write.
*kSEMC_SRAMCM_RegRead* SRAM memory register read.
*kSEMC_SRAMCM_RegWrite* SRAM memory register write.

### 39.5.30 enum semc_ipcmd_sdram_t

Enumerator

*kSEMC_SDRAMCM_Read* SDRAM memory read.
*kSEMC_SDRAMCM_Write* SDRAM memory write.
*kSEMC_SDRAMCM_Modeset* SDRAM MODE SET.
*kSEMC_SDRAMCM_Active* SDRAM active.
*kSEMC_SDRAMCM_AutoRefresh* SDRAM auto-refresh.
*kSEMC_SDRAMCM_SelfRefresh* SDRAM self-refresh.
*kSEMC_SDRAMCM_Precharge* SDRAM precharge.
*kSEMC_SDRAMCM_Prechargeall* SDRAM precharge all.

## 39.6 Function Documentation

### 39.6.1 void SEMC_GetDefaultConfig ( semc_config_t * *config* )

The purpose of this API is to get the default SEMC configure structure for SEMC_Init(). User may use the initialized structure unchanged in SEMC_Init(), or modify some fields of the structure before calling SEMC_Init(). Example:

```
semc_config_t config;
SEMC_GetDefaultConfig(&config);
```

Parameters

| | |
|---|---|
| *config* | The SEMC configuration structure pointer. |

### 39.6.2 void SEMC_Init ( SEMC_Type * *base*, semc_config_t * *configure* )

This function ungates the SEMC clock and initializes SEMC. This function must be called before calling any other SEMC driver functions.

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *configure* | The SEMC configuration structure pointer. |

### 39.6.3 void SEMC_Deinit ( SEMC_Type * *base* )

This function gates the SEMC clock. As a result, the SEMC module doesn't work after calling this function, for some IDE, calling this API may cause the next downloading operation failed. so, please call this API cautiously. Additional, users can using "#define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (1)" to disable the clock control operation in drivers.

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |

### 39.6.4 status_t SEMC_ConfigureSDRAM ( SEMC_Type * *base*, semc_sdram_cs_t *cs*, semc_sdram_config_t * *config*, uint32_t *clkSrc_Hz* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *cs* | The chip selection. |
| *config* | The sdram configuration. |

| | |
|---|---|
| *clkSrc_Hz* | The SEMC clock frequency. |

### 39.6.5 status_t SEMC_ConfigureNAND ( SEMC_Type * *base*, semc_nand_config_t * *config*, uint32_t *clkSrc_Hz* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *config* | The nand configuration. |
| *clkSrc_Hz* | The SEMC clock frequency. |

### 39.6.6 status_t SEMC_ConfigureNOR ( SEMC_Type * *base*, semc_nor_config_t * *config*, uint32_t *clkSrc_Hz* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *config* | The nor configuration. |
| *clkSrc_Hz* | The SEMC clock frequency. |

### 39.6.7 status_t SEMC_ConfigureSRAM ( SEMC_Type * *base*, semc_sram_config_t * *config*, uint32_t *clkSrc_Hz* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *config* | The sram configuration. |
| *clkSrc_Hz* | The SEMC clock frequency. |

### 39.6.8 status_t SEMC_ConfigureDBI ( SEMC_Type * *base*, semc_dbi_config_t * *config*, uint32_t *clkSrc_Hz* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *config* | The dbi configuration. |
| *clkSrc_Hz* | The SEMC clock frequency. |

### 39.6.9 static void SEMC_EnableInterrupts ( SEMC_Type * *base*, uint32_t *mask* ) [inline], [static]

This function enables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See semc_interrupt_enable_t. For example, to enable the IP command done and error interrupt, do the following.

```
*     SEMC_EnableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt |
      kSEMC_IPCmdErrInterrupt);
*
```

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *mask* | SEMC interrupts to enable. This is a logical OR of the enumeration :: semc_interrupt-_enable_t. |

### 39.6.10 static void SEMC_DisableInterrupts ( SEMC_Type * *base*, uint32_t *mask* ) [inline], [static]

This function disables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See semc_interrupt_enable_t. For example, to disable the IP command done and error interrupt, do the following.

```
*     SEMC_DisableInterrupts(ENET,
      kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
*
```

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *mask* | SEMC interrupts to disable. This is a logical OR of the enumeration :: semc_interrupt_enable_t. |

### 39.6.11 static bool SEMC_GetStatusFlag ( SEMC_Type * *base* ) [inline], [static]

This function gets the SEMC interrupts event status. User can use the a logical OR of enumeration member as a mask. See semc_interrupt_enable_t.

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |

Returns

status flag, use status flag in semc_interrupt_enable_t to get the related status.

### 39.6.12 static void SEMC_ClearStatusFlags ( SEMC_Type * *base*, uint32_t *mask* ) [inline], [static]

The following status register flags can be cleared SEMC interrupt status.

Parameters

| | |
|---|---|
| *base* | SEMC base pointer |
| *mask* | The status flag mask, a logical OR of enumeration member semc_interrupt_enable_t. |

### 39.6.13 static bool SEMC_IsInIdle ( SEMC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |

Returns

True SEMC is in idle, false is not in idle.

### 39.6.14 status_t SEMC_SendIPCommand ( SEMC_Type * *base,* semc_mem_type_t *type,* uint32_t *address,* uint16_t *command,* uint32_t *write,* uint32_t * *read* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *type* | SEMC memory type. refer to "semc_mem_type_t" |
| *address* | SEMC device address. |
| *command* | SEMC IP command. For NAND device, we should use the SEMC_BuildNandIPCommand to get the right nand command. For NOR/DBI device, take refer to "semc_ipcmd_nor_dbi_t". For SRAM device, take refer to "semc_ipcmd_sram_t". For SDRAM device, take refer to "semc_ipcmd_sdram_t". |
| *write* | Data for write access. |
| *read* | Data pointer for read data out. |

### 39.6.15 static uint16_t SEMC_BuildNandIPCommand ( uint8_t *userCommand,* semc_ipcmd_nand_addrmode_t *addrMode,* semc_ipcmd_nand_cmdmode_t *cmdMode* ) [inline], [static]

This function build SEMC NAND IP command. The command is build of user command code, SEMC address mode and SEMC command mode.

Parameters

| | |
|---|---|
| *userCommand* | NAND device normal command. |
| *addrMode* | NAND address mode. Refer to "semc_ipcmd_nand_addrmode_t". |
| *cmdMode* | NAND command mode. Refer to "semc_ipcmd_nand_cmdmode_t". |

### 39.6.16 static bool SEMC_IsNandReady ( SEMC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |

Returns

True NAND is ready, false NAND is not ready.

### 39.6.17 status_t SEMC_IPCommandNandWrite ( SEMC_Type * *base,* uint32_t *address,* uint8_t * *data,* uint32_t *size_bytes* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *address* | SEMC NAND device address. |
| *data* | Data for write access. |
| *size_bytes* | Data length. |

### 39.6.18 status_t SEMC_IPCommandNandRead ( SEMC_Type * *base*, uint32_t *address*, uint8_t * *data*, uint32_t *size_bytes* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *address* | SEMC NAND device address. |
| *data* | Data pointer for data read out. |
| *size_bytes* | Data length. |

### 39.6.19 status_t SEMC_IPCommandNorWrite ( SEMC_Type * *base*, uint32_t *address*, uint8_t * *data*, uint32_t *size_bytes* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *address* | SEMC NOR device address. |
| *data* | Data for write access. |
| *size_bytes* | Data length. |

### 39.6.20 status_t SEMC_IPCommandNorRead ( SEMC_Type * *base*, uint32_t *address*, uint8_t * *data*, uint32_t *size_bytes* )

Parameters

| | |
|---|---|
| *base* | SEMC peripheral base address. |
| *address* | SEMC NOR device address. |
| *data* | Data pointer for data read out. |
| *size_bytes* | Data length. |

# Chapter 40
# SNVS: Secure Non-Volatile Storage

## 40.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage (SNVS) module.

The SNVS module is designed to safely hold security-related data such as cryptographic key, time counter, monotonic counter, and general purpose security information. The SNVS includes a low power part, namely SNVS_LP, that is battery backed by the SVNS (or VBAT) power domain. This enables it to keep this data valid and continue to increment the time counter when the power goes down in the rest of the SoC. The always-powered-up part of the module is isolated from the rest of the logic to ensure that it does not get corrupted when the SoC is powered down. The SNVS is designed to comply with Digital Rights Management (DRM) and other security application rules and requirements. This trusted hardware provides features that allow the system software designer to ensure that the data kept by the device is certifiable. Specially, it incorporates a security monitor that checks for various security conditions. If a security violation is indicated then it invalidates access to its sensitive data, and the secret data, for instance, Zeroizable Secret Key, is zeroized. the SNVS can be also configured to bypass its security features and protection mechanism. In this case it can be used by systems that do not require security.

## Modules

- Secure Non-Volatile Storage High-Power

## 40.2 Secure Non-Volatile Storage High-Power

### 40.2.1 Overview

The MCUXpresso SDK provides a Peripheral driver for the Secure Non-Volatile Storage High-Power(S-NVS-HP) module.

The SNVS_HP is in the chip's power-supply domain and thus receives the power along with the rest of the chip. The SNVS_HP provides an interface between the SNVS_LP and the rest of the system; there is no way to access the SNVS_LP registers except through the SNVS_HP. For access to the SNVS_LP registers, the SNVS_HP must be powered up. It uses a register access permission policy to determine whether the access to the particular registers is permitted.

### 40.2.2 SNVS_HP Driver Initialization and Configuration

### 40.2.3 SNVS_HP Driver Examples

#### Data Structures

- struct snvs_hp_rtc_datetime_t
  *Structure is used to hold the date and time. More...*
- struct snvs_hp_rtc_config_t
  *SNVS config structure. More...*

#### Macros

- #define SNVS_MAKE_HP_SV_FLAG(x) (1U << (SNVS_HPSVSR_SV0_SHIFT + (x)))
  *Macro to make security violation flag.*

#### Enumerations

- enum snvs_hp_interrupts_t {
  kSNVS_RTC_AlarmInterrupt = SNVS_HPCR_HPTA_EN_MASK,
  kSNVS_RTC_PeriodicInterrupt = SNVS_HPCR_PI_EN_MASK }
  *List of SNVS interrupts.*
- enum snvs_hp_status_flags_t {
  kSNVS_RTC_AlarmInterruptFlag = SNVS_HPSR_HPTA_MASK,
  kSNVS_RTC_PeriodicInterruptFlag = SNVS_HPSR_PI_MASK,
  kSNVS_ZMK_ZeroFlag = (int)SNVS_HPSR_ZMK_ZERO_MASK,
  kSNVS_OTPMK_ZeroFlag = SNVS_HPSR_OTPMK_ZERO_MASK }
  *List of SNVS flags.*

- enum snvs_hp_sv_status_flags_t {
  kSNVS_LP_ViolationFlag = SNVS_HPSVSR_SW_LPSV_MASK,
  kSNVS_ZMK_EccFailFlag = SNVS_HPSVSR_ZMK_ECC_FAIL_MASK,
  kSNVS_LP_SoftwareViolationFlag = SNVS_HPSVSR_SW_LPSV_MASK,
  kSNVS_FatalSoftwareViolationFlag = SNVS_HPSVSR_SW_FSV_MASK,
  kSNVS_SoftwareViolationFlag = SNVS_HPSVSR_SW_SV_MASK,
  kSNVS_Violation0Flag = SNVS_HPSVSR_SV0_MASK,
  kSNVS_Violation1Flag = SNVS_HPSVSR_SV1_MASK,
  kSNVS_Violation2Flag = SNVS_HPSVSR_SV2_MASK,
  kSNVS_Violation3Flag = SNVS_HPSVSR_SV3_MASK,
  kSNVS_Violation4Flag = SNVS_HPSVSR_SV4_MASK,
  kSNVS_Violation5Flag = SNVS_HPSVSR_SV5_MASK }
  *List of SNVS security violation flags.*
- enum snvs_hp_ssm_state_t {
  kSNVS_SSMInit = 0x00,
  kSNVS_SSMHardFail = 0x01,
  kSNVS_SSMSoftFail = 0x03,
  kSNVS_SSMInitInter = 0x08,
  kSNVS_SSMCheck = 0x09,
  kSNVS_SSMNonSecure = 0x0B,
  kSNVS_SSMTrusted = 0x0D,
  kSNVS_SSMSecure = 0x0F }

## Functions

- static void SNVS_HP_EnableMasterKeySelection (SNVS_Type ∗base, bool enable)
  *Enable or disable master key selection.*
- static void SNVS_HP_ProgramZeroizableMasterKey (SNVS_Type ∗base)
  *Trigger to program Zeroizable Master Key.*
- static void SNVS_HP_ChangeSSMState (SNVS_Type ∗base)
  *Trigger SSM State Transition.*
- static void SNVS_HP_SetSoftwareFatalSecurityViolation (SNVS_Type ∗base)
  *Trigger Software Fatal Security Violation.*
- static void SNVS_HP_SetSoftwareSecurityViolation (SNVS_Type ∗base)
  *Trigger Software Security Violation.*
- static snvs_hp_ssm_state_t SNVS_HP_GetSSMState (SNVS_Type ∗base)
  *Get current SSM State.*
- static void SNVS_HP_ResetLP (SNVS_Type ∗base)
  *Reset the SNVS LP chapter.*
- static uint32_t SNVS_HP_GetStatusFlags (SNVS_Type ∗base)
  *Get the SNVS HP status flags.*
- static void SNVS_HP_ClearStatusFlags (SNVS_Type ∗base, uint32_t mask)
  *Clear the SNVS HP status flags.*
- static uint32_t SNVS_HP_GetSecurityViolationStatusFlags (SNVS_Type ∗base)
  *Get the SNVS HP security violation status flags.*
- static void SNVS_HP_ClearSecurityViolationStatusFlags (SNVS_Type ∗base, uint32_t mask)
  *Clear the SNVS HP security violation status flags.*

## Driver version

- #define FSL_SNVS_HP_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  *Version 2.1.2.*

## Initialization and deinitialization

- void SNVS_HP_Init (SNVS_Type *base)
  *Initialize the SNVS.*
- void SNVS_HP_Deinit (SNVS_Type *base)
  *Deinitialize the SNVS.*
- void SNVS_HP_RTC_Init (SNVS_Type *base, const snvs_hp_rtc_config_t *config)
  *Ungates the SNVS clock and configures the peripheral for basic operation.*
- void SNVS_HP_RTC_Deinit (SNVS_Type *base)
  *Stops the RTC and SRTC timers.*
- void SNVS_HP_RTC_GetDefaultConfig (snvs_hp_rtc_config_t *config)
  *Fills in the SNVS config struct with the default settings.*

## Non secure RTC current Time & Alarm

- status_t SNVS_HP_RTC_SetDatetime (SNVS_Type *base, const snvs_hp_rtc_datetime_t *datetime)
  *Sets the SNVS RTC date and time according to the given time structure.*
- void SNVS_HP_RTC_GetDatetime (SNVS_Type *base, snvs_hp_rtc_datetime_t *datetime)
  *Gets the SNVS RTC time and stores it in the given time structure.*
- status_t SNVS_HP_RTC_SetAlarm (SNVS_Type *base, const snvs_hp_rtc_datetime_t *alarmTime)
  *Sets the SNVS RTC alarm time.*
- void SNVS_HP_RTC_GetAlarm (SNVS_Type *base, snvs_hp_rtc_datetime_t *datetime)
  *Returns the SNVS RTC alarm time.*
- void SNVS_HP_RTC_TimeSynchronize (SNVS_Type *base)
  *The function synchronizes RTC counter value with SRTC.*

## Interrupt Interface

- static void SNVS_HP_RTC_EnableInterrupts (SNVS_Type *base, uint32_t mask)
  *Enables the selected SNVS interrupts.*
- static void SNVS_HP_RTC_DisableInterrupts (SNVS_Type *base, uint32_t mask)
  *Disables the selected SNVS interrupts.*
- uint32_t SNVS_HP_RTC_GetEnabledInterrupts (SNVS_Type *base)
  *Gets the enabled SNVS interrupts.*

## Status Interface

- uint32_t SNVS_HP_RTC_GetStatusFlags (SNVS_Type *base)

*Gets the SNVS status flags.*
- static void SNVS_HP_RTC_ClearStatusFlags (SNVS_Type *base, uint32_t mask)
  *Clears the SNVS status flags.*

### Timer Start and Stop

- static void SNVS_HP_RTC_StartTimer (SNVS_Type *base)
  *Starts the SNVS RTC time counter.*
- static void SNVS_HP_RTC_StopTimer (SNVS_Type *base)
  *Stops the SNVS RTC time counter.*

### High Assurance Counter (HAC)

- static void SNVS_HP_EnableHighAssuranceCounter (SNVS_Type *base, bool enable)
  *Enable or disable the High Assurance Counter (HAC)*
- static void SNVS_HP_StartHighAssuranceCounter (SNVS_Type *base, bool start)
  *Start or stop the High Assurance Counter (HAC)*
- static void SNVS_HP_SetHighAssuranceCounterInitialValue (SNVS_Type *base, uint32_t value)
  *Set the High Assurance Counter (HAC) initialize value.*
- static void SNVS_HP_LoadHighAssuranceCounter (SNVS_Type *base)
  *Load the High Assurance Counter (HAC)*
- static uint32_t SNVS_HP_GetHighAssuranceCounter (SNVS_Type *base)
  *Get the current High Assurance Counter (HAC) value.*
- static void SNVS_HP_ClearHighAssuranceCounter (SNVS_Type *base)
  *Clear the High Assurance Counter (HAC)*
- static void SNVS_HP_LockHighAssuranceCounter (SNVS_Type *base)
  *Lock the High Assurance Counter (HAC)*

## 40.2.4 Data Structure Documentation

### 40.2.4.1 struct snvs_hp_rtc_datetime_t

### Data Fields

- 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.*

#### 40.2.4.1.0.56 Field Documentation

#### 40.2.4.1.0.56.1 uint16_t snvs_hp_rtc_datetime_t::year

#### 40.2.4.1.0.56.2 uint8_t snvs_hp_rtc_datetime_t::month

#### 40.2.4.1.0.56.3 uint8_t snvs_hp_rtc_datetime_t::day

#### 40.2.4.1.0.56.4 uint8_t snvs_hp_rtc_datetime_t::hour

#### 40.2.4.1.0.56.5 uint8_t snvs_hp_rtc_datetime_t::minute

#### 40.2.4.1.0.56.6 uint8_t snvs_hp_rtc_datetime_t::second

### 40.2.4.2 struct snvs_hp_rtc_config_t

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

#### Data Fields

- bool rtcCalEnable
  *true: RTC calibration mechanism is enabled; false:No calibration is used*
- uint32_t rtcCalValue
  *Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15.*
- uint32_t periodicInterruptFreq
  *Defines frequency of the periodic interrupt; Range from 0 to 15.*

## 40.2.5 Macro Definition Documentation

### 40.2.5.1 #define SNVS_MAKE_HP_SV_FLAG( *x* ) (1U << (SNVS_HPSVSR_SV0_SHIFT + (x)))

Macro help to make security violation flag kSNVS_Violation0Flag to kSNVS_Violation5Flag, For example, SNVS_MAKE_HP_SV_FLAG(0) is kSNVS_Violation0Flag.

### 40.2.6 Enumeration Type Documentation

#### 40.2.6.1 enum snvs_hp_interrupts_t

Enumerator

***kSNVS_RTC_AlarmInterrupt*** RTC time alarm.
***kSNVS_RTC_PeriodicInterrupt*** RTC periodic interrupt.

#### 40.2.6.2 enum snvs_hp_status_flags_t

Enumerator

***kSNVS_RTC_AlarmInterruptFlag*** RTC time alarm flag.
***kSNVS_RTC_PeriodicInterruptFlag*** RTC periodic interrupt flag.
***kSNVS_ZMK_ZeroFlag*** The ZMK is zero.
***kSNVS_OTPMK_ZeroFlag*** The OTPMK is zero.

#### 40.2.6.3 enum snvs_hp_sv_status_flags_t

Enumerator

***kSNVS_LP_ViolationFlag*** Low Power chapter Security Violation.
***kSNVS_ZMK_EccFailFlag*** Zeroizable Master Key Error Correcting Code Check Failure.
***kSNVS_LP_SoftwareViolationFlag*** LP Software Security Violation.
***kSNVS_FatalSoftwareViolationFlag*** Software Fatal Security Violation.
***kSNVS_SoftwareViolationFlag*** Software Security Violation.
***kSNVS_Violation0Flag*** Security Violation 0.
***kSNVS_Violation1Flag*** Security Violation 1.
***kSNVS_Violation2Flag*** Security Violation 2.
***kSNVS_Violation3Flag*** Security Violation 3.
***kSNVS_Violation4Flag*** Security Violation 4.
***kSNVS_Violation5Flag*** Security Violation 5.

#### 40.2.6.4 enum snvs_hp_ssm_state_t

Enumerator

***kSNVS_SSMInit*** Init.
***kSNVS_SSMHardFail*** Hard Fail.
***kSNVS_SSMSoftFail*** Soft Fail.
***kSNVS_SSMInitInter*** Init Intermediate (transition state between Init and Check)
***kSNVS_SSMCheck*** Check.

***kSNVS_SSMNonSecure*** Non-Secure.
***kSNVS_SSMTrusted*** Trusted.
***kSNVS_SSMSecure*** Secure.

### 40.2.7 Function Documentation

#### 40.2.7.1 void SNVS_HP_Init ( SNVS_Type * *base* )

Note

This API should be called at the beginning of the application using the SNVS driver.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.2 void SNVS_HP_Deinit ( SNVS_Type * *base* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.3 void SNVS_HP_RTC_Init ( SNVS_Type * *base*, const snvs_hp_rtc_config_t * *config* )

Note

This API should be called at the beginning of the application using the SNVS driver.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *config* | Pointer to the user's SNVS configuration structure. |

#### 40.2.7.4 void SNVS_HP_RTC_Deinit ( SNVS_Type * *base* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.5 void SNVS_HP_RTC_GetDefaultConfig ( snvs_hp_rtc_config_t * *config* )

The default values are as follows.

```
*    config->rtccalenable = false;
*    config->rtccalvalue = 0U;
*    config->PIFreq = 0U;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to the user's SNVS configuration structure. |

#### 40.2.7.6 status_t SNVS_HP_RTC_SetDatetime ( SNVS_Type * *base,* const snvs_hp_rtc_datetime_t * *datetime* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *datetime* | Pointer to the structure where the date and time details are stored. |

Returns

kStatus_Success: Success in setting the time and starting the SNVS RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

#### 40.2.7.7 void SNVS_HP_RTC_GetDatetime ( SNVS_Type * *base,* snvs_hp_rtc_datetime_t * *datetime* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

| | |
|---|---|
| *datetime* | Pointer to the structure where the date and time details are stored. |

#### 40.2.7.8 status_t SNVS_HP_RTC_SetAlarm ( SNVS_Type * *base*, const snvs_hp_rtc_datetime_t * *alarmTime* )

The function sets the RTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *alarmTime* | Pointer to the structure where the alarm time is stored. |

Returns

kStatus_Success: success in setting the SNVS RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

#### 40.2.7.9 void SNVS_HP_RTC_GetAlarm ( SNVS_Type * *base*, snvs_hp_rtc_datetime_t * *datetime* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *datetime* | Pointer to the structure where the alarm date and time details are stored. |

#### 40.2.7.10 void SNVS_HP_RTC_TimeSynchronize ( SNVS_Type * *base* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.11 static void SNVS_HP_RTC_EnableInterrupts ( SNVS_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration ::snvs_interrupt_enable_t |

#### 40.2.7.12 static void SNVS_HP_RTC_DisableInterrupts ( SNVS_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *mask* | The interrupts to enable. This is a logical OR of members of the enumeration ::snvs_interrupt_enable_t |

#### 40.2.7.13 uint32_t SNVS_HP_RTC_GetEnabledInterrupts ( SNVS_Type * *base* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

Returns

The enabled interrupts. This is the logical OR of members of the enumeration ::snvs_interrupt_enable_t

#### 40.2.7.14 uint32_t SNVS_HP_RTC_GetStatusFlags ( SNVS_Type * *base* )

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

Returns

The status flags. This is the logical OR of members of the enumeration ::snvs_status_flags_t

#### 40.2.7.15 static void SNVS_HP_RTC_ClearStatusFlags ( SNVS_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *mask* | The status flags to clear. This is a logical OR of members of the enumeration ::snvs_status_flags_t |

#### 40.2.7.16 static void SNVS_HP_RTC_StartTimer ( SNVS_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.17 static void SNVS_HP_RTC_StopTimer ( SNVS_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.18 static void SNVS_HP_EnableMasterKeySelection ( SNVS_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *enable* | Pass true to enable, false to disable. |

#### 40.2.7.19 static void SNVS_HP_ProgramZeroizableMasterKey ( SNVS_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *enable* | Pass true to enable, false to disable. |

#### 40.2.7.20 static void SNVS_HP_ChangeSSMState ( SNVS_Type * *base* ) [inline], [static]

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State –> Trusted (when Secure Boot or in Fab Configuration )
- Trusted State –> Secure
- Secure State –> Trusted
- Soft Fail –> Non-Secure

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.21 static void SNVS_HP_SetSoftwareFatalSecurityViolation ( SNVS_Type * *base* ) [inline], [static]

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.22 static void SNVS_HP_SetSoftwareSecurityViolation ( SNVS_Type * *base* ) [inline], [static]

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.23 static snvs_hp_ssm_state_t SNVS_HP_GetSSMState ( SNVS_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

Returns

Current SSM state

#### 40.2.7.24 static void SNVS_HP_ResetLP ( SNVS_Type * *base* ) [inline], [static]

Reset the LP chapter except SRTC and Time alarm.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.25 static void SNVS_HP_EnableHighAssuranceCounter ( SNVS_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *enable* | Pass true to enable, false to disable. |

#### 40.2.7.26 static void SNVS_HP_StartHighAssuranceCounter ( SNVS_Type * *base,* bool *start* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *start* | Pass true to start, false to stop. |

#### 40.2.7.27 static void SNVS_HP_SetHighAssuranceCounterInitialValue ( SNVS_Type * *base,* uint32_t *value* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *value* | The initial value to set. |

#### 40.2.7.28 static void SNVS_HP_LoadHighAssuranceCounter ( SNVS_Type * *base* ) [inline], [static]

This function loads the HAC initialize value to counter register.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

#### 40.2.7.29 static uint32_t SNVS_HP_GetHighAssuranceCounter ( SNVS_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

Returns

HAC currnet value.

#### 40.2.7.30 static void SNVS_HP_ClearHighAssuranceCounter ( SNVS_Type * *base* ) [inline], [static]

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

### 40.2.7.31 static void SNVS_HP_LockHighAssuranceCounter ( SNVS_Type * *base* ) [inline], [static]

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

### 40.2.7.32 static uint32_t SNVS_HP_GetStatusFlags ( SNVS_Type * *base* ) [inline], [static]

The flags are returned as the OR'ed value of snvs_hp_sgtatus_flags_t.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

Returns

The OR'ed value of status flags.

### 40.2.7.33 static void SNVS_HP_ClearStatusFlags ( SNVS_Type * *base,* uint32_t *mask* ) [inline], [static]

The flags to clear are passed in as the OR'ed value of snvs_hp_status_flags_t. Only these flags could be cleared using this API.

- kSNVS_RTC_PeriodicInterruptFlag
- kSNVS_RTC_AlarmInterruptFlag

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *mask* | OR'ed value of the flags to clear. |

#### 40.2.7.34 static uint32_t SNVS_HP_GetSecurityViolationStatusFlags ( SNVS_Type * *base* ) [inline], [static]

The flags are returned as the OR'ed value of snvs_hp_sv_status_flags_t.

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |

Returns

The OR'ed value of security violation status flags.

#### 40.2.7.35 static void SNVS_HP_ClearSecurityViolationStatusFlags ( SNVS_Type * *base*, uint32_t *mask* ) [inline], [static]

The flags to clear are passed in as the OR'ed value of snvs_hp_sv_status_flags_t. Only these flags could be cleared using this API.

- kSNVS_ZMK_EccFailFlag
- kSNVS_Violation0Flag
- kSNVS_Violation1Flag
- kSNVS_Violation2Flag
- kSNVS_Violation3Flag
- kSNVS_Violation4Flag
- kSNVS_Violation5Flag

Parameters

| | |
|---|---|
| *base* | SNVS peripheral base address |
| *mask* | OR'ed value of the flags to clear. |

# Chapter 41
# SPDIF: Sony/Philips Digital Interface

## 41.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Sony/Philips Digital Interface (SPDIF) module of MCUXpresso SDK devices.

SPDIF driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SPDIF initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPDIF peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPDIF functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the spdif_handle_t as the first parameter. Initialize the handle by calling the SPDIF_TransferTxCreateHandle() or SPDIF_TransferRxCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SPDIF_TransferSendNonBlocking() and SPDIF_TransferReceiveNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_SPDIF_TxIdle and kStatus_SPDIF_RxIdle status.

## 41.2 Typical use case

### 41.2.1 SPDIF Send/receive using an interrupt method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/spdif

### 41.2.2 SPDIF Send/receive using a DMA method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/spdif

## Modules

- SPDIF eDMA Driver

## Data Structures

- struct spdif_config_t

*SPDIF user configuration structure. More...*
- struct spdif_transfer_t
  *SPDIF transfer structure. More...*
- struct spdif_handle_t
  *SPDIF handle structure. More...*

## Macros

- #define SPDIF_XFER_QUEUE_SIZE (4)
  *SPDIF transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(∗ spdif_transfer_callback_t )(SPDIF_Type ∗base, spdif_handle_t ∗handle, status_t status, void ∗userData)
  *SPDIF transfer callback prototype.*

## Enumerations

- enum _spdif_status_t {
  kStatus_SPDIF_RxDPLLLocked = MAKE_STATUS(kStatusGroup_SPDIF, 0),
  kStatus_SPDIF_TxFIFOError = MAKE_STATUS(kStatusGroup_SPDIF, 1),
  kStatus_SPDIF_TxFIFOResync = MAKE_STATUS(kStatusGroup_SPDIF, 2),
  kStatus_SPDIF_RxCnew = MAKE_STATUS(kStatusGroup_SPDIF, 3),
  kStatus_SPDIF_ValidatyNoGood = MAKE_STATUS(kStatusGroup_SPDIF, 4),
  kStatus_SPDIF_RxIllegalSymbol = MAKE_STATUS(kStatusGroup_SPDIF, 5),
  kStatus_SPDIF_RxParityBitError = MAKE_STATUS(kStatusGroup_SPDIF, 6),
  kStatus_SPDIF_UChannelOverrun = MAKE_STATUS(kStatusGroup_SPDIF, 7),
  kStatus_SPDIF_QChannelOverrun = MAKE_STATUS(kStatusGroup_SPDIF, 8),
  kStatus_SPDIF_UQChannelSync = MAKE_STATUS(kStatusGroup_SPDIF, 9),
  kStatus_SPDIF_UQChannelFrameError = MAKE_STATUS(kStatusGroup_SPDIF, 10),
  kStatus_SPDIF_RxFIFOError = MAKE_STATUS(kStatusGroup_SPDIF, 11),
  kStatus_SPDIF_RxFIFOResync = MAKE_STATUS(kStatusGroup_SPDIF, 12),
  kStatus_SPDIF_LockLoss = MAKE_STATUS(kStatusGroup_SPDIF, 13),
  kStatus_SPDIF_TxIdle = MAKE_STATUS(kStatusGroup_SPDIF, 14),
  kStatus_SPDIF_RxIdle = MAKE_STATUS(kStatusGroup_SPDIF, 15),
  kStatus_SPDIF_QueueFull = MAKE_STATUS(kStatusGroup_SPDIF, 16) }
  *SPDIF return status.*
- enum spdif_rxfull_select_t {
  kSPDIF_RxFull1Sample = 0x0u,
  kSPDIF_RxFull4Samples,
  kSPDIF_RxFull8Samples,
  kSPDIF_RxFull16Samples }
  *SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.*
- enum spdif_txempty_select_t {
  kSPDIF_TxEmpty0Sample = 0x0u,
  kSPDIF_TxEmpty4Samples,
  kSPDIF_TxEmpty8Samples,

kSPDIF_TxEmpty12Samples }
*SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.*

- enum spdif_uchannel_source_t {
  kSPDIF_NoUChannel = 0x0U,
  kSPDIF_UChannelFromRx = 0x1U,
  kSPDIF_UChannelFromTx = 0x3U }
  *SPDIF U channel source.*
- enum spdif_gain_select_t {
  kSPDIF_GAIN_24 = 0x0U,
  kSPDIF_GAIN_16,
  kSPDIF_GAIN_12,
  kSPDIF_GAIN_8,
  kSPDIF_GAIN_6,
  kSPDIF_GAIN_4,
  kSPDIF_GAIN_3 }
  *SPDIF clock gain.*
- enum spdif_tx_source_t {
  kSPDIF_txFromReceiver = 0x1U,
  kSPDIF_txNormal = 0x5U }
  *SPDIF tx data source.*
- enum spdif_validity_config_t {
  kSPDIF_validityFlagAlwaysSet = 0x0U,
  kSPDIF_validityFlagAlwaysClear }
  *SPDIF tx data source.*
- enum _spdif_interrupt_enable_t {
  kSPDIF_RxDPLLLocked = SPDIF_SIE_LOCK_MASK,
  kSPDIF_TxFIFOError = SPDIF_SIE_TXUNOV_MASK,
  kSPDIF_TxFIFOResync = SPDIF_SIE_TXRESYN_MASK,
  kSPDIF_RxControlChannelChange = SPDIF_SIE_CNEW_MASK,
  kSPDIF_ValidityFlagNoGood = SPDIF_SIE_VALNOGOOD_MASK,
  kSPDIF_RxIllegalSymbol = SPDIF_SIE_SYMERR_MASK,
  kSPDIF_RxParityBitError = SPDIF_SIE_BITERR_MASK,
  kSPDIF_UChannelReceiveRegisterFull = SPDIF_SIE_URXFUL_MASK,
  kSPDIF_UChannelReceiveRegisterOverrun = SPDIF_SIE_URXOV_MASK,
  kSPDIF_QChannelReceiveRegisterFull = SPDIF_SIE_QRXFUL_MASK,
  kSPDIF_QChannelReceiveRegisterOverrun = SPDIF_SIE_QRXOV_MASK,
  kSPDIF_UQChannelSync = SPDIF_SIE_UQSYNC_MASK,
  kSPDIF_UQChannelFrameError = SPDIF_SIE_UQERR_MASK,
  kSPDIF_RxFIFOError = SPDIF_SIE_RXFIFOUNOV_MASK,
  kSPDIF_RxFIFOResync = SPDIF_SIE_RXFIFORESYN_MASK,
  kSPDIF_LockLoss = SPDIF_SIE_LOCKLOSS_MASK,
  kSPDIF_TxFIFOEmpty = SPDIF_SIE_TXEM_MASK,
  kSPDIF_RxFIFOFull = SPDIF_SIE_RXFIFOFUL_MASK }
  *The SPDIF interrupt enable flag.*
- enum _spdif_dma_enable_t {
  kSPDIF_RxDMAEnable = SPDIF_SCR_DMA_RX_EN_MASK,

kSPDIF_TxDMAEnable = SPDIF_SCR_DMA_TX_EN_MASK }
*The DMA request sources.*

## Driver version

- #define FSL_SPDIF_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *Version 2.0.2.*

## Initialization and deinitialization

- void SPDIF_Init (SPDIF_Type *base, const spdif_config_t *config)
  *Initializes the SPDIF peripheral.*
- void SPDIF_GetDefaultConfig (spdif_config_t *config)
  *Sets the SPDIF configuration structure to default values.*
- void SPDIF_Deinit (SPDIF_Type *base)
  *De-initializes the SPDIF peripheral.*
- static void SPDIF_TxFIFOReset (SPDIF_Type *base)
  *Resets the SPDIF Tx.*
- static void SPDIF_RxFIFOReset (SPDIF_Type *base)
  *Resets the SPDIF Rx.*
- void SPDIF_TxEnable (SPDIF_Type *base, bool enable)
  *Enables/disables the SPDIF Tx.*
- static void SPDIF_RxEnable (SPDIF_Type *base, bool enable)
  *Enables/disables the SPDIF Rx.*

## Status

- static uint32_t SPDIF_GetStatusFlag (SPDIF_Type *base)
  *Gets the SPDIF status flag state.*
- static void SPDIF_ClearStatusFlags (SPDIF_Type *base, uint32_t mask)
  *Clears the SPDIF status flag state.*

## Interrupts

- static void SPDIF_EnableInterrupts (SPDIF_Type *base, uint32_t mask)
  *Enables the SPDIF Tx interrupt requests.*
- static void SPDIF_DisableInterrupts (SPDIF_Type *base, uint32_t mask)
  *Disables the SPDIF Tx interrupt requests.*

## DMA Control

- static void SPDIF_EnableDMA (SPDIF_Type *base, uint32_t mask, bool enable)
  *Enables/disables the SPDIF DMA requests.*
- static uint32_t SPDIF_TxGetLeftDataRegisterAddress (SPDIF_Type *base)
  *Gets the SPDIF Tx left data register address.*
- static uint32_t SPDIF_TxGetRightDataRegisterAddress (SPDIF_Type *base)
  *Gets the SPDIF Tx right data register address.*
- static uint32_t SPDIF_RxGetLeftDataRegisterAddress (SPDIF_Type *base)
  *Gets the SPDIF Rx left data register address.*
- static uint32_t SPDIF_RxGetRightDataRegisterAddress (SPDIF_Type *base)
  *Gets the SPDIF Rx right data register address.*

## Bus Operations

- void SPDIF_TxSetSampleRate (SPDIF_Type *base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz)
  *Configures the SPDIF Tx sample rate.*
- uint32_t SPDIF_GetRxSampleRate (SPDIF_Type *base, uint32_t clockSourceFreq_Hz)
  *Configures the SPDIF Rx audio format.*
- void SPDIF_WriteBlocking (SPDIF_Type *base, uint8_t *buffer, uint32_t size)
  *Sends data using a blocking method.*
- static void SPDIF_WriteLeftData (SPDIF_Type *base, uint32_t data)
  *Writes data into SPDIF FIFO.*
- static void SPDIF_WriteRightData (SPDIF_Type *base, uint32_t data)
  *Writes data into SPDIF FIFO.*
- static void SPDIF_WriteChannelStatusHigh (SPDIF_Type *base, uint32_t data)
  *Writes data into SPDIF FIFO.*
- static void SPDIF_WriteChannelStatusLow (SPDIF_Type *base, uint32_t data)
  *Writes data into SPDIF FIFO.*
- void SPDIF_ReadBlocking (SPDIF_Type *base, uint8_t *buffer, uint32_t size)
  *Receives data using a blocking method.*
- static uint32_t SPDIF_ReadLeftData (SPDIF_Type *base)
  *Reads data from the SPDIF FIFO.*
- static uint32_t SPDIF_ReadRightData (SPDIF_Type *base)
  *Reads data from the SPDIF FIFO.*
- static uint32_t SPDIF_ReadChannelStatusHigh (SPDIF_Type *base)
  *Reads data from the SPDIF FIFO.*
- static uint32_t SPDIF_ReadChannelStatusLow (SPDIF_Type *base)
  *Reads data from the SPDIF FIFO.*
- static uint32_t SPDIF_ReadQChannel (SPDIF_Type *base)
  *Reads data from the SPDIF FIFO.*
- static uint32_t SPDIF_ReadUChannel (SPDIF_Type *base)
  *Reads data from the SPDIF FIFO.*

## Transactional

- void SPDIF_TransferTxCreateHandle (SPDIF_Type *base, spdif_handle_t *handle, spdif_transfer_callback_t callback, void *userData)
  *Initializes the SPDIF Tx handle.*
- void SPDIF_TransferRxCreateHandle (SPDIF_Type *base, spdif_handle_t *handle, spdif_transfer_callback_t callback, void *userData)
  *Initializes the SPDIF Rx handle.*
- status_t SPDIF_TransferSendNonBlocking (SPDIF_Type *base, spdif_handle_t *handle, spdif_transfer_t *xfer)
  *Performs an interrupt non-blocking send transfer on SPDIF.*
- status_t SPDIF_TransferReceiveNonBlocking (SPDIF_Type *base, spdif_handle_t *handle, spdif_transfer_t *xfer)
  *Performs an interrupt non-blocking receive transfer on SPDIF.*
- status_t SPDIF_TransferGetSendCount (SPDIF_Type *base, spdif_handle_t *handle, size_t *count)
  *Gets a set byte count.*
- status_t SPDIF_TransferGetReceiveCount (SPDIF_Type *base, spdif_handle_t *handle, size_t *count)
  *Gets a received byte count.*

- void SPDIF_TransferAbortSend (SPDIF_Type *base, spdif_handle_t *handle)
  *Aborts the current send.*
- void SPDIF_TransferAbortReceive (SPDIF_Type *base, spdif_handle_t *handle)
  *Aborts the current IRQ receive.*
- void SPDIF_TransferTxHandleIRQ (SPDIF_Type *base, spdif_handle_t *handle)
  *Tx interrupt handler.*
- void SPDIF_TransferRxHandleIRQ (SPDIF_Type *base, spdif_handle_t *handle)
  *Tx interrupt handler.*

## 41.3 Data Structure Documentation

### 41.3.1 struct spdif_config_t

#### Data Fields

- bool isTxAutoSync
  *If auto sync mechanism open.*
- bool isRxAutoSync
  *If auto sync mechanism open.*
- uint8_t DPLLClkSource
  *SPDIF DPLL clock source, range from 0~15, meaning is chip-specific.*
- uint8_t txClkSource
  *SPDIF tx clock source, range from 0~7, meaning is chip-specific.*
- spdif_rxfull_select_t rxFullSelect
  *SPDIF rx buffer full select.*
- spdif_txempty_select_t txFullSelect
  *SPDIF tx buffer empty select.*
- spdif_uchannel_source_t uChannelSrc
  *U channel source.*
- spdif_tx_source_t txSource
  *SPDIF tx data source.*
- spdif_validity_config_t validityConfig
  *Validity flag config.*
- spdif_gain_select_t gain
  *Rx receive clock measure gain parameter.*

##### 41.3.1.0.0.57 Field Documentation

###### 41.3.1.0.0.57.1 spdif_gain_select_t spdif_config_t::gain

### 41.3.2 struct spdif_transfer_t

#### Data Fields

- uint8_t * data
  *Data start address to transfer.*
- uint8_t * qdata
  *Data buffer for Q channel.*
- uint8_t * udata

*Data buffer for C channel.*
- size_t dataSize
  *Transfer size.*

#### 41.3.2.0.0.58 Field Documentation

#### 41.3.2.0.0.58.1 uint8_t* spdif_transfer_t::data

#### 41.3.2.0.0.58.2 size_t spdif_transfer_t::dataSize

### 41.3.3 struct _spdif_handle

### Data Fields

- uint32_t state
  *Transfer status.*
- spdif_transfer_callback_t callback
  *Callback function called at transfer event.*
- void * userData
  *Callback parameter passed to callback function.*
- spdif_transfer_t spdifQueue [SPDIF_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*
- size_t transferSize [SPDIF_XFER_QUEUE_SIZE]
  *Data bytes need to transfer.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*
- uint8_t watermark
  *Watermark value.*

## 41.4 Macro Definition Documentation

### 41.4.1 #define SPDIF_XFER_QUEUE_SIZE (4)

## 41.5 Enumeration Type Documentation

### 41.5.1 enum _spdif_status_t

Enumerator

***kStatus_SPDIF_RxDPLLLocked*** SPDIF Rx PLL locked.
***kStatus_SPDIF_TxFIFOError*** SPDIF Tx FIFO error.
***kStatus_SPDIF_TxFIFOResync*** SPDIF Tx left and right FIFO resync.
***kStatus_SPDIF_RxCnew*** SPDIF Rx status channel value updated.
***kStatus_SPDIF_ValidatyNoGood*** SPDIF validaty flag not good.
***kStatus_SPDIF_RxIllegalSymbol*** SPDIF Rx receive illegal symbol.
***kStatus_SPDIF_RxParityBitError*** SPDIF Rx parity bit error.
***kStatus_SPDIF_UChannelOverrun*** SPDIF receive U channel overrun.

***kStatus_SPDIF_QChannelOverrun*** SPDIF receive Q channel overrun.
***kStatus_SPDIF_UQChannelSync*** SPDIF U/Q channel sync found.
***kStatus_SPDIF_UQChannelFrameError*** SPDIF U/Q channel frame error.
***kStatus_SPDIF_RxFIFOError*** SPDIF Rx FIFO error.
***kStatus_SPDIF_RxFIFOResync*** SPDIF Rx left and right FIFO resync.
***kStatus_SPDIF_LockLoss*** SPDIF Rx PLL clock lock loss.
***kStatus_SPDIF_TxIdle*** SPDIF Tx is idle.
***kStatus_SPDIF_RxIdle*** SPDIF Rx is idle.
***kStatus_SPDIF_QueueFull*** SPDIF queue full.

### 41.5.2 enum spdif_rxfull_select_t

Enumerator

***kSPDIF_RxFull1Sample*** Rx full at least 1 sample in left and right FIFO.
***kSPDIF_RxFull4Samples*** Rx full at least 4 sample in left and right FIFO.
***kSPDIF_RxFull8Samples*** Rx full at least 8 sample in left and right FIFO.
***kSPDIF_RxFull16Samples*** Rx full at least 16 sample in left and right FIFO.

### 41.5.3 enum spdif_txempty_select_t

Enumerator

***kSPDIF_TxEmpty0Sample*** Tx empty at most 0 sample in left and right FIFO.
***kSPDIF_TxEmpty4Samples*** Tx empty at most 4 sample in left and right FIFO.
***kSPDIF_TxEmpty8Samples*** Tx empty at most 8 sample in left and right FIFO.
***kSPDIF_TxEmpty12Samples*** Tx empty at most 12 sample in left and right FIFO.

### 41.5.4 enum spdif_uchannel_source_t

Enumerator

***kSPDIF_NoUChannel*** No embedded U channel.
***kSPDIF_UChannelFromRx*** U channel from receiver, it is CD mode.
***kSPDIF_UChannelFromTx*** U channel from on chip tx.

### 41.5.5 enum spdif_gain_select_t

Enumerator

***kSPDIF_GAIN_24*** Gain select is 24.

***kSPDIF_GAIN_16*** Gain select is 16.
***kSPDIF_GAIN_12*** Gain select is 12.
***kSPDIF_GAIN_8*** Gain select is 8.
***kSPDIF_GAIN_6*** Gain select is 6.
***kSPDIF_GAIN_4*** Gain select is 4.
***kSPDIF_GAIN_3*** Gain select is 3.

### 41.5.6 enum spdif_tx_source_t

Enumerator

***kSPDIF_txFromReceiver*** Tx data directly through SPDIF receiver.
***kSPDIF_txNormal*** Normal operation, data from processor.

### 41.5.7 enum spdif_validity_config_t

Enumerator

***kSPDIF_validityFlagAlwaysSet*** Outgoing validity flags always set.
***kSPDIF_validityFlagAlwaysClear*** Outgoing validity flags always clear.

### 41.5.8 enum _spdif_interrupt_enable_t

Enumerator

***kSPDIF_RxDPLLLocked*** SPDIF DPLL locked.
***kSPDIF_TxFIFOError*** Tx FIFO underrun or overrun.
***kSPDIF_TxFIFOResync*** Tx FIFO left and right channel resync.
***kSPDIF_RxControlChannelChange*** SPDIF Rx control channel value changed.
***kSPDIF_ValidityFlagNoGood*** SPDIF validity flag no good.
***kSPDIF_RxIllegalSymbol*** SPDIF receiver found illegal symbol.
***kSPDIF_RxParityBitError*** SPDIF receiver found parity bit error.
***kSPDIF_UChannelReceiveRegisterFull*** SPDIF U channel revceive register full.
***kSPDIF_UChannelReceiveRegisterOverrun*** SPDIF U channel receive register overrun.
***kSPDIF_QChannelReceiveRegisterFull*** SPDIF Q channel receive reigster full.
***kSPDIF_QChannelReceiveRegisterOverrun*** SPDIF Q channel receive register overrun.
***kSPDIF_UQChannelSync*** SPDIF U/Q channel sync found.
***kSPDIF_UQChannelFrameError*** SPDIF U/Q channel frame error.
***kSPDIF_RxFIFOError*** SPDIF Rx FIFO underrun/overrun.
***kSPDIF_RxFIFOResync*** SPDIF Rx left and right FIFO resync.
***kSPDIF_LockLoss*** SPDIF receiver loss of lock.

***kSPDIF_TxFIFOEmpty*** SPDIF Tx FIFO empty.
***kSPDIF_RxFIFOFull*** SPDIF Rx FIFO full.

### 41.5.9 enum _spdif_dma_enable_t

Enumerator

***kSPDIF_RxDMAEnable*** Rx FIFO full.
***kSPDIF_TxDMAEnable*** Tx FIFO empty.

## 41.6 Function Documentation

### 41.6.1 void SPDIF_Init ( SPDIF_Type * *base*, const spdif_config_t * *config* )

Ungates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by SPDIF_GetDefaultConfig().

Note

This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *config* | SPDIF configuration structure. |

### 41.6.2 void SPDIF_GetDefaultConfig ( spdif_config_t * *config* )

This API initializes the configuration structure for use in SPDIF_Init. The initialized structure can remain unchanged in SPDIF_Init, or it can be modified before calling SPDIF_Init. This is an example.

```
spdif_config_t config;
SPDIF_GetDefaultConfig(&config);
```

Parameters

| | |
|---|---|
| *config* | pointer to master configuration structure |

### 41.6.3 void SPDIF_Deinit ( SPDIF_Type * *base* )

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

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |

### 41.6.4 static void SPDIF_TxFIFOReset ( SPDIF_Type * *base* ) [inline], [static]

This function makes Tx FIFO in reset mode.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |

### 41.6.5 static void SPDIF_RxFIFOReset ( SPDIF_Type * *base* ) [inline], [static]

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |

### 41.6.6 void SPDIF_TxEnable ( SPDIF_Type * *base*, bool *enable* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *enable* | True means enable SPDIF Tx, false means disable. |

### 41.6.7 static void SPDIF_RxEnable ( SPDIF_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *enable* | True means enable SPDIF Rx, false means disable. |

### 41.6.8 static uint32_t SPDIF_GetStatusFlag ( SPDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |

Returns

SPDIF status flag value. Use the _spdif_interrupt_enable_t to get the status value needed.

### 41.6.9 static void SPDIF_ClearStatusFlags ( SPDIF_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *mask* | State mask. It can be a combination of the _spdif_interrupt_enable_t member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits:<br>• kSPDIF_UChannelReceiveRegisterFull<br>• kSPDIF_QChannelReceiveRegisterFull<br>• kSPDIF_TxFIFOEmpty<br>• kSPDIF_RxFIFOFull |

### 41.6.10 static void SPDIF_EnableInterrupts ( SPDIF_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *mask* | interrupt source The parameter can be a combination of the following sources if defined.<br>• kSPDIF_WordStartInterruptEnable<br>• kSPDIF_SyncErrorInterruptEnable<br>• kSPDIF_FIFOWarningInterruptEnable<br>• kSPDIF_FIFORequestInterruptEnable<br>• kSPDIF_FIFOErrorInterruptEnable |

### 41.6.11 static void SPDIF_DisableInterrupts ( SPDIF_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *mask* | interrupt source The parameter can be a combination of the following sources if defined.<br>• kSPDIF_WordStartInterruptEnable<br>• kSPDIF_SyncErrorInterruptEnable<br>• kSPDIF_FIFOWarningInterruptEnable<br>• kSPDIF_FIFORequestInterruptEnable<br>• kSPDIF_FIFOErrorInterruptEnable |

### 41.6.12 static void SPDIF_EnableDMA ( SPDIF_Type * *base,* uint32_t *mask,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *mask* | SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined<br>• kSPDIF_RxDMAEnable<br>• kSPDIF_TxDMAEnable |
| *enable* | True means enable DMA, false means disable DMA. |

### 41.6.13 static uint32_t SPDIF_TxGetLeftDataRegisterAddress ( SPDIF_Type * *base* ) [inline], [static]

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

data register address.

### 41.6.14 static uint32_t SPDIF_TxGetRightDataRegisterAddress ( SPDIF_Type * *base* ) [inline], [static]

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

data register address.

### 41.6.15 static uint32_t SPDIF_RxGetLeftDataRegisterAddress ( SPDIF_Type * *base* ) [inline], [static]

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

data register address.

### 41.6.16 static uint32_t SPDIF_RxGetRightDataRegisterAddress ( SPDIF_Type * *base* ) [inline], [static]

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

data register address.

### 41.6.17 void SPDIF_TxSetSampleRate ( SPDIF_Type * *base*, uint32_t *sampleRate_Hz*, uint32_t *sourceClockFreq_Hz* )

The audio format can be changed at run-time. This function configures the sample rate.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *sampleRate_Hz* | SPDIF sample rate frequency in Hz. |
| *sourceClock-Freq_Hz* | SPDIF tx clock source frequency in Hz. |

### 41.6.18 uint32_t SPDIF_GetRxSampleRate ( SPDIF_Type * *base*, uint32_t *clockSourceFreq_Hz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *clockSource-Freq_Hz* | SPDIF system clock frequency in hz. |

### 41.6.19 void SPDIF_WriteBlocking ( SPDIF_Type * *base*, uint8_t * *buffer*, uint32_t *size* )

Note

This function blocks by polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *buffer* | Pointer to the data to be written. |
| *size* | Bytes to be written. |

### 41.6.20 static void SPDIF_WriteLeftData ( SPDIF_Type * *base,* uint32_t *data* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *data* | Data needs to be written. |

### 41.6.21 static void SPDIF_WriteRightData ( SPDIF_Type * *base,* uint32_t *data* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *data* | Data needs to be written. |

### 41.6.22 static void SPDIF_WriteChannelStatusHigh ( SPDIF_Type * *base,* uint32_t *data* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *data* | Data needs to be written. |

### 41.6.23 static void SPDIF_WriteChannelStatusLow ( SPDIF_Type * *base,* uint32_t *data* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *data* | Data needs to be written. |

### 41.6.24 void SPDIF_ReadBlocking ( SPDIF_Type * *base,* uint8_t * *buffer,* uint32_t *size* )

Note

This function blocks by polling until data is ready to be sent.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *buffer* | Pointer to the data to be read. |
| *size* | Bytes to be read. |

### 41.6.25 static uint32_t SPDIF_ReadLeftData ( SPDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

Data in SPDIF FIFO.

### 41.6.26 static uint32_t SPDIF_ReadRightData ( SPDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

Data in SPDIF FIFO.

### 41.6.27 static uint32_t SPDIF_ReadChannelStatusHigh ( SPDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

Data in SPDIF FIFO.

### 41.6.28 static uint32_t SPDIF_ReadChannelStatusLow ( SPDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

Data in SPDIF FIFO.

### 41.6.29 static uint32_t SPDIF_ReadQChannel ( SPDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

Data in SPDIF FIFO.

### 41.6.30 static uint32_t SPDIF_ReadUChannel ( SPDIF_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

Returns

Data in SPDIF FIFO.

### 41.6.31 void SPDIF_TransferTxCreateHandle ( SPDIF_Type * *base,* spdif_handle_t * *handle,* spdif_transfer_callback_t *callback,* void * *userData* )

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *handle* | SPDIF handle pointer. |
| *callback* | Pointer to the user callback function. |
| *userData* | User parameter passed to the callback function |

### 41.6.32 void SPDIF_TransferRxCreateHandle ( SPDIF_Type * *base,* spdif_handle_t * *handle,* spdif_transfer_callback_t *callback,* void * *userData* )

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | SPDIF handle pointer. |
| *callback* | Pointer to the user callback function. |
| *userData* | User parameter passed to the callback function. |

### 41.6.33 status_t SPDIF_TransferSendNonBlocking ( SPDIF_Type * *base*, spdif_handle_t * *handle*, spdif_transfer_t * *xfer* )

Note

This API returns immediately after the transfer initiates. Call the SPDIF_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | Pointer to the spdif_handle_t structure which stores the transfer state. |
| *xfer* | Pointer to the spdif_transfer_t structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully started the data receive. |
| *kStatus_SPDIF_TxBusy* | Previous receive still not finished. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

### 41.6.34 status_t SPDIF_TransferReceiveNonBlocking ( SPDIF_Type * *base*, spdif_handle_t * *handle*, spdif_transfer_t * *xfer* )

Note

This API returns immediately after the transfer initiates. Call the SPDIF_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *handle* | Pointer to the spdif_handle_t structure which stores the transfer state. |
| *xfer* | Pointer to the spdif_transfer_t structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Successfully started the data receive. |
| *kStatus_SPDIF_RxBusy* | Previous receive still not finished. |
| *kStatus_InvalidArgument* | The input parameter is invalid. |

### 41.6.35 status_t SPDIF_TransferGetSendCount ( SPDIF_Type * *base*, spdif_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | Pointer to the spdif_handle_t structure which stores the transfer state. |
| *count* | Bytes count sent. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

### 41.6.36 status_t SPDIF_TransferGetReceiveCount ( SPDIF_Type * *base*, spdif_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |

| | |
|---|---|
| *handle* | Pointer to the spdif_handle_t structure which stores the transfer state. |
| *count* | Bytes count received. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is not a non-blocking transaction currently in progress. |

### 41.6.37 void SPDIF_TransferAbortSend ( SPDIF_Type * *base,* spdif_handle_t * *handle* )

Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | Pointer to the spdif_handle_t structure which stores the transfer state. |

### 41.6.38 void SPDIF_TransferAbortReceive ( SPDIF_Type * *base,* spdif_handle_t * *handle* )

Note

This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *handle* | Pointer to the spdif_handle_t structure which stores the transfer state. |

### 41.6.39 void SPDIF_TransferTxHandleIRQ ( SPDIF_Type * *base,* spdif_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | Pointer to the spdif_handle_t structure. |

### 41.6.40 void SPDIF_TransferRxHandleIRQ ( SPDIF_Type * *base,* spdif_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | Pointer to the spdif_handle_t structure. |

## 41.7 SPDIF eDMA Driver

### 41.7.1 Overview

#### Data Structures

- struct spdif_edma_transfer_t
  *SPDIF transfer structure. More...*
- struct spdif_edma_handle_t
  *SPDIF DMA transfer handle, users should not touch the content of the handle. More...*

#### Typedefs

- typedef void(∗ spdif_edma_callback_t )(SPDIF_Type ∗base, spdif_edma_handle_t ∗handle, status_t status, void ∗userData)
  *SPDIF eDMA transfer callback function for finish and error.*

#### Driver version

- #define FSL_SPDIF_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *Version 2.0.1.*

#### eDMA Transactional

- void SPDIF_TransferTxCreateHandleEDMA (SPDIF_Type ∗base, spdif_edma_handle_t ∗handle, spdif_edma_callback_t callback, void ∗userData, edma_handle_t ∗dmaLeftHandle, edma_handle_t ∗dmaRightHandle)
  *Initializes the SPDIF eDMA handle.*
- void SPDIF_TransferRxCreateHandleEDMA (SPDIF_Type ∗base, spdif_edma_handle_t ∗handle, spdif_edma_callback_t callback, void ∗userData, edma_handle_t ∗dmaLeftHandle, edma_handle_t ∗dmaRightHandle)
  *Initializes the SPDIF Rx eDMA handle.*
- status_t SPDIF_TransferSendEDMA (SPDIF_Type ∗base, spdif_edma_handle_t ∗handle, spdif_edma_transfer_t ∗xfer)
  *Performs a non-blocking SPDIF transfer using DMA.*
- status_t SPDIF_TransferReceiveEDMA (SPDIF_Type ∗base, spdif_edma_handle_t ∗handle, spdif_edma_transfer_t ∗xfer)
  *Performs a non-blocking SPDIF receive using eDMA.*
- void SPDIF_TransferAbortSendEDMA (SPDIF_Type ∗base, spdif_edma_handle_t ∗handle)
  *Aborts a SPDIF transfer using eDMA.*
- void SPDIF_TransferAbortReceiveEDMA (SPDIF_Type ∗base, spdif_edma_handle_t ∗handle)
  *Aborts a SPDIF receive using eDMA.*
- status_t SPDIF_TransferGetSendCountEDMA (SPDIF_Type ∗base, spdif_edma_handle_t ∗handle, size_t ∗count)
  *Gets byte count sent by SPDIF.*

- status_t SPDIF_TransferGetReceiveCountEDMA (SPDIF_Type *base, spdif_edma_handle_t *handle, size_t *count)
  *Gets byte count received by SPDIF.*

## 41.7.2 Data Structure Documentation

### 41.7.2.1 struct spdif_edma_transfer_t

#### Data Fields

- uint8_t * leftData
  *Data start address to transfer.*
- uint8_t * rightData
  *Data start address to transfer.*
- size_t dataSize
  *Transfer size.*

#### 41.7.2.1.0.59 Field Documentation

##### 41.7.2.1.0.59.1 uint8_t* spdif_edma_transfer_t::leftData

##### 41.7.2.1.0.59.2 uint8_t* spdif_edma_transfer_t::rightData

##### 41.7.2.1.0.59.3 size_t spdif_edma_transfer_t::dataSize

### 41.7.2.2 struct _spdif_edma_handle

#### Data Fields

- edma_handle_t * dmaLeftHandle
  *DMA handler for SPDIF left channel.*
- edma_handle_t * dmaRightHandle
  *DMA handler for SPDIF right channel.*
- uint8_t nbytes
  *eDMA minor byte transfer count initially configured.*
- uint8_t count
  *The transfer data count in a DMA request.*
- uint32_t state
  *Internal state for SPDIF eDMA transfer.*
- spdif_edma_callback_t callback
  *Callback for users while transfer finish or error occurs.*
- void * userData
  *User callback parameter.*
- edma_tcd_t leftTcd [SPDIF_XFER_QUEUE_SIZE+1U]
  *TCD pool for eDMA transfer.*
- edma_tcd_t rightTcd [SPDIF_XFER_QUEUE_SIZE+1U]
  *TCD pool for eDMA transfer.*
- spdif_edma_transfer_t spdifQueue [SPDIF_XFER_QUEUE_SIZE]
  *Transfer queue storing queued transfer.*

- size_t transferSize [SPDIF_XFER_QUEUE_SIZE]
  *Data bytes need to transfer, left and right are the same, so use one.*
- volatile uint8_t queueUser
  *Index for user to queue transfer.*
- volatile uint8_t queueDriver
  *Index for driver to get the transfer data and size.*

#### 41.7.2.2.0.60 Field Documentation

##### 41.7.2.2.0.60.1 uint8_t spdif_edma_handle_t::nbytes

##### 41.7.2.2.0.60.2 edma_tcd_t spdif_edma_handle_t::leftTcd[SPDIF_XFER_QUEUE_SIZE+1U]

##### 41.7.2.2.0.60.3 edma_tcd_t spdif_edma_handle_t::rightTcd[SPDIF_XFER_QUEUE_SIZE+1U]

##### 41.7.2.2.0.60.4 spdif_edma_transfer_t spdif_edma_handle_t::spdifQueue[SPDIF_XFER_QUEUE_SIZE]

##### 41.7.2.2.0.60.5 volatile uint8_t spdif_edma_handle_t::queueUser

### 41.7.3 Function Documentation

#### 41.7.3.1 void SPDIF_TransferTxCreateHandleEDMA ( SPDIF_Type * *base*, spdif_edma_handle_t * *handle*, spdif_edma_callback_t *callback*, void * *userData*, edma_handle_t * *dmaLeftHandle*, edma_handle_t * *dmaRightHandle* )

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

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | SPDIF eDMA handle pointer. |
| *base* | SPDIF peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaLeftHandle* | eDMA handle pointer for left channel, this handle shall be static allocated by users. |
| *dmaRightHandle* | eDMA handle pointer for right channel, this handle shall be static allocated by users. |

#### 41.7.3.2 void SPDIF_TransferRxCreateHandleEDMA ( SPDIF_Type * *base,* spdif_edma_handle_t * *handle,* spdif_edma_callback_t *callback,* void * *userData,* edma_handle_t * *dmaLeftHandle,* edma_handle_t * *dmaRightHandle* )

This function initializes the SPDIF slave DMA handle, which can be used for other SPDIF master transactional APIs. Usually, for a specified SPDIF instance, call this API once to get the initialized handle.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | SPDIF eDMA handle pointer. |
| *base* | SPDIF peripheral base address. |
| *callback* | Pointer to user callback function. |
| *userData* | User parameter passed to the callback function. |
| *dmaLeftHandle* | eDMA handle pointer for left channel, this handle shall be static allocated by users. |
| *dmaRight-Handle* | eDMA handle pointer for right channel, this handle shall be static allocated by users. |

#### 41.7.3.3 status_t SPDIF_TransferSendEDMA ( SPDIF_Type * *base*, spdif_edma_handle_t * *handle*, spdif_edma_transfer_t * *xfer* )

Note

This interface returns immediately after the transfer initiates. Call SPDIF_GetTransferStatus to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | SPDIF eDMA handle pointer. |
| *xfer* | Pointer to the DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a SPDIF eDMA send successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |
| *kStatus_TxBusy* | SPDIF is busy sending data. |

#### 41.7.3.4 status_t SPDIF_TransferReceiveEDMA ( SPDIF_Type * *base*, spdif_edma_handle_t * *handle*, spdif_edma_transfer_t * *xfer* )

Note

This interface returns immediately after the transfer initiates. Call the SPDIF_GetReceive-RemainingBytes to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *handle* | SPDIF eDMA handle pointer. |
| *xfer* | Pointer to DMA transfer structure. |

Return values

| | |
|---|---|
| *kStatus_Success* | Start a SPDIF eDMA receive successfully. |
| *kStatus_InvalidArgument* | The input argument is invalid. |
| *kStatus_RxBusy* | SPDIF is busy receiving data. |

#### 41.7.3.5 void SPDIF_TransferAbortSendEDMA ( SPDIF_Type * *base*, spdif_edma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | SPDIF eDMA handle pointer. |

#### 41.7.3.6 void SPDIF_TransferAbortReceiveEDMA ( SPDIF_Type * *base*, spdif_edma_handle_t * *handle* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *handle* | SPDIF eDMA handle pointer. |

#### 41.7.3.7 status_t SPDIF_TransferGetSendCountEDMA ( SPDIF_Type * *base*, spdif_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer. |
| *handle* | SPDIF eDMA handle pointer. |
| *count* | Bytes count sent by SPDIF. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is no non-blocking transaction in progress. |

#### 41.7.3.8 status_t SPDIF_TransferGetReceiveCountEDMA ( SPDIF_Type * *base*, spdif_edma_handle_t * *handle*, size_t * *count* )

Parameters

| | |
|---|---|
| *base* | SPDIF base pointer |
| *handle* | SPDIF eDMA handle pointer. |
| *count* | Bytes count received by SPDIF. |

Return values

| | |
|---|---|
| *kStatus_Success* | Succeed get the transfer count. |
| *kStatus_NoTransferInProgress* | There is no non-blocking transaction in progress. |

# Chapter 42
# SRC: System Reset Controller Driver

## 42.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Reset Controller (SRC) module.

The System Reset Controller (SRC) controls the reset and boot operation of the SoC. It is responsible for the generation of all reset signals and boot decoding. The reset controller determines the source and the type of reset, such as POR, WARM, COLD, and performs the necessary reset qualification and stretching sequences. Based on the type of reset, the reset logic generates the reset sequence for the entire IC.

## Enumerations

- enum _src_reset_status_flags {
  kSRC_TemperatureSensorResetFlag = SRC_SRSR_TSR_MASK,
  kSRC_Wdog3ResetFlag = SRC_SRSR_WDOG3_RST_B_MASK,
  kSRC_JTAGSoftwareResetFlag = SRC_SRSR_SJC_MASK,
  kSRC_JTAGGeneratedResetFlag = SRC_SRSR_JTAG_MASK,
  kSRC_WatchdogResetFlag = SRC_SRSR_WDOG_MASK,
  kSRC_IppUserResetFlag = SRC_SRSR_IPP_USER_RESET_B_MASK,
  kSRC_CsuResetFlag = SRC_SRSR_CSU_RESET_B_MASK,
  kSRC_LockupSysResetFlag,
  kSRC_IppResetPinFlag = SRC_SRSR_IPP_RESET_B_MASK }
  *SRC reset status flags.*
- enum src_warm_reset_bypass_count_t {
  kSRC_WarmResetWaitAlways = 0U,
  kSRC_WarmResetWaitClk16 = 1U,
  kSRC_WarmResetWaitClk32 = 2U,
  kSRC_WarmResetWaitClk64 = 3U }
  *Selection of WARM reset bypass count.*

## Functions

- static void SRC_EnableWDOG3Reset (SRC_Type *base, bool enable)
  *Enable the WDOG3 reset.*
- static void SRC_EnableCoreDebugResetAfterPowerGate (SRC_Type *base, bool enable)
  *Debug reset would be asserted after power gating event.*
- static void SRC_DoSoftwareResetARMCore0 (SRC_Type *base)
  *Do software reset the ARM core0 only.*
- static bool SRC_GetSoftwareResetARMCore0Done (SRC_Type *base)
  *Check if the software for ARM core0 is done.*
- static void SRC_EnableWDOGReset (SRC_Type *base, bool enable)
  *Enable the WDOG Reset in SRC.*
- static void SRC_EnableLockupReset (SRC_Type *base, bool enable)

*Enable the lockup reset.*
- static uint32_t SRC_GetBootModeWord1 (SRC_Type *base)
  *Get the boot mode register 1 value.*
- static uint32_t SRC_GetBootModeWord2 (SRC_Type *base)
  *Get the boot mode register 2 value.*
- static uint32_t SRC_GetResetStatusFlags (SRC_Type *base)
  *Get the status flags of SRC.*
- void SRC_ClearResetStatusFlags (SRC_Type *base, uint32_t flags)
  *Clear the status flags of SRC.*
- static void SRC_SetGeneralPurposeRegister (SRC_Type *base, uint32_t index, uint32_t value)
  *Set value to general purpose registers.*
- static uint32_t SRC_GetGeneralPurposeRegister (SRC_Type *base, uint32_t index)
  *Get the value from general purpose registers.*

## Driver version

- #define FSL_SRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *SRC driver version 2.0.1.*

## 42.2 Macro Definition Documentation

### 42.2.1 #define FSL_SRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

## 42.3 Enumeration Type Documentation

### 42.3.1 enum _src_reset_status_flags

Enumerator

***kSRC_TemperatureSensorResetFlag*** Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt needs to be served before this bit can be cleaned.

***kSRC_Wdog3ResetFlag*** IC Watchdog3 Time-out reset. Indicates whether the reset was the result of the watchdog3 time-out event.

***kSRC_JTAGSoftwareResetFlag*** Indicates whether the reset was the result of setting SJC_GPCCR bit 31.

***kSRC_JTAGGeneratedResetFlag*** Indicates a reset has been caused by JTAG selection of certain IR codes: EXTEST or HIGHZ.

***kSRC_WatchdogResetFlag*** Indicates a reset has been caused by the watchdog timer timing out. This reset source can be blocked by disabling the watchdog.

***kSRC_IppUserResetFlag*** Indicates whether the reset was the result of the ipp_user_reset_b qualified reset.

***kSRC_CsuResetFlag*** Indicates whether the reset was the result of the csu_reset_b input.

***kSRC_LockupSysResetFlag*** Indicates a reset has been caused by CPU lockup or software setting of SYSRESETREQ bit in Application Interrupt and Reset Control Register of the ARM core.

***kSRC_IppResetPinFlag*** Indicates whether reset was the result of ipp_reset_b pin (Power-up sequence).

### 42.3.2 enum src_warm_reset_bypass_count_t

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Enumerator

***kSRC_WarmResetWaitAlways*** System will wait until MMDC acknowledge is asserted.
***kSRC_WarmResetWaitClk16*** Wait 16 32KHz clock cycles before switching the reset.
***kSRC_WarmResetWaitClk32*** Wait 32 32KHz clock cycles before switching the reset.
***kSRC_WarmResetWaitClk64*** Wait 64 32KHz clock cycles before switching the reset.

## 42.4 Function Documentation

### 42.4.1 static void SRC_EnableWDOG3Reset ( SRC_Type * *base,* bool *enable* ) [inline], [static]

The WDOG3 reset is enabled by default.

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |
| *enable* | Enable the reset or not. |

### 42.4.2 static void SRC_EnableCoreDebugResetAfterPowerGate ( SRC_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |
| *enable* | Enable the reset or not. |

### 42.4.3 static void SRC_DoSoftwareResetARMCore0 ( SRC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |

### 42.4.4 static bool SRC_GetSoftwareResetARMCore0Done ( SRC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |

Returns

If the reset is done.

### 42.4.5 static void SRC_EnableWDOGReset ( SRC_Type * *base,* bool *enable* ) [inline], [static]

WDOG Reset is enabled in SRC by default. If the WDOG event to SRC is masked, it would not create a reset to the chip. During the time the WDOG event is masked, when the WDOG event flag is asserted, it would remain asserted regardless of servicing the WDOG module. The only way to clear that bit is the hardware reset.

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |
| *enable* | Enable the reset or not. |

### 42.4.6 static void SRC_EnableLockupReset ( SRC_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |

| | |
|---|---|
| *enable* | Enable the reset or not. |

### 42.4.7 static uint32_t SRC_GetBootModeWord1 ( SRC_Type * *base* ) [inline], [static]

The Boot Mode register contains bits that reflect the status of BOOT_CFGx pins of the chip. See to chip-specific document for detail information about value.

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |

Returns

status of BOOT_CFGx pins of the chip.

### 42.4.8 static uint32_t SRC_GetBootModeWord2 ( SRC_Type * *base* ) [inline], [static]

The Boot Mode register contains bits that reflect the status of BOOT_MODEx Pins and fuse values that controls boot of the chip. See to chip-specific document for detail information about value.

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |

Returns

status of BOOT_MODEx Pins and fuse values that controls boot of the chip.

### 42.4.9 static uint32_t SRC_GetResetStatusFlags ( SRC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |

Returns

Mask value of status flags, see to _src_reset_status_flags.

### 42.4.10 void SRC_ClearResetStatusFlags ( SRC_Type * *base*, uint32_t *flags* )

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |
| *Mask* | value of status flags to be cleared, see to _src_reset_status_flags. |

### 42.4.11 static void SRC_SetGeneralPurposeRegister ( SRC_Type * *base*, uint32_t *index*, uint32_t *value* ) [inline], [static]

General purpose registers (GPRx) would hold the value during reset process. Wakeup function could be kept in these register. For example, the GPR1 holds the entry function for waking-up from Partial SLEEP mode while the GPR2 holds the argument. Other GPRx register would store the arbitray values.

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |
| *index* | The index of GPRx register array. Note index 0 reponses the GPR1 register. |
| *value* | Setting value for GPRx register. |

### 42.4.12 static uint32_t SRC_GetGeneralPurposeRegister ( SRC_Type * *base*, uint32_t *index* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | SRC peripheral base address. |

| | |
|---|---|
| *index* | The index of GPRx register array. Note index 0 reponses the GPR1 register. |

Returns

The setting value for GPRx register.

# Chapter 43
# TRNG: True Random Number Generator

## 43.1 Overview

The MCUXpresso SDK provides a peripheral driver for the True Random Number Generator (TRNG) module of MCUXpresso SDK devices.

The True Random Number Generator is a hardware accelerator module that generates a 512-bit entropy as needed by an entropy consuming module or by other post processing functions. A typical entropy consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the TRNG output as its entropy seed. The entropy generated by a TRNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

## 43.2 TRNG Initialization

1. Define the TRNG user configuration structure. Use TRNG_InitUserConfigDefault() function to set it to default TRNG configuration values.
2. Initialize the TRNG module, call the TRNG_Init() function, and pass the user configuration structure. This function automatically enables the TRNG module and its clock. After that, the TRNG is enabled and the entropy generation starts working.
3. To disable the TRNG module, call the TRNG_Deinit() function.

## 43.3 Get random data from TRNG

1. TRNG_GetRandomData() function gets random data from the TRNG module.

This example code shows how to initialize and get random data from the TRNG driver.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/trng

## Data Structures

- struct trng_statistical_check_limit_t
  *Data structure for definition of statistical check limits.* *More...*
- struct trng_config_t
  *Data structure for the TRNG initialization.* *More...*

## Enumerations

- enum trng_sample_mode_t {
  kTRNG_SampleModeVonNeumann = 0U,
  kTRNG_SampleModeRaw = 1U,
  kTRNG_SampleModeVonNeumannRaw }
  *TRNG sample mode.*

- enum trng_clock_mode_t {
  kTRNG_ClockModeRingOscillator = 0U,
  kTRNG_ClockModeSystem = 1U }
  *TRNG clock mode.*
- enum trng_ring_osc_div_t {
  kTRNG_RingOscDiv0 = 0U,
  kTRNG_RingOscDiv2 = 1U,
  kTRNG_RingOscDiv4 = 2U,
  kTRNG_RingOscDiv8 = 3U }
  *TRNG ring oscillator divide.*

## Functions

- status_t TRNG_GetDefaultConfig (trng_config_t *userConfig)
  *Initializes the user configuration structure to default values.*
- status_t TRNG_Init (TRNG_Type *base, const trng_config_t *userConfig)
  *Initializes the TRNG.*
- void TRNG_Deinit (TRNG_Type *base)
  *Shuts down the TRNG.*
- status_t TRNG_GetRandomData (TRNG_Type *base, void *data, size_t dataSize)
  *Gets random data.*

## Driver version

- #define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  *TRNG driver version 2.0.4.*

## 43.4 Data Structure Documentation

### 43.4.1 struct trng_statistical_check_limit_t

Used by trng_config_t.

### Data Fields

- uint32_t maximum
  *Maximum limit.*
- uint32_t minimum
  *Minimum limit.*

#### 43.4.1.0.0.61 Field Documentation

##### 43.4.1.0.0.61.1 uint32_t trng_statistical_check_limit_t::maximum

##### 43.4.1.0.0.61.2 uint32_t trng_statistical_check_limit_t::minimum

### 43.4.2 struct trng_config_t

This structure initializes the TRNG by calling the TRNG_Init() function. It contains all TRNG configurations.

## Data Fields

- bool lock
  *Disable programmability of TRNG registers.*
- trng_clock_mode_t clockMode
  *Clock mode used to operate TRNG.*
- trng_ring_osc_div_t ringOscDiv
  *Ring oscillator divide used by TRNG.*
- trng_sample_mode_t sampleMode
  *Sample mode of the TRNG ring oscillator.*
- uint16_t entropyDelay
  *Entropy Delay.*
- uint16_t sampleSize
  *Sample Size.*
- uint16_t sparseBitLimit
  *Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated.*
- uint8_t retryCount
  *Retry count.*
- uint8_t longRunMaxLimit
  *Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.*
- trng_statistical_check_limit_t monobitLimit
  *Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.*
- trng_statistical_check_limit_t runBit1Limit
  *Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.*
- trng_statistical_check_limit_t runBit2Limit
  *Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.*
- trng_statistical_check_limit_t runBit3Limit
  *Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.*
- trng_statistical_check_limit_t runBit4Limit
  *Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.*
- trng_statistical_check_limit_t runBit5Limit

*Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.*

- trng_statistical_check_limit_t runBit6PlusLimit
  *Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.*
- trng_statistical_check_limit_t pokerLimit
  *Maximum and minimum limits for statistical check of "Poker Test".*
- trng_statistical_check_limit_t frequencyCountLimit
  *Maximum and minimum limits for statistical check of entropy sample frequency count.*

#### 43.4.2.0.0.62 Field Documentation

##### 43.4.2.0.0.62.1 bool trng_config_t::lock

##### 43.4.2.0.0.62.2 trng_clock_mode_t trng_config_t::clockMode

##### 43.4.2.0.0.62.3 trng_ring_osc_div_t trng_config_t::ringOscDiv

##### 43.4.2.0.0.62.4 trng_sample_mode_t trng_config_t::sampleMode

##### 43.4.2.0.0.62.5 uint16_t trng_config_t::entropyDelay

Defines the length (in system clocks) of each Entropy sample taken.

##### 43.4.2.0.0.62.6 uint16_t trng_config_t::sampleSize

Defines the total number of Entropy samples that will be taken during Entropy generation.

##### 43.4.2.0.0.62.7 uint16_t trng_config_t::sparseBitLimit

This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

##### 43.4.2.0.0.62.8 uint8_t trng_config_t::retryCount

It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

#### 43.4.2.0.0.62.9 uint8_t trng_config_t::longRunMaxLimit

#### 43.4.2.0.0.62.10 trng_statistical_check_limit_t trng_config_t::monobitLimit

#### 43.4.2.0.0.62.11 trng_statistical_check_limit_t trng_config_t::runBit1Limit

#### 43.4.2.0.0.62.12 trng_statistical_check_limit_t trng_config_t::runBit2Limit

#### 43.4.2.0.0.62.13 trng_statistical_check_limit_t trng_config_t::runBit3Limit

#### 43.4.2.0.0.62.14 trng_statistical_check_limit_t trng_config_t::runBit4Limit

#### 43.4.2.0.0.62.15 trng_statistical_check_limit_t trng_config_t::runBit5Limit

#### 43.4.2.0.0.62.16 trng_statistical_check_limit_t trng_config_t::runBit6PlusLimit

#### 43.4.2.0.0.62.17 trng_statistical_check_limit_t trng_config_t::pokerLimit

#### 43.4.2.0.0.62.18 trng_statistical_check_limit_t trng_config_t::frequencyCountLimit

## 43.5 Macro Definition Documentation

### 43.5.1 #define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

Current version: 2.0.4

Change log:

- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG_Init to restart entropy generation
- Version 2.0.2
  - fix MISRA issues
- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

## 43.6 Enumeration Type Documentation

### 43.6.1 enum trng_sample_mode_t

Used by trng_config_t.

Enumerator

***kTRNG_SampleModeVonNeumann*** Use von Neumann data in both Entropy shifter and Statistical Checker.

***kTRNG_SampleModeRaw*** Use raw data into both Entropy shifter and Statistical Checker.

***kTRNG_SampleModeVonNeumannRaw*** Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

### 43.6.2 enum trng_clock_mode_t

Used by trng_config_t.

Enumerator

***kTRNG_ClockModeRingOscillator*** Ring oscillator is used to operate the TRNG (default).
***kTRNG_ClockModeSystem*** System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

### 43.6.3 enum trng_ring_osc_div_t

Used by trng_config_t.

Enumerator

***kTRNG_RingOscDiv0*** Ring oscillator with no divide.
***kTRNG_RingOscDiv2*** Ring oscillator divided-by-2.
***kTRNG_RingOscDiv4*** Ring oscillator divided-by-4.
***kTRNG_RingOscDiv8*** Ring oscillator divided-by-8.

## 43.7 Function Documentation

### 43.7.1 status_t TRNG_GetDefaultConfig ( trng_config_t * *userConfig* )

This function initializes the configuration structure to default values. The default values are as follows.

```
*     user_config->lock = 0;
*     user_config->clockMode = kTRNG_ClockModeRingOscillator;
*     user_config->ringOscDiv = kTRNG_RingOscDiv0;  Or  to other kTRNG_RingOscDiv[2|8]
      depending on the platform.
*     user_config->sampleMode = kTRNG_SampleModeRaw;
*     user_config->entropyDelay = 3200;
*     user_config->sampleSize = 2500;
*     user_config->sparseBitLimit = TRNG_USER_CONFIG_DEFAULT_SPARSE_BIT_LIMIT;
*     user_config->retryCount = 63;
*     user_config->longRunMaxLimit = 34;
*     user_config->monobitLimit.maximum = 1384;
*     user_config->monobitLimit.minimum = 1116;
*     user_config->runBit1Limit.maximum = 405;
*     user_config->runBit1Limit.minimum = 227;
*     user_config->runBit2Limit.maximum = 220;
*     user_config->runBit2Limit.minimum = 98;
*     user_config->runBit3Limit.maximum = 125;
*     user_config->runBit3Limit.minimum = 37;
*     user_config->runBit4Limit.maximum = 75;
```

```
*     user_config->runBit4Limit.minimum = 11;
*     user_config->runBit5Limit.maximum = 47;
*     user_config->runBit5Limit.minimum = 1;
*     user_config->runBit6PlusLimit.maximum = 47;
*     user_config->runBit6PlusLimit.minimum = 1;
*     user_config->pokerLimit.maximum = 26912;
*     user_config->pokerLimit.minimum = 24445;
*     user_config->frequencyCountLimit.maximum = 25600;
*     user_config->frequencyCountLimit.minimum = 1600;
*
```

Parameters

| | |
|---|---|
| *user_config* | User configuration structure. |

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

### 43.7.2 status_t TRNG_Init ( TRNG_Type * *base,* const trng_config_t * *userConfig* )

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

| | |
|---|---|
| *base* | TRNG base address |
| *userConfig* | Pointer to the initialization configuration structure. |

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

### 43.7.3 void TRNG_Deinit ( TRNG_Type * *base* )

This function shuts down the TRNG.

Parameters

| | |
|---|---|
| *base* | TRNG base address. |

### 43.7.4 status_t TRNG_GetRandomData ( TRNG_Type * *base,* void * *data,* size_t *dataSize* )

This function gets random data from the TRNG.

Parameters

| | |
|---|---|
| *base* | TRNG base address. |
| *data* | Pointer address used to store random data. |
| *dataSize* | Size of the buffer pointed by the data parameter. |

Returns

random data

# Chapter 44
# TSC: Touch Screen Controller Driver

## 44.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Touch Screen Controller(TSC) module of MCUXpresso SDK devices.

## 44.2 Typical use case

### 44.2.1 4-wire Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/fsl_-tsc

### 44.2.2 4-wire Interrupt Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/fsl_-tsc

## Data Structures

- struct tsc_config_t
  @ *Controller configuration.* *More...*

## Macros

- #define FSL_TSC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  *TSC driver version.*

## Enumerations

- enum tsc_detection_mode_t {
  kTSC_Detection4WireMode = 0U,
  kTSC_Detection5WireMode = 1U }
  @ *Controller detection mode.*
- enum tsc_corrdinate_value_selection_t {
  kTSC_XCoordinateValueSelection = 0U,
  kTSC_YCoordinateValueSelection = 1U }
  @ *Coordinate value mask.*

**Typical use case**

- enum _tsc_interrupt_signal_mask {
  kTSC_IdleSoftwareSignalEnable = TSC_INT_SIG_EN_IDLE_SW_SIG_EN_MASK,
  kTSC_ValidSignalEnable = TSC_INT_SIG_EN_VALID_SIG_EN_MASK,
  kTSC_DetectSignalEnable,
  kTSC_MeasureSignalEnable = TSC_INT_SIG_EN_MEASURE_SIG_EN_MASK }
  @ *Interrupt signal enable/disable mask.*
- enum _tsc_interrupt_mask {
  kTSC_IdleSoftwareInterruptEnable,
  kTSC_DetectInterruptEnable,
  kTSC_MeasureInterruptEnable = TSC_INT_EN_MEASURE_INT_EN_MASK }
  @ *Interrupt enable/disable mask.*
- enum _tsc_interrupt_status_flag_mask {
  kTSC_IdleSoftwareFlag,
  kTSC_ValidSignalFlag,
  kTSC_DetectSignalFlag = TSC_INT_STATUS_DETECT_MASK,
  kTSC_MeasureSignalFlag }
  @ *Interrupt Status flag mask.*
- enum _tsc_adc_status_flag_mask {
  kTSC_ADCCOCOSignalFlag,
  kTSC_ADCConversionValueFlag = TSC_DEBUG_MODE_ADC_CONV_VALUE_MASK }
  @ *ADC status flag mask.*
- enum _tsc_status_flag_mask {
  kTSC_IntermediateStateFlag = TSC_DEBUG_MODE2_INTERMEDIATE_MASK,
  kTSC_DetectFiveWireFlag = TSC_DEBUG_MODE2_DETECT_FIVE_WIRE_MASK,
  kTSC_DetectFourWireFlag = TSC_DEBUG_MODE2_DETECT_FOUR_WIRE_MASK,
  kTSC_GlitchThresholdFlag = TSC_DEBUG_MODE2_DE_GLITCH_MASK,
  kTSC_StateMachineFlag }
  @ *TSC status flag mask.*
- enum tsc_state_machine_t {
  kTSC_IdleState = 0U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT,
  kTSC_1stPreChargeState = 1U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT,
  kTSC_1stDetectState = 2U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT,
  kTSC_XMeasureState = 3U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT,
  kTSC_YMeasureState = 4U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT,
  kTSC_2ndPreChargeState = 5U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT,
  kTSC_2ndDetectState = 6U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT }
  *TSC state machine.*
- enum tsc_glitch_threshold_t {
  kTSC_glitchThresholdALT0 = 0U << TSC_DEBUG_MODE2_DE_GLITCH_SHIFT,
  kTSC_glitchThresholdALT1 = 1U << TSC_DEBUG_MODE2_DE_GLITCH_SHIFT,
  kTSC_glitchThresholdALT2 = 2U << TSC_DEBUG_MODE2_DE_GLITCH_SHIFT,
  kTSC_glitchThresholdALT3 }
  *TSC glitch threshold.*
- enum tsc_trigger_signal_t {

kTSC_TriggerToChannel0 = 1U << 0U,
kTSC_TriggerToChannel1 = 1U << 1U,
kTSC_TriggerToChannel2 = 1U << 2U,
kTSC_TriggerToChannel3 = 1U << 3U,
kTSC_TriggerToChannel4 = 1U << 4U }
@ *Hardware trigger select signal, select which ADC channel to start conversion.*

- enum tsc_port_source_t {
  kTSC_WiperPortSource = 0U,
  kTSC_YnlrPortSource = 1U,
  kTSC_YpllPortSource = 2U,
  kTSC_XnurPortSource = 3U,
  kTSC_XpulPortSource = 4U }
  @ *TSC controller ports.*
- enum tsc_port_mode_t {
  kTSC_PortOffMode = 0U,
  kTSC_Port200k_PullUpMode = 1U << 2U,
  kTSC_PortPullUpMode = 1U << 1U,
  kTSC_PortPullDownMode = 1U << 0U }
  @ *TSC port mode.*

## Functions

- void TSC_Init (TSC_Type *base, const tsc_config_t *config)
  *Initialize the TSC module.*
- void TSC_Deinit (TSC_Type *base)
  *De-initializes the TSC module.*

## Variables

- bool tsc_config_t::enableAutoMeasure
  *Enable the auto-measure.*
- uint32_t tsc_config_t::measureDelayTime
  *Set delay time(0U~0xFFFFFFU) to even potential distribution ready.It is a preparation for measure stage.*
- uint32_t tsc_config_t::prechargeTime
  *Set pre-charge time(1U~0xFFFFFFFFU) to make the upper layer of screen to charge to positive high.*
- tsc_detection_mode_t tsc_config_t::detectionMode
  *Select the detection mode.*

# 44.3 Data Structure Documentation

## 44.3.1 struct tsc_config_t

## Data Fields

- bool enableAutoMeasure
  *Enable the auto-measure.*
- uint32_t measureDelayTime
  *Set delay time(0U~0xFFFFFFU) to even potential distribution ready.It is a preparation for measure stage.*

- uint32_t prechargeTime
  *Set pre-charge time(1U∼0xFFFFFFFFU) to make the upper layer of screen to charge to positive high.*
- tsc_detection_mode_t detectionMode
  *Select the detection mode.*

## 44.4 Macro Definition Documentation

### 44.4.1 #define FSL_TSC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Version 2.0.2.

## 44.5 Enumeration Type Documentation

### 44.5.1 enum tsc_detection_mode_t

Enumerator

***kTSC_Detection4WireMode*** 4-Wire Detection Mode.
***kTSC_Detection5WireMode*** 5-Wire Detection Mode.

### 44.5.2 enum tsc_corrdinate_value_selection_t

Enumerator

***kTSC_XCoordinateValueSelection*** X coordinate value is selected.
***kTSC_YCoordinateValueSelection*** Y coordinate value is selected.

### 44.5.3 enum _tsc_interrupt_signal_mask

Enumerator

***kTSC_IdleSoftwareSignalEnable*** Enable the interrupt signal when the controller has return to idle status. The signal is only valid after using TSC_ReturnToIdleStatus API.
***kTSC_ValidSignalEnable*** Enable the interrupt signal when controller receives a detect signal after measurement.
***kTSC_DetectSignalEnable*** Enable the interrupt signal when controller receives a detect signal.
***kTSC_MeasureSignalEnable*** Enable the interrupt signal after the touch detection which follows measurement.

### 44.5.4 enum _tsc_interrupt_mask

Enumerator

***kTSC_IdleSoftwareInterruptEnable*** Enable the interrupt when the controller has return to idle status. The interrupt is only valid after using TSC_ReturnToIdleStatus API.

***kTSC_DetectInterruptEnable*** Enable the interrupt when controller receive a detect signal.

***kTSC_MeasureInterruptEnable*** Enable the interrupt after the touch detection which follows measurement.

### 44.5.5 enum _tsc_interrupt_status_flag_mask

Enumerator

***kTSC_IdleSoftwareFlag*** This flag is set if the controller has return to idle status. The flag is only valid after using TSC_ReturnToIdleStatus API.

***kTSC_ValidSignalFlag*** This flag is set if controller receives a detect signal after measurement.

***kTSC_DetectSignalFlag*** This flag is set if controller receives a detect signal.

***kTSC_MeasureSignalFlag*** This flag is set after the touch detection which follows measurement. Note: Valid signal falg will be cleared along with measure signal flag.

### 44.5.6 enum _tsc_adc_status_flag_mask

Enumerator

***kTSC_ADCCOCOSignalFlag*** This signal is generated by ADC when a conversion is completed.

***kTSC_ADCConversionValueFlag*** This signal is generated by ADC and indicates the result of an ADC conversion.

### 44.5.7 enum _tsc_status_flag_mask

Enumerator

***kTSC_IntermediateStateFlag*** This flag is set if TSC is in intermediate state, between two state machine states.

***kTSC_DetectFiveWireFlag*** This flag is set if TSC receives a 5-wire detect signal. It is only valid when the TSC in detect state and DETECT_ENABLE_FIVE_WIRE bit is set.

***kTSC_DetectFourWireFlag*** This flag is set if TSC receives a 4-wire detect signal. It is only valid when the TSC in detect state and DETECT_ENABLE_FOUR_WIRE bit is set.

***kTSC_GlitchThresholdFlag*** This field indicates glitch threshold.The threshold is defined by number of clock cycles. See "tsc_glitch_threshold_t". If value = 00, Normal function: 0x1fff ipg clock cycles, Low power mode: 0x9 low power clock cycles. If value = 01, Normal function: 0xfff ipg

clock cycles, Low power mode: :0x7 low power clock cycles. If value = 10, Normal function: 0x7ff ipg clock cycles, Low power mode:0x5 low power clock cycles. If value = 11, Normal function: 0x3 ipg clock cycles, Low power mode:0x3 low power clock cycles.

***kTSC_StateMachineFlag*** This field indicates the state of TSC. See "tsc_state_machine_t"; if value = 000, Controller is in idle state. if value = 001, Controller is in 1st-Pre-charge state. if value = 010, Controller is in 1st-detect state. if value = 011, Controller is in x-measure state. if value = 100, Controller is in y-measure state. if value = 101, Controller is in 2nd-Pre-charge state. if value = 110, Controller is in 2nd-detect state.

### 44.5.8 enum tsc_state_machine_t

These seven states are TSC complete workflow.

Enumerator

***kTSC_IdleState*** Controller is in idle state.
***kTSC_1stPreChargeState*** Controller is in 1st-Pre-charge state.
***kTSC_1stDetectState*** Controller is in 1st-detect state.
***kTSC_XMeasureState*** Controller is in x-measure state.
***kTSC_YMeasureState*** Controller is in y-measure state.
***kTSC_2ndPreChargeState*** Controller is in 2nd-Pre-charge state.
***kTSC_2ndDetectState*** Controller is in 2nd-detect state.

### 44.5.9 enum tsc_glitch_threshold_t

Enumerator

***kTSC_glitchThresholdALT0*** Normal function: 0x1fff ipg clock cycles, Low power mode: 0x9 low power clock cycles.
***kTSC_glitchThresholdALT1*** Normal function: 0xfff ipg clock cycles, Low power mode: :0x7 low power clock cycles.
***kTSC_glitchThresholdALT2*** Normal function: 0x7ff ipg clock cycles, Low power mode: :0x5 low power clock cycles.
***kTSC_glitchThresholdALT3*** Normal function: 0x3 ipg clock cycles, Low power mode: :0x3 low power clock cycles.

### 44.5.10 enum tsc_trigger_signal_t

Enumerator

***kTSC_TriggerToChannel0*** Trigger to ADC channel0. ADC_HC0 register will be used to conversion.

***kTSC_TriggerToChannel1*** Trigger to ADC channel1. ADC_HC1 register will be used to conversion.
***kTSC_TriggerToChannel2*** Trigger to ADC channel2. ADC_HC2 register will be used to conversion.
***kTSC_TriggerToChannel3*** Trigger to ADC channel3. ADC_HC3 register will be used to conversion.
***kTSC_TriggerToChannel4*** Trigger to ADC channel4. ADC_HC4 register will be used to conversion.

### 44.5.11 enum tsc_port_source_t

Enumerator

***kTSC_WiperPortSource*** TSC controller wiper port.
***kTSC_YnlrPortSource*** TSC controller ynlr port.
***kTSC_YpllPortSource*** TSC controller ypll port.
***kTSC_XnurPortSource*** TSC controller xnur port.
***kTSC_XpulPortSource*** TSC controller xpul port.

### 44.5.12 enum tsc_port_mode_t

Enumerator

***kTSC_PortOffMode*** Disable pull up/down mode.
***kTSC_Port200k_PullUpMode*** 200k-pull up mode.
***kTSC_PortPullUpMode*** Pull up mode.
***kTSC_PortPullDownMode*** Pull down mode.

## 44.6 Function Documentation

### 44.6.1 void TSC_Init ( TSC_Type * *base,* const tsc_config_t * *config* )

Parameters

| | |
|---|---|
| *base* | TSC peripheral base address. |
| *config* | Pointer to "tsc_config_t" structure. |

### 44.6.2 void TSC_Deinit ( TSC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | TSC peripheral base address. |

## 44.7 Variable Documentation

### 44.7.1 bool tsc_config_t::enableAutoMeasure

It indicates after detect touch, whether automatic start measurement

### 44.7.2 uint32_t tsc_config_t::measureDelayTime

If measure dalay time is too short, maybe it would have an undesired effect on measure value.

### 44.7.3 uint32_t tsc_config_t::prechargeTime

It is a preparation for detection stage. Pre-charge time must is greater than 0U, otherwise TSC could not work normally. If pre-charge dalay time is too short, maybe it would have an undesired effect on generation of valid signal(kTSC_ValidSignalFlag).

### 44.7.4 tsc_detection_mode_t tsc_config_t::detectionMode

See "tsc_detection_mode_t".

# Chapter 45
# USDHC: ultra Secured Digital Host Controller Driver

## 45.1 Overview

The MCUXpresso SDK provides a peripheral driver for the ultra Secured Digital Host Controller (USDHC) module of MCUXpresso SDK/i.MX devices.

## 45.2 Typical use case

### 45.2.1 USDHC Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/usdhc

## Data Structures

- struct usdhc_adma2_descriptor_t
  *Defines the ADMA2 descriptor structure. More...*
- struct usdhc_capability_t
  *USDHC capability information. More...*
- struct usdhc_boot_config_t
  *Data structure to configure the MMC boot feature. More...*
- struct usdhc_config_t
  *Data structure to initialize the USDHC. More...*
- struct usdhc_data_t
  *Card data descriptor. More...*
- struct usdhc_command_t
  *Card command descriptor. More...*
- struct usdhc_adma_config_t
  *ADMA configuration. More...*
- struct usdhc_transfer_t
  *Transfer state. More...*
- struct usdhc_transfer_callback_t
  *USDHC callback functions. More...*
- struct usdhc_handle_t
  *USDHC handle. More...*
- struct usdhc_host_t
  *USDHC host descriptor. More...*

## Macros

- #define USDHC_MAX_BLOCK_COUNT (USDHC_BLK_ATT_BLKCNT_MASK >> USDHC_BLK_ATT_BLKCNT_SHIFT)
  *Maximum block count can be set one time.*
- #define USDHC_ADMA1_ADDRESS_ALIGN (4096U)
  *The alignment size for ADDRESS filed in ADMA1's descriptor.*

**Typical use case**

- #define USDHC_ADMA1_LENGTH_ALIGN (4096U)
  *The alignment size for LENGTH field in ADMA1's descriptor.*
- #define USDHC_ADMA2_ADDRESS_ALIGN (4U)
  *The alignment size for ADDRESS field in ADMA2's descriptor.*
- #define USDHC_ADMA2_LENGTH_ALIGN (4U)
  *The alignment size for LENGTH filed in ADMA2's descriptor.*
- #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (12U)
  *The bit shift for ADDRESS filed in ADMA1's descriptor.*
- #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (0xFFFFFU)
  *The bit mask for ADDRESS field in ADMA1's descriptor.*
- #define USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (12U)
  *The bit shift for LENGTH filed in ADMA1's descriptor.*
- #define USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (0xFFFFU)
  *The mask for LENGTH field in ADMA1's descriptor.*
- #define USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (USDHC_ADMA1_-DESCRIPTOR_LENGTH_MASK + 1U - 4096U)
  *The maximum value of LENGTH filed in ADMA1's descriptor Since the ADMA1 support max transfer size is 65535 which is not divisible by 4096, so to make sure a large data load transfer(>64KB) continuously(require the data address should be alwawys align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.*
- #define USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (16U)
  *The bit shift for LENGTH field in ADMA2's descriptor.*
- #define USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (0xFFFFU)
  *The bit mask for LENGTH field in ADMA2's descriptor.*
- #define USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (USDHC_ADMA2_-DESCRIPTOR_LENGTH_MASK - 3U)
  *The maximum value of LENGTH field in ADMA2's descriptor.*

## Typedefs

- typedef uint32_t usdhc_adma1_descriptor_t
  *Defines the adma1 descriptor structure.*
- typedef status_t(∗ usdhc_transfer_function_t )(USDHC_Type ∗base, usdhc_transfer_t ∗content)
  *USDHC transfer function.*

## Enumerations

- enum _usdhc_status {
  kStatus_USDHC_BusyTransferring = MAKE_STATUS(kStatusGroup_USDHC, 0U),
  kStatus_USDHC_PrepareAdmaDescriptorFailed = MAKE_STATUS(kStatusGroup_USDHC, 1U),
  kStatus_USDHC_SendCommandFailed = MAKE_STATUS(kStatusGroup_USDHC, 2U),
  kStatus_USDHC_TransferDataFailed = MAKE_STATUS(kStatusGroup_USDHC, 3U),
  kStatus_USDHC_DMADataAddrNotAlign = MAKE_STATUS(kStatusGroup_USDHC, 4U),
  kStatus_USDHC_ReTuningRequest = MAKE_STATUS(kStatusGroup_USDHC, 5U),
  kStatus_USDHC_TuningError = MAKE_STATUS(kStatusGroup_USDHC, 6U),
  kStatus_USDHC_NotSupport = MAKE_STATUS(kStatusGroup_USDHC, 7U) }
  *USDHC status.*
- enum _usdhc_capability_flag {

kUSDHC_SupportAdmaFlag = USDHC_HOST_CTRL_CAP_ADMAS_MASK,
kUSDHC_SupportHighSpeedFlag = USDHC_HOST_CTRL_CAP_HSS_MASK,
kUSDHC_SupportDmaFlag = USDHC_HOST_CTRL_CAP_DMAS_MASK,
kUSDHC_SupportSuspendResumeFlag = USDHC_HOST_CTRL_CAP_SRS_MASK,
kUSDHC_SupportV330Flag = USDHC_HOST_CTRL_CAP_VS33_MASK,
kUSDHC_SupportV300Flag = USDHC_HOST_CTRL_CAP_VS30_MASK,
kUSDHC_SupportV180Flag = USDHC_HOST_CTRL_CAP_VS18_MASK,
kUSDHC_Support4BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 0U),
kUSDHC_Support8BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 1U),
kUSDHC_SupportDDR50Flag = USDHC_HOST_CTRL_CAP_DDR50_SUPPORT_MASK,
kUSDHC_SupportSDR104Flag = USDHC_HOST_CTRL_CAP_SDR104_SUPPORT_MASK,
kUSDHC_SupportSDR50Flag = USDHC_HOST_CTRL_CAP_SDR50_SUPPORT_MASK }
*Host controller capabilities flag mask.*

- enum _usdhc_wakeup_event {
kUSDHC_WakeupEventOnCardInt = USDHC_PROT_CTRL_WECINT_MASK,
kUSDHC_WakeupEventOnCardInsert = USDHC_PROT_CTRL_WECINS_MASK,
kUSDHC_WakeupEventOnCardRemove = USDHC_PROT_CTRL_WECRM_MASK,
kUSDHC_WakeupEventsAll }
*Wakeup event mask.*
- enum _usdhc_reset {
kUSDHC_ResetAll = USDHC_SYS_CTRL_RSTA_MASK,
kUSDHC_ResetCommand = USDHC_SYS_CTRL_RSTC_MASK,
kUSDHC_ResetData = USDHC_SYS_CTRL_RSTD_MASK,
kUSDHC_ResetTuning = USDHC_SYS_CTRL_RSTT_MASK,
kUSDHC_ResetsAll }
*Reset type mask.*
- enum _usdhc_transfer_flag {
kUSDHC_EnableDmaFlag = USDHC_MIX_CTRL_DMAEN_MASK,
kUSDHC_CommandTypeSuspendFlag = (USDHC_CMD_XFR_TYP_CMDTYP(1U)),
kUSDHC_CommandTypeResumeFlag = (USDHC_CMD_XFR_TYP_CMDTYP(2U)),
kUSDHC_CommandTypeAbortFlag = (USDHC_CMD_XFR_TYP_CMDTYP(3U)),
kUSDHC_EnableBlockCountFlag = USDHC_MIX_CTRL_BCEN_MASK,
kUSDHC_EnableAutoCommand12Flag = USDHC_MIX_CTRL_AC12EN_MASK,
kUSDHC_DataReadFlag = USDHC_MIX_CTRL_DTDSEL_MASK,
kUSDHC_MultipleBlockFlag = USDHC_MIX_CTRL_MSBSEL_MASK,
kUSDHC_EnableAutoCommand23Flag = USDHC_MIX_CTRL_AC23EN_MASK,
kUSDHC_ResponseLength136Flag = USDHC_CMD_XFR_TYP_RSPTYP(1U),
kUSDHC_ResponseLength48Flag = USDHC_CMD_XFR_TYP_RSPTYP(2U),
kUSDHC_ResponseLength48BusyFlag = USDHC_CMD_XFR_TYP_RSPTYP(3U),
kUSDHC_EnableCrcCheckFlag = USDHC_CMD_XFR_TYP_CCCEN_MASK,
kUSDHC_EnableIndexCheckFlag = USDHC_CMD_XFR_TYP_CICEN_MASK,
kUSDHC_DataPresentFlag = USDHC_CMD_XFR_TYP_DPSEL_MASK }
*Transfer flag mask.*
- enum _usdhc_present_status_flag {

kUSDHC_CommandInhibitFlag = USDHC_PRES_STATE_CIHB_MASK,
kUSDHC_DataInhibitFlag = USDHC_PRES_STATE_CDIHB_MASK,
kUSDHC_DataLineActiveFlag = USDHC_PRES_STATE_DLA_MASK,
kUSDHC_SdClockStableFlag = USDHC_PRES_STATE_SDSTB_MASK,
kUSDHC_WriteTransferActiveFlag = USDHC_PRES_STATE_WTA_MASK,
kUSDHC_ReadTransferActiveFlag = USDHC_PRES_STATE_RTA_MASK,
kUSDHC_BufferWriteEnableFlag = USDHC_PRES_STATE_BWEN_MASK,
kUSDHC_BufferReadEnableFlag = USDHC_PRES_STATE_BREN_MASK,
kUSDHC_ReTuningRequestFlag = USDHC_PRES_STATE_RTR_MASK,
kUSDHC_DelaySettingFinishedFlag = USDHC_PRES_STATE_TSCD_MASK,
kUSDHC_CardInsertedFlag = USDHC_PRES_STATE_CINST_MASK,
kUSDHC_CommandLineLevelFlag = USDHC_PRES_STATE_CLSL_MASK,
kUSDHC_Data0LineLevelFlag = 1U << USDHC_PRES_STATE_DLSL_SHIFT,
kUSDHC_Data1LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 1U),
kUSDHC_Data2LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 2U),
kUSDHC_Data3LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 3U),
kUSDHC_Data4LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 4U),
kUSDHC_Data5LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 5U),
kUSDHC_Data6LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 6U),
kUSDHC_Data7LineLevelFlag = (int)(1U << (USDHC_PRES_STATE_DLSL_SHIFT + 7U)) }

*Present status flag mask.*

- enum _usdhc_interrupt_status_flag {

kUSDHC_CommandCompleteFlag = USDHC_INT_STATUS_CC_MASK,
kUSDHC_DataCompleteFlag = USDHC_INT_STATUS_TC_MASK,
kUSDHC_BlockGapEventFlag = USDHC_INT_STATUS_BGE_MASK,
kUSDHC_DmaCompleteFlag = USDHC_INT_STATUS_DINT_MASK,
kUSDHC_BufferWriteReadyFlag = USDHC_INT_STATUS_BWR_MASK,
kUSDHC_BufferReadReadyFlag = USDHC_INT_STATUS_BRR_MASK,
kUSDHC_CardInsertionFlag = USDHC_INT_STATUS_CINS_MASK,
kUSDHC_CardRemovalFlag = USDHC_INT_STATUS_CRM_MASK,
kUSDHC_CardInterruptFlag = USDHC_INT_STATUS_CINT_MASK,
kUSDHC_ReTuningEventFlag = USDHC_INT_STATUS_RTE_MASK,
kUSDHC_TuningPassFlag = USDHC_INT_STATUS_TP_MASK,
kUSDHC_TuningErrorFlag = USDHC_INT_STATUS_TNE_MASK,
kUSDHC_CommandTimeoutFlag = USDHC_INT_STATUS_CTOE_MASK,
kUSDHC_CommandCrcErrorFlag = USDHC_INT_STATUS_CCE_MASK,
kUSDHC_CommandEndBitErrorFlag = USDHC_INT_STATUS_CEBE_MASK,
kUSDHC_CommandIndexErrorFlag = USDHC_INT_STATUS_CIE_MASK,
kUSDHC_DataTimeoutFlag = USDHC_INT_STATUS_DTOE_MASK,
kUSDHC_DataCrcErrorFlag = USDHC_INT_STATUS_DCE_MASK,
kUSDHC_DataEndBitErrorFlag = USDHC_INT_STATUS_DEBE_MASK,
kUSDHC_AutoCommand12ErrorFlag = USDHC_INT_STATUS_AC12E_MASK,
kUSDHC_DmaErrorFlag = USDHC_INT_STATUS_DMAE_MASK,
kUSDHC_CommandErrorFlag,
kUSDHC_DataErrorFlag,
kUSDHC_ErrorFlag = (kUSDHC_CommandErrorFlag | kUSDHC_DataErrorFlag | kUSDHC_-DmaErrorFlag),
kUSDHC_DataFlag,
kUSDHC_CommandFlag = (kUSDHC_CommandErrorFlag | kUSDHC_CommandCompleteFlag),
kUSDHC_CardDetectFlag = (kUSDHC_CardInsertionFlag | kUSDHC_CardRemovalFlag) ,
kUSDHC_AllInterruptFlags }

*Interrupt status flag mask.*

- enum _usdhc_auto_command12_error_status_flag {
kUSDHC_AutoCommand12NotExecutedFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12-NE_MASK,
kUSDHC_AutoCommand12TimeoutFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12TOE-_MASK,
kUSDHC_AutoCommand12EndBitErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12-EBE_MASK,
kUSDHC_AutoCommand12CrcErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12CE_-MASK,
kUSDHC_AutoCommand12IndexErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12IE-_MASK,
kUSDHC_AutoCommand12NotIssuedFlag = USDHC_AUTOCMD12_ERR_STATUS_CNIBA-C12E_MASK }

  *Auto CMD12 error status flag mask.*

- enum _usdhc_standard_tuning {

kUSDHC_ExecuteTuning = USDHC_AUTOCMD12_ERR_STATUS_EXECUTE_TUNING_MASK,
kUSDHC_TuningSampleClockSel = USDHC_AUTOCMD12_ERR_STATUS_SMP_CLK_SEL_MASK }
*standard tuning flag*

- enum _usdhc_adma_error_status_flag {
kUSDHC_AdmaLenghMismatchFlag = USDHC_ADMA_ERR_STATUS_ADMALME_MASK,
kUSDHC_AdmaDescriptorErrorFlag = USDHC_ADMA_ERR_STATUS_ADMADCE_MASK }
*ADMA error status flag mask.*
- enum _usdhc_adma_error_state {
kUSDHC_AdmaErrorStateStopDma,
kUSDHC_AdmaErrorStateFetchDescriptor,
kUSDHC_AdmaErrorStateChangeAddress = 0x02U,
kUSDHC_AdmaErrorStateTransferData,
kUSDHC_AdmaErrorStateInvalidLength = 0x04U,
kUSDHC_AdmaErrorStateInvalidDescriptor = 0x08U,
kUSDHC_AdmaErrorState }
*ADMA error state.*
- enum _usdhc_force_event {
kUSDHC_ForceEventAutoCommand12NotExecuted = USDHC_FORCE_EVENT_FEVTAC12NE_MASK,
kUSDHC_ForceEventAutoCommand12Timeout = USDHC_FORCE_EVENT_FEVTAC12TOE_MASK,
kUSDHC_ForceEventAutoCommand12CrcError = USDHC_FORCE_EVENT_FEVTAC12CE_MASK,
kUSDHC_ForceEventEndBitError = USDHC_FORCE_EVENT_FEVTAC12EBE_MASK,
kUSDHC_ForceEventAutoCommand12IndexError = USDHC_FORCE_EVENT_FEVTAC12IE_MASK,
kUSDHC_ForceEventAutoCommand12NotIssued = USDHC_FORCE_EVENT_FEVTCNIBAC12E_MASK,
kUSDHC_ForceEventCommandTimeout = USDHC_FORCE_EVENT_FEVTCTOE_MASK,
kUSDHC_ForceEventCommandCrcError = USDHC_FORCE_EVENT_FEVTCCE_MASK,
kUSDHC_ForceEventCommandEndBitError = USDHC_FORCE_EVENT_FEVTCEBE_MASK,
kUSDHC_ForceEventCommandIndexError = USDHC_FORCE_EVENT_FEVTCIE_MASK,
kUSDHC_ForceEventDataTimeout = USDHC_FORCE_EVENT_FEVTDTOE_MASK,
kUSDHC_ForceEventDataCrcError = USDHC_FORCE_EVENT_FEVTDCE_MASK,
kUSDHC_ForceEventDataEndBitError = USDHC_FORCE_EVENT_FEVTDEBE_MASK,
kUSDHC_ForceEventAutoCommand12Error = USDHC_FORCE_EVENT_FEVTAC12E_MASK,
kUSDHC_ForceEventCardInt = (int)USDHC_FORCE_EVENT_FEVTCINT_MASK,
kUSDHC_ForceEventDmaError = USDHC_FORCE_EVENT_FEVTDMAE_MASK,
kUSDHC_ForceEventTuningError = USDHC_FORCE_EVENT_FEVTTNE_MASK,
kUSDHC_ForceEventsAll }
*Force event bit position.*
- enum usdhc_data_bus_width_t {

kUSDHC_DataBusWidth1Bit = 0U,
kUSDHC_DataBusWidth4Bit = 1U,
kUSDHC_DataBusWidth8Bit = 2U }
*Data transfer width.*

- enum usdhc_endian_mode_t {
kUSDHC_EndianModeBig = 0U,
kUSDHC_EndianModeHalfWordBig = 1U,
kUSDHC_EndianModeLittle = 2U }
*Endian mode.*
- enum usdhc_dma_mode_t {
kUSDHC_DmaModeSimple = 0U,
kUSDHC_DmaModeAdma1 = 1U,
kUSDHC_DmaModeAdma2 = 2U,
kUSDHC_ExternalDMA = 3U }
*DMA mode.*
- enum _usdhc_sdio_control_flag {
kUSDHC_StopAtBlockGapFlag = USDHC_PROT_CTRL_SABGREQ_MASK,
kUSDHC_ReadWaitControlFlag = USDHC_PROT_CTRL_RWCTL_MASK,
kUSDHC_InterruptAtBlockGapFlag = USDHC_PROT_CTRL_IABG_MASK,
kUSDHC_ReadDoneNo8CLK = USDHC_PROT_CTRL_RD_DONE_NO_8CLK_MASK,
kUSDHC_ExactBlockNumberReadFlag = USDHC_PROT_CTRL_NON_EXACT_BLK_RD_MASK }
*SDIO control flag mask.*
- enum usdhc_boot_mode_t {
kUSDHC_BootModeNormal = 0U,
kUSDHC_BootModeAlternative = 1U }
*MMC card boot mode.*
- enum usdhc_card_command_type_t {
kCARD_CommandTypeNormal = 0U,
kCARD_CommandTypeSuspend = 1U,
kCARD_CommandTypeResume = 2U,
kCARD_CommandTypeAbort = 3U,
kCARD_CommandTypeEmpty = 4U }
*The command type.*
- enum usdhc_card_response_type_t {
kCARD_ResponseTypeNone = 0U,
kCARD_ResponseTypeR1 = 1U,
kCARD_ResponseTypeR1b = 2U,
kCARD_ResponseTypeR2 = 3U,
kCARD_ResponseTypeR3 = 4U,
kCARD_ResponseTypeR4 = 5U,
kCARD_ResponseTypeR5 = 6U,
kCARD_ResponseTypeR5b = 7U,
kCARD_ResponseTypeR6 = 8U,
kCARD_ResponseTypeR7 = 9U }
*The command response type.*

- enum _usdhc_adma1_descriptor_flag {
  kUSDHC_Adma1DescriptorValidFlag = (1U << 0U),
  kUSDHC_Adma1DescriptorEndFlag = (1U << 1U),
  kUSDHC_Adma1DescriptorInterrupFlag = (1U << 2U),
  kUSDHC_Adma1DescriptorActivity1Flag = (1U << 4U),
  kUSDHC_Adma1DescriptorActivity2Flag = (1U << 5U),
  kUSDHC_Adma1DescriptorTypeNop = (kUSDHC_Adma1DescriptorValidFlag),
  kUSDHC_Adma1DescriptorTypeTransfer,
  kUSDHC_Adma1DescriptorTypeLink,
  kUSDHC_Adma1DescriptorTypeSetLength }
  *The mask for the control/status field in ADMA1 descriptor.*
- enum _usdhc_adma2_descriptor_flag {
  kUSDHC_Adma2DescriptorValidFlag = (1U << 0U),
  kUSDHC_Adma2DescriptorEndFlag = (1U << 1U),
  kUSDHC_Adma2DescriptorInterruptFlag = (1U << 2U),
  kUSDHC_Adma2DescriptorActivity1Flag = (1U << 4U),
  kUSDHC_Adma2DescriptorActivity2Flag = (1U << 5U),
  kUSDHC_Adma2DescriptorTypeNop = (kUSDHC_Adma2DescriptorValidFlag),
  kUSDHC_Adma2DescriptorTypeReserved,
  kUSDHC_Adma2DescriptorTypeTransfer,
  kUSDHC_Adma2DescriptorTypeLink }
  *ADMA1 descriptor control and status mask.*
- enum _usdhc_adma_flag {
  kUSDHC_AdmaDescriptorSingleFlag,
  kUSDHC_AdmaDescriptorMultipleFlag = 1U }
  *ADMA descriptor configuration flag.*
- enum usdhc_burst_len_t {
  kUSDHC_EnBurstLenForINCR = 0x01U,
  kUSDHC_EnBurstLenForINCR4816 = 0x02U,
  kUSDHC_EnBurstLenForINCR4816WRAP = 0x04U }
  *dma transfer burst len config.*
- enum _usdhc_transfer_data_type {
  kUSDHC_TransferDataNormal = 0U,
  kUSDHC_TransferDataTuning = 1U,
  kUSDHC_TransferDataBoot = 2U,
  kUSDHC_TransferDataBootcontinous = 3U }
  *transfer data type definition.*

## Driver version

- #define FSL_USDHC_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 8U))
  *Driver version 2.2.8.*

## Initialization and deinitialization

- void USDHC_Init (USDHC_Type *base, const usdhc_config_t *config)
  *USDHC module initialization function.*

- void USDHC_Deinit (USDHC_Type *base)
  *Deinitializes the USDHC.*
- bool USDHC_Reset (USDHC_Type *base, uint32_t mask, uint32_t timeout)
  *Resets the USDHC.*

## DMA Control

- status_t USDHC_SetAdmaTableConfig (USDHC_Type *base, usdhc_adma_config_t *dmaConfig, usdhc_data_t *dataConfig, uint32_t flags)
  *Sets the DMA descriptor table configuration.*
- status_t USDHC_SetInternalDmaConfig (USDHC_Type *base, usdhc_adma_config_t *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)
  *Internal DMA configuration.*
- status_t USDHC_SetADMA2Descriptor (uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)
  *Sets the ADMA2 descriptor table configuration.*
- status_t USDHC_SetADMA1Descriptor (uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)
  *Sets the ADMA1 descriptor table configuration.*
- static void USDHC_EnableInternalDMA (USDHC_Type *base, bool enable)
  *enable internal DMA.*

## Interrupts

- static void USDHC_EnableInterruptStatus (USDHC_Type *base, uint32_t mask)
  *Enables the interrupt status.*
- static void USDHC_DisableInterruptStatus (USDHC_Type *base, uint32_t mask)
  *Disables the interrupt status.*
- static void USDHC_EnableInterruptSignal (USDHC_Type *base, uint32_t mask)
  *Enables the interrupt signal corresponding to the interrupt status flag.*
- static void USDHC_DisableInterruptSignal (USDHC_Type *base, uint32_t mask)
  *Disables the interrupt signal corresponding to the interrupt status flag.*

## Status

- static uint32_t USDHC_GetEnabledInterruptStatusFlags (USDHC_Type *base)
  *Gets the enabled interrupt status.*
- static uint32_t USDHC_GetInterruptStatusFlags (USDHC_Type *base)
  *Gets the current interrupt status.*
- static void USDHC_ClearInterruptStatusFlags (USDHC_Type *base, uint32_t mask)
  *Clears a specified interrupt status.*
- static uint32_t USDHC_GetAutoCommand12ErrorStatusFlags (USDHC_Type *base)
  *Gets the status of auto command 12 error.*
- static uint32_t USDHC_GetAdmaErrorStatusFlags (USDHC_Type *base)
  *Gets the status of the ADMA error.*
- static uint32_t USDHC_GetPresentStatusFlags (USDHC_Type *base)
  *Gets a present status.*

## Bus Operations

- void USDHC_GetCapability (USDHC_Type *base, usdhc_capability_t *capability)

*Gets the capability information.*

- static void USDHC_ForceClockOn (USDHC_Type *base, bool enable)
  *force the card clock on.*
- uint32_t USDHC_SetSdClock (USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)
  *Sets the SD bus clock frequency.*
- bool USDHC_SetCardActive (USDHC_Type *base, uint32_t timeout)
  *Sends 80 clocks to the card to set it to the active state.*
- static void USDHC_AssertHardwareReset (USDHC_Type *base, bool high)
  *trigger a hardware reset.*
- static void USDHC_SetDataBusWidth (USDHC_Type *base, usdhc_data_bus_width_t width)
  *Sets the data transfer width.*
- static void USDHC_WriteData (USDHC_Type *base, uint32_t data)
  *Fills the data port.*
- static uint32_t USDHC_ReadData (USDHC_Type *base)
  *Retrieves the data from the data port.*
- void USDHC_SendCommand (USDHC_Type *base, usdhc_command_t *command)
  *send command function*
- static void USDHC_EnableWakeupEvent (USDHC_Type *base, uint32_t mask, bool enable)
  *Enables or disables a wakeup event in low-power mode.*
- static void USDHC_CardDetectByData3 (USDHC_Type *base, bool enable)
  *detect card insert status.*
- static bool USDHC_DetectCardInsert (USDHC_Type *base)
  *detect card insert status.*
- static void USDHC_EnableSdioControl (USDHC_Type *base, uint32_t mask, bool enable)
  *Enables or disables the SDIO card control.*
- static void USDHC_SetContinueRequest (USDHC_Type *base)
  *Restarts a transaction which has stopped at the block GAP for the SDIO card.*
- static void USDHC_RequestStopAtBlockGap (USDHC_Type *base, bool enable)
  *Request stop at block gap function.*
- void USDHC_SetMmcBootConfig (USDHC_Type *base, const usdhc_boot_config_t *config)
  *Configures the MMC boot feature.*
- static void USDHC_EnableMmcBoot (USDHC_Type *base, bool enable)
  *Enables or disables the mmc boot mode.*
- static void USDHC_SetForceEvent (USDHC_Type *base, uint32_t mask)
  *Forces generating events according to the given mask.*
- static void UDSHC_SelectVoltage (USDHC_Type *base, bool en18v)
  *select the usdhc output voltage*
- static bool USDHC_RequestTuningForSDR50 (USDHC_Type *base)
  *check the SDR50 mode request tuning bit When this bit set, user should call USDHC_StandardTuning function*
- static bool USDHC_RequestReTuning (USDHC_Type *base)
  *check the request re-tuning bit When this bit is set, user should do manual tuning or standard tuning function*
- static void USDHC_EnableAutoTuning (USDHC_Type *base, bool enable)
  *the SDR104 mode auto tuning enable and disable This function should call after tuning function execute pass, auto tuning will handle by hardware*
- static void USDHC_SetRetuningTimer (USDHC_Type *base, uint32_t counter)
  *the config the re-tuning timer for mode 1 and mode 3 This timer is used for standard tuning auto re-tuning,*
- void USDHC_EnableAutoTuningForCmdAndData (USDHC_Type *base)
  *the auto tuning enbale for CMD/DATA line*

- void USDHC_EnableManualTuning (USDHC_Type *base, bool enable)
  *manual tuning trigger or abort User should handle the tuning cmd and find the boundary of the delay then calucate a average value which will be config to the CLK_TUNE_CTRL_STATUS This function should called before USDHC_AdjustDelayforSDR104 function*
- status_t USDHC_AdjustDelayForManualTuning (USDHC_Type *base, uint32_t delay)
  *the SDR104 mode delay setting adjust This function should called after USDHC_ManualTuningForSDR104*
- void USDHC_EnableStandardTuning (USDHC_Type *base, uint32_t tuningStartTap, uint32_t step, bool enable)
  *the enable standard tuning function The standard tuning window and tuning counter use the default config tuning cmd is send by the software, user need to check the tuning result can be used for SDR50,SDR104,HS200 mode tuning*
- static uint32_t USDHC_GetExecuteStdTuningStatus (USDHC_Type *base)
  *Get execute std tuning status.*
- static uint32_t USDHC_CheckStdTuningResult (USDHC_Type *base)
  *check std tuning result*
- static uint32_t USDHC_CheckTuningError (USDHC_Type *base)
  *check tuning error*
- void USDHC_EnableDDRMode (USDHC_Type *base, bool enable, uint32_t nibblePos)
  *the enable/disable DDR mode*

## Transactional

the enable/disable HS400 mode

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable/disable* | flag |

- status_t USDHC_TransferBlocking (USDHC_Type *base, usdhc_adma_config_t *dmaConfig, usdhc_transfer_t *transfer)
  *Transfers the command/data using a blocking method.*
- void USDHC_TransferCreateHandle (USDHC_Type *base, usdhc_handle_t *handle, const usdhc_transfer_callback_t *callback, void *userData)
  *Creates the USDHC handle.*
- status_t USDHC_TransferNonBlocking (USDHC_Type *base, usdhc_handle_t *handle, usdhc_adma_config_t *dmaConfig, usdhc_transfer_t *transfer)
  *Transfers the command/data using an interrupt and an asynchronous method.*
- void USDHC_TransferHandleIRQ (USDHC_Type *base, usdhc_handle_t *handle)
  *IRQ handler for the USDHC.*

## 45.3 Data Structure Documentation

### 45.3.1 struct usdhc_adma2_descriptor_t

### Data Fields

- uint32_t attribute

*The control and status field.*
- const uint32_t * address
  *The address field.*

### 45.3.2 struct usdhc_capability_t

Defines a structure to save the capability information of USDHC.

## Data Fields

- uint32_t sdVersion
  *support SD card/sdio version*
- uint32_t mmcVersion
  *support emmc card version*
- uint32_t maxBlockLength
  *Maximum block length united as byte.*
- uint32_t maxBlockCount
  *Maximum block count can be set one time.*
- uint32_t flags
  *Capability flags to indicate the support information(_usdhc_capability_flag)*

### 45.3.3 struct usdhc_boot_config_t

## Data Fields

- uint32_t ackTimeoutCount
  *Timeout value for the boot ACK.*
- usdhc_boot_mode_t bootMode
  *Boot mode selection.*
- uint32_t blockCount
  *Stop at block gap value of automatic mode.*
- size_t blockSize
  *Block size.*
- bool enableBootAck
  *Enable or disable boot ACK.*
- bool enableAutoStopAtBlockGap
  *Enable or disable auto stop at block gap function in boot period.*

#### 45.3.3.0.0.63 Field Documentation

##### 45.3.3.0.0.63.1 uint32_t usdhc_boot_config_t::ackTimeoutCount

The available range is 0 ~ 15.

#### 45.3.3.0.0.63.2 usdhc_boot_mode_t usdhc_boot_config_t::bootMode

#### 45.3.3.0.0.63.3 uint32_t usdhc_boot_config_t::blockCount

Available range is 0 ~ 65535.

### 45.3.4 struct usdhc_config_t

## Data Fields

- uint32_t dataTimeout
  *Data timeout value.*
- usdhc_endian_mode_t endianMode
  *Endian mode.*
- uint8_t readWatermarkLevel
  *Watermark level for DMA read operation.*
- uint8_t writeWatermarkLevel
  *Watermark level for DMA write operation.*
- uint8_t readBurstLen
  *Read burst len.*
- uint8_t writeBurstLen
  *Write burst len.*

#### 45.3.4.0.0.64 Field Documentation

#### 45.3.4.0.0.64.1 uint8_t usdhc_config_t::readWatermarkLevel

Available range is 1 ~ 128.

#### 45.3.4.0.0.64.2 uint8_t usdhc_config_t::writeWatermarkLevel

Available range is 1 ~ 128.

### 45.3.5 struct usdhc_data_t

Defines a structure to contain data-related attribute. 'enableIgnoreError' is used for the case that upper card driver want to ignore the error event to read/write all the data not to stop read/write immediately when error event happen for example bus testing procedure for MMC card.

## Data Fields

- bool enableAutoCommand12
  *Enable auto CMD12.*
- bool enableAutoCommand23
  *Enable auto CMD23.*
- bool enableIgnoreError

*Enable to ignore error event to read/write all the data.*
- uint8_t dataType
  *this is used to distinguish the normal/tuning/boot data*
- size_t blockSize
  *Block size.*
- uint32_t blockCount
  *Block count.*
- uint32_t * rxData
  *Buffer to save data read.*
- const uint32_t * txData
  *Data buffer to write.*

### 45.3.6 struct usdhc_command_t

Define card command-related attribute.

#### Data Fields

- uint32_t index
  *Command index.*
- uint32_t argument
  *Command argument.*
- usdhc_card_command_type_t type
  *Command type.*
- usdhc_card_response_type_t responseType
  *Command response type.*
- uint32_t response [4U]
  *Response for this command.*
- uint32_t responseErrorFlags
  *response error flag, the flag which need to check the command reponse*
- uint32_t flags
  *Cmd flags.*

### 45.3.7 struct usdhc_adma_config_t

#### Data Fields

- usdhc_dma_mode_t dmaMode
  *DMA mode.*
- usdhc_burst_len_t burstLen
  *burst len config*
- uint32_t * admaTable
  *ADMA table address, can't be null if transfer way is ADMA1/ADMA2.*
- uint32_t admaTableWords
  *ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.*

### 45.3.8 struct usdhc_transfer_t

#### Data Fields

- usdhc_data_t * data
  *Data to transfer.*
- usdhc_command_t * command
  *Command to send.*

### 45.3.9 struct usdhc_transfer_callback_t

#### Data Fields

- void(* CardInserted )(USDHC_Type *base, void *userData)
  *Card inserted occurs when DAT3/CD pin is for card detect.*
- void(* CardRemoved )(USDHC_Type *base, void *userData)
  *Card removed occurs.*
- void(* SdioInterrupt )(USDHC_Type *base, void *userData)
  *SDIO card interrupt occurs.*
- void(* BlockGap )(USDHC_Type *base, void *userData)
  *stopped at block gap event*
- void(* TransferComplete )(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)
  *Transfer complete callback.*
- void(* ReTuning )(USDHC_Type *base, void *userData)
  *handle the re-tuning*

### 45.3.10 struct _usdhc_handle

USDHC handle typedef.

Defines the structure to save the USDHC state information and callback function. The detailed interrupt status when sending a command or transfering data can be obtained from the interruptFlags field by using the mask defined in usdhc_interrupt_flag_t.

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

#### Data Fields

- usdhc_data_t *volatile data
  *Data to transfer.*
- usdhc_command_t *volatile command
  *Command to send.*

- volatile uint32_t transferredWords
  *Words transferred by DATAPORT way.*
- usdhc_transfer_callback_t callback
  *Callback function.*
- void * userData
  *Parameter for transfer complete callback.*

### 45.3.11 struct usdhc_host_t

### Data Fields

- USDHC_Type * base
  *USDHC peripheral base address.*
- uint32_t sourceClock_Hz
  *USDHC source clock frequency united in Hz.*
- usdhc_config_t config
  *USDHC configuration.*
- usdhc_capability_t capability
  *USDHC capability information.*
- usdhc_transfer_function_t transfer
  *USDHC transfer function.*

## 45.4 Macro Definition Documentation

### 45.4.1 #define FSL_USDHC_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 8U))

## 45.5 Typedef Documentation

### 45.5.1 typedef uint32_t usdhc_adma1_descriptor_t

### 45.5.2 typedef status_t(* usdhc_transfer_function_t)(USDHC_Type *base, usdhc_transfer_t *content)

## 45.6 Enumeration Type Documentation

### 45.6.1 enum _usdhc_status

Enumerator

***kStatus_USDHC_BusyTransferring*** Transfer is on-going.
***kStatus_USDHC_PrepareAdmaDescriptorFailed*** Set DMA descriptor failed.
***kStatus_USDHC_SendCommandFailed*** Send command failed.
***kStatus_USDHC_TransferDataFailed*** Transfer data failed.
***kStatus_USDHC_DMADataAddrNotAlign*** data address not align
***kStatus_USDHC_ReTuningRequest*** re-tuning request
***kStatus_USDHC_TuningError*** tuning error

***kStatus_USDHC_NotSupport*** not support

### 45.6.2 enum _usdhc_capability_flag

Enumerator

***kUSDHC_SupportAdmaFlag*** Support ADMA.
***kUSDHC_SupportHighSpeedFlag*** Support high-speed.
***kUSDHC_SupportDmaFlag*** Support DMA.
***kUSDHC_SupportSuspendResumeFlag*** Support suspend/resume.
***kUSDHC_SupportV330Flag*** Support voltage 3.3V.
***kUSDHC_SupportV300Flag*** Support voltage 3.0V.
***kUSDHC_SupportV180Flag*** Support voltage 1.8V.
***kUSDHC_Support4BitFlag*** Support 4 bit mode.
***kUSDHC_Support8BitFlag*** Support 8 bit mode.
***kUSDHC_SupportDDR50Flag*** support DDR50 mode
***kUSDHC_SupportSDR104Flag*** support SDR104 mode
***kUSDHC_SupportSDR50Flag*** support SDR50 mode

### 45.6.3 enum _usdhc_wakeup_event

Enumerator

***kUSDHC_WakeupEventOnCardInt*** Wakeup on card interrupt.
***kUSDHC_WakeupEventOnCardInsert*** Wakeup on card insertion.
***kUSDHC_WakeupEventOnCardRemove*** Wakeup on card removal.
***kUSDHC_WakeupEventsAll*** All wakeup events.

### 45.6.4 enum _usdhc_reset

Enumerator

***kUSDHC_ResetAll*** Reset all except card detection.
***kUSDHC_ResetCommand*** Reset command line.
***kUSDHC_ResetData*** Reset data line.
***kUSDHC_ResetTuning*** reset tuning circuit
***kUSDHC_ResetsAll*** All reset types.

### 45.6.5 enum _usdhc_transfer_flag

Enumerator

***kUSDHC_EnableDmaFlag*** Enable DMA.
***kUSDHC_CommandTypeSuspendFlag*** Suspend command.
***kUSDHC_CommandTypeResumeFlag*** Resume command.
***kUSDHC_CommandTypeAbortFlag*** Abort command.
***kUSDHC_EnableBlockCountFlag*** Enable block count.
***kUSDHC_EnableAutoCommand12Flag*** Enable auto CMD12.
***kUSDHC_DataReadFlag*** Enable data read.
***kUSDHC_MultipleBlockFlag*** Multiple block data read/write.
***kUSDHC_EnableAutoCommand23Flag*** Enable auto CMD23.
***kUSDHC_ResponseLength136Flag*** 136 bit response length
***kUSDHC_ResponseLength48Flag*** 48 bit response length
***kUSDHC_ResponseLength48BusyFlag*** 48 bit response length with busy status
***kUSDHC_EnableCrcCheckFlag*** Enable CRC check.
***kUSDHC_EnableIndexCheckFlag*** Enable index check.
***kUSDHC_DataPresentFlag*** Data present flag.

### 45.6.6 enum _usdhc_present_status_flag

Enumerator

***kUSDHC_CommandInhibitFlag*** Command inhibit.
***kUSDHC_DataInhibitFlag*** Data inhibit.
***kUSDHC_DataLineActiveFlag*** Data line active.
***kUSDHC_SdClockStableFlag*** SD bus clock stable.
***kUSDHC_WriteTransferActiveFlag*** Write transfer active.
***kUSDHC_ReadTransferActiveFlag*** Read transfer active.
***kUSDHC_BufferWriteEnableFlag*** Buffer write enable.
***kUSDHC_BufferReadEnableFlag*** Buffer read enable.
***kUSDHC_ReTuningRequestFlag*** re-tuning request flag ,only used for SDR104 mode
***kUSDHC_DelaySettingFinishedFlag*** delay setting finished flag
***kUSDHC_CardInsertedFlag*** Card inserted.
***kUSDHC_CommandLineLevelFlag*** Command line signal level.
***kUSDHC_Data0LineLevelFlag*** Data0 line signal level.
***kUSDHC_Data1LineLevelFlag*** Data1 line signal level.
***kUSDHC_Data2LineLevelFlag*** Data2 line signal level.
***kUSDHC_Data3LineLevelFlag*** Data3 line signal level.
***kUSDHC_Data4LineLevelFlag*** Data4 line signal level.
***kUSDHC_Data5LineLevelFlag*** Data5 line signal level.
***kUSDHC_Data6LineLevelFlag*** Data6 line signal level.
***kUSDHC_Data7LineLevelFlag*** Data7 line signal level.

### 45.6.7 enum _usdhc_interrupt_status_flag

Enumerator

***kUSDHC_CommandCompleteFlag*** Command complete.
***kUSDHC_DataCompleteFlag*** Data complete.
***kUSDHC_BlockGapEventFlag*** Block gap event.
***kUSDHC_DmaCompleteFlag*** DMA interrupt.
***kUSDHC_BufferWriteReadyFlag*** Buffer write ready.
***kUSDHC_BufferReadReadyFlag*** Buffer read ready.
***kUSDHC_CardInsertionFlag*** Card inserted.
***kUSDHC_CardRemovalFlag*** Card removed.
***kUSDHC_CardInterruptFlag*** Card interrupt.
***kUSDHC_ReTuningEventFlag*** Re-Tuning event,only for SD3.0 SDR104 mode.
***kUSDHC_TuningPassFlag*** SDR104 mode tuning pass flag.
***kUSDHC_TuningErrorFlag*** SDR104 tuning error flag.
***kUSDHC_CommandTimeoutFlag*** Command timeout error.
***kUSDHC_CommandCrcErrorFlag*** Command CRC error.
***kUSDHC_CommandEndBitErrorFlag*** Command end bit error.
***kUSDHC_CommandIndexErrorFlag*** Command index error.
***kUSDHC_DataTimeoutFlag*** Data timeout error.
***kUSDHC_DataCrcErrorFlag*** Data CRC error.
***kUSDHC_DataEndBitErrorFlag*** Data end bit error.
***kUSDHC_AutoCommand12ErrorFlag*** Auto CMD12 error.
***kUSDHC_DmaErrorFlag*** DMA error.
***kUSDHC_CommandErrorFlag*** Command error.
***kUSDHC_DataErrorFlag*** Data error.
***kUSDHC_ErrorFlag*** All error.
***kUSDHC_DataFlag*** Data interrupts.
***kUSDHC_CommandFlag*** Command interrupts.
***kUSDHC_CardDetectFlag*** Card detection interrupts.
***kUSDHC_AllInterruptFlags*** All flags mask.

### 45.6.8 enum _usdhc_auto_command12_error_status_flag

Enumerator

***kUSDHC_AutoCommand12NotExecutedFlag*** Not executed error.
***kUSDHC_AutoCommand12TimeoutFlag*** Timeout error.
***kUSDHC_AutoCommand12EndBitErrorFlag*** End bit error.
***kUSDHC_AutoCommand12CrcErrorFlag*** CRC error.
***kUSDHC_AutoCommand12IndexErrorFlag*** Index error.
***kUSDHC_AutoCommand12NotIssuedFlag*** Not issued error.

### 45.6.9 enum _usdhc_standard_tuning

Enumerator

***kUSDHC_ExecuteTuning*** used to start tuning procedure
***kUSDHC_TuningSampleClockSel*** when std_tuning_en bit is set, this bit is used select sampleing clock

### 45.6.10 enum _usdhc_adma_error_status_flag

Enumerator

***kUSDHC_AdmaLenghMismatchFlag*** Length mismatch error.
***kUSDHC_AdmaDescriptorErrorFlag*** Descriptor error.

### 45.6.11 enum _usdhc_adma_error_state

This state is the detail state when ADMA error has occurred.

Enumerator

***kUSDHC_AdmaErrorStateStopDma*** Stop DMA, previous location set in the ADMA system address is error address.
***kUSDHC_AdmaErrorStateFetchDescriptor*** Fetch descriptor, current location set in the ADMA system address is error address.
***kUSDHC_AdmaErrorStateChangeAddress*** Change address, no DMA error is occurred.
***kUSDHC_AdmaErrorStateTransferData*** Transfer data, previous location set in the ADMA system address is error address.
***kUSDHC_AdmaErrorStateInvalidLength*** Invalid length in ADMA descriptor.
***kUSDHC_AdmaErrorStateInvalidDescriptor*** Invalid descriptor fetched by ADMA.
***kUSDHC_AdmaErrorState*** ADMA error state.

### 45.6.12 enum _usdhc_force_event

Enumerator

***kUSDHC_ForceEventAutoCommand12NotExecuted*** Auto CMD12 not executed error.
***kUSDHC_ForceEventAutoCommand12Timeout*** Auto CMD12 timeout error.
***kUSDHC_ForceEventAutoCommand12CrcError*** Auto CMD12 CRC error.
***kUSDHC_ForceEventEndBitError*** Auto CMD12 end bit error.
***kUSDHC_ForceEventAutoCommand12IndexError*** Auto CMD12 index error.
***kUSDHC_ForceEventAutoCommand12NotIssued*** Auto CMD12 not issued error.

***kUSDHC_ForceEventCommandTimeout*** Command timeout error.
***kUSDHC_ForceEventCommandCrcError*** Command CRC error.
***kUSDHC_ForceEventCommandEndBitError*** Command end bit error.
***kUSDHC_ForceEventCommandIndexError*** Command index error.
***kUSDHC_ForceEventDataTimeout*** Data timeout error.
***kUSDHC_ForceEventDataCrcError*** Data CRC error.
***kUSDHC_ForceEventDataEndBitError*** Data end bit error.
***kUSDHC_ForceEventAutoCommand12Error*** Auto CMD12 error.
***kUSDHC_ForceEventCardInt*** Card interrupt.
***kUSDHC_ForceEventDmaError*** Dma error.
***kUSDHC_ForceEventTuningError*** Tuning error.
***kUSDHC_ForceEventsAll*** All force event flags mask.

### 45.6.13 enum usdhc_data_bus_width_t

Enumerator

***kUSDHC_DataBusWidth1Bit*** 1-bit mode
***kUSDHC_DataBusWidth4Bit*** 4-bit mode
***kUSDHC_DataBusWidth8Bit*** 8-bit mode

### 45.6.14 enum usdhc_endian_mode_t

Enumerator

***kUSDHC_EndianModeBig*** Big endian mode.
***kUSDHC_EndianModeHalfWordBig*** Half word big endian mode.
***kUSDHC_EndianModeLittle*** Little endian mode.

### 45.6.15 enum usdhc_dma_mode_t

Enumerator

***kUSDHC_DmaModeSimple*** external DMA
***kUSDHC_DmaModeAdma1*** ADMA1 is selected.
***kUSDHC_DmaModeAdma2*** ADMA2 is selected.
***kUSDHC_ExternalDMA*** external dma mode select

### 45.6.16 enum _usdhc_sdio_control_flag

Enumerator

***kUSDHC_StopAtBlockGapFlag*** Stop at block gap.
***kUSDHC_ReadWaitControlFlag*** Read wait control.
***kUSDHC_InterruptAtBlockGapFlag*** Interrupt at block gap.
***kUSDHC_ReadDoneNo8CLK*** read done without 8 clk for block gap
***kUSDHC_ExactBlockNumberReadFlag*** Exact block number read.

### 45.6.17 enum usdhc_boot_mode_t

Enumerator

***kUSDHC_BootModeNormal*** Normal boot.
***kUSDHC_BootModeAlternative*** Alternative boot.

### 45.6.18 enum usdhc_card_command_type_t

Enumerator

***kCARD_CommandTypeNormal*** Normal command.
***kCARD_CommandTypeSuspend*** Suspend command.
***kCARD_CommandTypeResume*** Resume command.
***kCARD_CommandTypeAbort*** Abort command.
***kCARD_CommandTypeEmpty*** Empty command.

### 45.6.19 enum usdhc_card_response_type_t

Define the command response type from card to host controller.

Enumerator

***kCARD_ResponseTypeNone*** Response type: none.
***kCARD_ResponseTypeR1*** Response type: R1.
***kCARD_ResponseTypeR1b*** Response type: R1b.
***kCARD_ResponseTypeR2*** Response type: R2.
***kCARD_ResponseTypeR3*** Response type: R3.
***kCARD_ResponseTypeR4*** Response type: R4.
***kCARD_ResponseTypeR5*** Response type: R5.
***kCARD_ResponseTypeR5b*** Response type: R5b.
***kCARD_ResponseTypeR6*** Response type: R6.
***kCARD_ResponseTypeR7*** Response type: R7.

### 45.6.20 enum _usdhc_adma1_descriptor_flag

Enumerator

***kUSDHC_Adma1DescriptorValidFlag*** Valid flag.
***kUSDHC_Adma1DescriptorEndFlag*** End flag.
***kUSDHC_Adma1DescriptorInterrupFlag*** Interrupt flag.
***kUSDHC_Adma1DescriptorActivity1Flag*** Activity 1 flag.
***kUSDHC_Adma1DescriptorActivity2Flag*** Activity 2 flag.
***kUSDHC_Adma1DescriptorTypeNop*** No operation.
***kUSDHC_Adma1DescriptorTypeTransfer*** Transfer data.
***kUSDHC_Adma1DescriptorTypeLink*** Link descriptor.
***kUSDHC_Adma1DescriptorTypeSetLength*** Set data length.

### 45.6.21 enum _usdhc_adma2_descriptor_flag

Enumerator

***kUSDHC_Adma2DescriptorValidFlag*** Valid flag.
***kUSDHC_Adma2DescriptorEndFlag*** End flag.
***kUSDHC_Adma2DescriptorInterruptFlag*** Interrupt flag.
***kUSDHC_Adma2DescriptorActivity1Flag*** Activity 1 mask.
***kUSDHC_Adma2DescriptorActivity2Flag*** Activity 2 mask.
***kUSDHC_Adma2DescriptorTypeNop*** No operation.
***kUSDHC_Adma2DescriptorTypeReserved*** Reserved.
***kUSDHC_Adma2DescriptorTypeTransfer*** Transfer type.
***kUSDHC_Adma2DescriptorTypeLink*** Link type.

### 45.6.22 enum _usdhc_adma_flag

Enumerator

***kUSDHC_AdmaDescriptorSingleFlag*** try to finish the transfer in a single ADMA descriptor, if transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer

***kUSDHC_AdmaDescriptorMultipleFlag*** create multiple ADMA descriptor within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combine with stop at block gap feature

### 45.6.23 enum usdhc_burst_len_t

Enumerator

***kUSDHC_EnBurstLenForINCR*** enable burst len for INCR
***kUSDHC_EnBurstLenForINCR4816*** enable burst len for INCR4/INCR8/INCR16
***kUSDHC_EnBurstLenForINCR4816WRAP*** enable burst len for INCR4/8/16 WRAP

### 45.6.24 enum _usdhc_transfer_data_type

Enumerator

***kUSDHC_TransferDataNormal*** transfer normal read/write data
***kUSDHC_TransferDataTuning*** transfer tuning data
***kUSDHC_TransferDataBoot*** transfer boot data
***kUSDHC_TransferDataBootcontinous*** transfer boot data continous

## 45.7 Function Documentation

### 45.7.1 void USDHC_Init ( USDHC_Type * *base*, const usdhc_config_t * *config* )

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *config* | USDHC configuration information. |

Return values

| | |
|---|---|
| *kStatus_Success* | Operate successfully. |

### 45.7.2 void USDHC_Deinit ( USDHC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.3 bool USDHC_Reset ( USDHC_Type * *base,* uint32_t *mask,* uint32_t *timeout* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | The reset type mask(_usdhc_reset). |
| *timeout* | Timeout for reset. |

Return values

| | |
|---|---|
| *true* | Reset successfully. |
| *false* | Reset failed. |

### 45.7.4 status_t USDHC_SetAdmaTableConfig ( USDHC_Type * *base,* usdhc_adma_config_t * *dmaConfig,* usdhc_data_t * *dataConfig,* uint32_t *flags* )

A high level DMA descriptor configuration function.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *adma* | configuration |
| *data* | Data descriptor |
| *flags* | ADAM descriptor flag, used to indicate to create multiple or single descriptor, please reference _usdhc_adma_flag |

Return values

| | |
|---|---|
| *kStatus_OutOfRange* | ADMA descriptor table length isn't enough to describe data. |
| *kStatus_Success* | Operate successfully. |

### 45.7.5 status_t USDHC_SetInternalDmaConfig ( USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, const uint32_t * *dataAddr*, bool *enAutoCmd23* )

This function is used to config the USDHC DMA related registers.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *adma* | configuration |
| *dataAddr* | transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL. |
| *enAutoCmd23* | flag to indicate Auto CMD23 is enable or not, a simple DMA parameter,if ADMA is used, leave it to false. |

Return values

| | |
|---|---|
| *kStatus_OutOfRange* | ADMA descriptor table length isn't enough to describe data. |
| *kStatus_Success* | Operate successfully. |

### 45.7.6 status_t USDHC_SetADMA2Descriptor ( uint32_t * *admaTable*, uint32_t *admaTableWords*, const uint32_t * *dataBufferAddr*, uint32_t *dataBytes*, uint32_t *flags* )

Parameters

| | |
|---|---|
| *admaTable* | Adma table address. |
| *admaTable-Words* | Adma table length. |
| *dataBufferAddr* | Data buffer address. |
| *dataBytes* | Data Data length. |
| *flags* | ADAM descriptor flag, used to indicate to create multiple or single descriptor, please reference _usdhc_adma_flag. |

Return values

| | |
|---|---|
| *kStatus_OutOfRange* | ADMA descriptor table length isn't enough to describe data. |
| *kStatus_Success* | Operate successfully. |

### 45.7.7 status_t USDHC_SetADMA1Descriptor ( uint32_t * *admaTable*, uint32_t *admaTableWords*, const uint32_t * *dataBufferAddr*, uint32_t *dataBytes*, uint32_t *flags* )

Parameters

| | |
|---|---|
| *admaTable* | Adma table address. |
| *admaTable-Words* | Adma table length. |
| *dataBufferAddr* | Data buffer address. |
| *dataBytes* | Data length. |
| *flags* | ADAM descriptor flag, used to indicate to create multiple or single descriptor, please reference _usdhc_adma_flag. |

Return values

| | |
|---|---|
| *kStatus_OutOfRange* | ADMA descriptor table length isn't enough to describe data. |
| *kStatus_Success* | Operate successfully. |

### 45.7.8 static void USDHC_EnableInternalDMA ( USDHC_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable* | enable or disable flag |

### 45.7.9 static void USDHC_EnableInterruptStatus ( USDHC_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | Interrupt status flags mask(_usdhc_interrupt_status_flag). |

### 45.7.10 static void USDHC_DisableInterruptStatus ( USDHC_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | The interrupt status flags mask(_usdhc_interrupt_status_flag). |

### 45.7.11 static void USDHC_EnableInterruptSignal ( USDHC_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | The interrupt status flags mask(_usdhc_interrupt_status_flag). |

### 45.7.12 static void USDHC_DisableInterruptSignal ( USDHC_Type * *base*, uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | The interrupt status flags mask(_usdhc_interrupt_status_flag). |

### 45.7.13 static uint32_t USDHC_GetEnabledInterruptStatusFlags ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

### 45.7.14 static uint32_t USDHC_GetInterruptStatusFlags ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

### 45.7.15 static void USDHC_ClearInterruptStatusFlags ( USDHC_Type * *base,* uint32_t *mask* ) [inline], [static]

write 1 clears

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | The interrupt status flags mask(_usdhc_interrupt_status_flag). |

### 45.7.16 static uint32_t USDHC_GetAutoCommand12ErrorStatusFlags ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

Returns

Auto command 12 error status flags mask(_usdhc_auto_command12_error_status_flag).

### 45.7.17 static uint32_t USDHC_GetAdmaErrorStatusFlags ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

Returns

ADMA error status flags mask(_usdhc_adma_error_status_flag).

### 45.7.18 static uint32_t USDHC_GetPresentStatusFlags ( USDHC_Type * *base* ) [inline], [static]

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

Returns

Present USDHC's status flags mask(_usdhc_present_status_flag).

### 45.7.19 void USDHC_GetCapability ( USDHC_Type * *base*, usdhc_capability_t * *capability* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *capability* | Structure to save capability information. |

### 45.7.20 static void USDHC_ForceClockOn ( USDHC_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable/disable* | flag. |

### 45.7.21 uint32_t USDHC_SetSdClock ( USDHC_Type * *base,* uint32_t *srcClock_Hz,* uint32_t *busClock_Hz* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *srcClock_Hz* | USDHC source clock frequency united in Hz. |
| *busClock_Hz* | SD bus clock frequency united in Hz. |

Returns

The nearest frequency of busClock_Hz configured to SD bus.

### 45.7.22 bool USDHC_SetCardActive ( USDHC_Type * *base*, uint32_t *timeout* )

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *timeout* | Timeout to initialize card. |

Return values

| | |
|---|---|
| *true* | Set card active successfully. |
| *false* | Set card active failed. |

### 45.7.23 static void USDHC_AssertHardwareReset ( USDHC_Type * *base*, bool *high* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *1* | or 0 level |

### 45.7.24 static void USDHC_SetDataBusWidth ( USDHC_Type * *base*, usdhc_data_bus_width_t *width* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *width* | Data transfer width. |

### 45.7.25 static void USDHC_WriteData ( USDHC_Type * *base*, uint32_t *data* ) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *data* | The data about to be sent. |

### 45.7.26 static uint32_t USDHC_ReadData ( USDHC_Type * *base* ) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

Returns

The data has been read.

### 45.7.27 void USDHC_SendCommand ( USDHC_Type * *base*, usdhc_command_t * *command* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *command* | configuration |

### 45.7.28 static void USDHC_EnableWakeupEvent ( USDHC_Type * *base,* uint32_t *mask,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | Wakeup events mask(_usdhc_wakeup_event). |
| *enable* | True to enable, false to disable. |

### 45.7.29 static void USDHC_CardDetectByData3 ( USDHC_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable/disable* | flag |

### 45.7.30 static bool USDHC_DetectCardInsert ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.31 static void USDHC_EnableSdioControl ( USDHC_Type * *base,* uint32_t *mask,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | SDIO card control flags mask(_usdhc_sdio_control_flag). |
| *enable* | True to enable, false to disable. |

### 45.7.32 static void USDHC_SetContinueRequest ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.33 static void USDHC_RequestStopAtBlockGap ( USDHC_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable* | true to stop at block gap, false to normal transfer |

### 45.7.34 void USDHC_SetMmcBootConfig ( USDHC_Type * *base*, const usdhc_boot_config_t * *config* )

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *config* | The MMC boot configuration information. |

### 45.7.35 static void USDHC_EnableMmcBoot ( USDHC_Type * *base*, bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable* | True to enable, false to disable. |

### 45.7.36 static void USDHC_SetForceEvent ( USDHC_Type * *base,* uint32_t *mask* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *mask* | The force events bit posistion (_usdhc_force_event). |

### 45.7.37 static void UDSHC_SelectVoltage ( USDHC_Type * *base,* bool *en18v* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *true* | 1.8V, false 3.0V |

### 45.7.38 static bool USDHC_RequestTuningForSDR50 ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.39 static bool USDHC_RequestReTuning ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.40 static void USDHC_EnableAutoTuning ( USDHC_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable/disable* | flag |

### 45.7.41 static void USDHC_SetRetuningTimer ( USDHC_Type * *base,* uint32_t *counter* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *timer* | counter value |

### 45.7.42 void USDHC_EnableAutoTuningForCmdAndData ( USDHC_Type * *base* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.43 void USDHC_EnableManualTuning ( USDHC_Type * *base,* bool *enable* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *tuning* | enable flag |

### 45.7.44 status_t USDHC_AdjustDelayForManualTuning ( USDHC_Type * *base,* uint32_t *delay* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *delay* | setting configuration |

Return values

| | |
|---|---|
| *kStatus_Fail* | config the delay setting fail |
| *kStatus_Success* | config the delay setting success |

### 45.7.45 void USDHC_EnableStandardTuning ( USDHC_Type * *base,* uint32_t *tuningStartTap,* uint32_t *step,* bool *enable* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *tuning* | start tap |
| *tuning* | step |
| *enable/disable* | flag |

### 45.7.46 static uint32_t USDHC_GetExecuteStdTuningStatus ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.47 static uint32_t USDHC_CheckStdTuningResult ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.48 static uint32_t USDHC_CheckTuningError ( USDHC_Type * *base* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |

### 45.7.49 void USDHC_EnableDDRMode ( USDHC_Type * *base*, bool *enable*, uint32_t *nibblePos* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *enable/disable* | flag |
| *nibble* | position |

### 45.7.50 status_t USDHC_TransferBlocking ( USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, usdhc_transfer_t * *transfer* )

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

There is no need to call the API 'USDHC_TransferCreateHandle' when calling this API.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *adma* | configuration |
| *transfer* | Transfer content. |

Return values

| | |
|---|---|
| *kStatus_InvalidArgument* | Argument is invalid. |
| *kStatus_USDHC_PrepareAdmaDescriptorFailed* | Prepare ADMA descriptor failed. |
| *kStatus_USDHC_SendCommandFailed* | Send command failed. |
| *kStatus_USDHC_TransferDataFailed* | Transfer data failed. |
| *kStatus_Success* | Operate successfully. |

### 45.7.51 void USDHC_TransferCreateHandle ( USDHC_Type * *base*, usdhc_handle_t * *handle*, const usdhc_transfer_callback_t * *callback*, void * *userData* )

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *handle* | USDHC handle pointer. |
| *callback* | Structure pointer to contain all callback functions. |
| *userData* | Callback function parameter. |

### 45.7.52 status_t USDHC_TransferNonBlocking ( USDHC_Type * *base*, usdhc_handle_t * *handle*, usdhc_adma_config_t * *dmaConfig*, usdhc_transfer_t * *transfer* )

This function sends a command and data and returns immediately. It doesn't wait the transfer complete or encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

Call the API 'USDHC_TransferCreateHandle' when calling this API.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *handle* | USDHC handle. |
| *adma* | configuration. |
| *transfer* | Transfer content. |

Return values

| | |
|---|---|
| *kStatus_InvalidArgument* | Argument is invalid. |
| *kStatus_USDHC_Busy-Transferring* | Busy transferring. |
| *kStatus_USDHC_-PrepareAdmaDescriptor-Failed* | Prepare ADMA descriptor failed. |
| *kStatus_Success* | Operate successfully. |

### 45.7.53 void USDHC_TransferHandleIRQ ( USDHC_Type * *base*, usdhc_handle_t * *handle* )

This function deals with the IRQs on the given host controller.

Parameters

| | |
|---|---|
| *base* | USDHC peripheral base address. |
| *handle* | USDHC handle. |

# Chapter 46
# WDOG: Watchdog Timer Driver

## 46.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

## 46.2 Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/wdog

## Data Structures

- struct wdog_work_mode_t
  *Defines WDOG work mode. More...*
- struct wdog_config_t
  *Describes WDOG configuration structure. More...*
- struct wdog_test_config_t
  *Describes WDOG test mode configuration structure. More...*

## Enumerations

- enum wdog_clock_source_t {
  kWDOG_LpoClockSource = 0U,
  kWDOG_AlternateClockSource = 1U }
  *Describes WDOG clock source.*
- enum wdog_clock_prescaler_t {
  kWDOG_ClockPrescalerDivide1 = 0x0U,
  kWDOG_ClockPrescalerDivide2 = 0x1U,
  kWDOG_ClockPrescalerDivide3 = 0x2U,
  kWDOG_ClockPrescalerDivide4 = 0x3U,
  kWDOG_ClockPrescalerDivide5 = 0x4U,
  kWDOG_ClockPrescalerDivide6 = 0x5U,
  kWDOG_ClockPrescalerDivide7 = 0x6U,
  kWDOG_ClockPrescalerDivide8 = 0x7U }
  *Describes the selection of the clock prescaler.*
- enum wdog_test_mode_t {
  kWDOG_QuickTest = 0U,
  kWDOG_ByteTest = 1U }
  *Describes WDOG test mode.*
- enum wdog_tested_byte_t {
  kWDOG_TestByte0 = 0U,
  kWDOG_TestByte1 = 1U,
  kWDOG_TestByte2 = 2U,

kWDOG_TestByte3 = 3U }
*Describes WDOG tested byte selection in byte test mode.*
- enum _wdog_interrupt_enable_t { kWDOG_InterruptEnable = WDOG_STCTRLH_IRQRSTEN_-MASK }
  *WDOG interrupt configuration structure, default settings all disabled.*
- enum _wdog_status_flags_t {
  kWDOG_RunningFlag = WDOG_STCTRLH_WDOGEN_MASK,
  kWDOG_TimeoutFlag = WDOG_STCTRLL_INTFLG_MASK }
  *WDOG status flags.*

## Driver version

- #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *Defines WDOG driver version 2.0.0.*

## Unlock sequence

- #define WDOG_FIRST_WORD_OF_UNLOCK (0xC520U)
  *First word of unlock sequence.*
- #define WDOG_SECOND_WORD_OF_UNLOCK (0xD928U)
  *Second word of unlock sequence.*

## Refresh sequence

- #define WDOG_FIRST_WORD_OF_REFRESH (0xA602U)
  *First word of refresh sequence.*
- #define WDOG_SECOND_WORD_OF_REFRESH (0xB480U)
  *Second word of refresh sequence.*

## WDOG Initialization and De-initialization

- void WDOG_GetDefaultConfig (wdog_config_t *config)
  *Initializes the WDOG configuration structure.*
- void WDOG_Init (WDOG_Type *base, const wdog_config_t *config)
  *Initializes the WDOG.*
- void WDOG_Deinit (WDOG_Type *base)
  *Shuts down the WDOG.*
- void WDOG_SetTestModeConfig (WDOG_Type *base, wdog_test_config_t *config)
  *Configures the WDOG functional test.*

## WDOG Functional Operation

- static void WDOG_Enable (WDOG_Type *base)
  *Enables the WDOG module.*
- static void WDOG_Disable (WDOG_Type *base)
  *Disables the WDOG module.*
- static void WDOG_EnableInterrupts (WDOG_Type *base, uint32_t mask)
  *Enables the WDOG interrupt.*
- static void WDOG_DisableInterrupts (WDOG_Type *base, uint32_t mask)
  *Disables the WDOG interrupt.*

- uint32_t WDOG_GetStatusFlags (WDOG_Type *base)
  *Gets the WDOG all status flags.*
- void WDOG_ClearStatusFlags (WDOG_Type *base, uint32_t mask)
  *Clears the WDOG flag.*
- static void WDOG_SetTimeoutValue (WDOG_Type *base, uint32_t timeoutCount)
  *Sets the WDOG timeout value.*
- static void WDOG_SetWindowValue (WDOG_Type *base, uint32_t windowValue)
  *Sets the WDOG window value.*
- static void WDOG_Unlock (WDOG_Type *base)
  *Unlocks the WDOG register written.*
- void WDOG_Refresh (WDOG_Type *base)
  *Refreshes the WDOG timer.*
- static uint16_t WDOG_GetResetCount (WDOG_Type *base)
  *Gets the WDOG reset count.*
- static void WDOG_ClearResetCount (WDOG_Type *base)
  *Clears the WDOG reset count.*

## 46.3 Data Structure Documentation

### 46.3.1 struct wdog_work_mode_t

#### Data Fields

- bool enableStop
  *Enables or disables WDOG in stop mode.*
- bool enableDebug
  *Enables or disables WDOG in debug mode.*

### 46.3.2 struct wdog_config_t

#### Data Fields

- bool enableWdog
  *Enables or disables WDOG.*
- wdog_clock_source_t clockSource
  *Clock source select.*
- wdog_clock_prescaler_t prescaler
  *Clock prescaler value.*
- wdog_work_mode_t workMode
  *Configures WDOG work mode in debug stop and wait mode.*
- bool enableUpdate
  *Update write-once register enable.*
- bool enableInterrupt
  *Enables or disables WDOG interrupt.*
- bool enableWindowMode
  *Enables or disables WDOG window mode.*
- uint32_t windowValue
  *Window value.*

- uint32_t timeoutValue
  *Timeout value.*

### 46.3.3 struct wdog_test_config_t

### Data Fields

- wdog_test_mode_t testMode
  *Selects test mode.*
- wdog_tested_byte_t testedByte
  *Selects tested byte in byte test mode.*
- uint32_t timeoutValue
  *Timeout value.*

## 46.4 Macro Definition Documentation

### 46.4.1 #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

## 46.5 Enumeration Type Documentation

### 46.5.1 enum wdog_clock_source_t

Enumerator

***kWDOG_LpoClockSource*** WDOG clock sourced from LPO.
***kWDOG_AlternateClockSource*** WDOG clock sourced from alternate clock source.

### 46.5.2 enum wdog_clock_prescaler_t

Enumerator

***kWDOG_ClockPrescalerDivide1*** Divided by 1.
***kWDOG_ClockPrescalerDivide2*** Divided by 2.
***kWDOG_ClockPrescalerDivide3*** Divided by 3.
***kWDOG_ClockPrescalerDivide4*** Divided by 4.
***kWDOG_ClockPrescalerDivide5*** Divided by 5.
***kWDOG_ClockPrescalerDivide6*** Divided by 6.
***kWDOG_ClockPrescalerDivide7*** Divided by 7.
***kWDOG_ClockPrescalerDivide8*** Divided by 8.

### 46.5.3 enum wdog_test_mode_t

Enumerator

***kWDOG_QuickTest*** Selects quick test.
***kWDOG_ByteTest*** Selects byte test.

### 46.5.4 enum wdog_tested_byte_t

Enumerator

***kWDOG_TestByte0*** Byte 0 selected in byte test mode.
***kWDOG_TestByte1*** Byte 1 selected in byte test mode.
***kWDOG_TestByte2*** Byte 2 selected in byte test mode.
***kWDOG_TestByte3*** Byte 3 selected in byte test mode.

### 46.5.5 enum _wdog_interrupt_enable_t

This structure contains the settings for all of the WDOG interrupt configurations.

Enumerator

***kWDOG_InterruptEnable*** WDOG timeout generates an interrupt before reset.

### 46.5.6 enum _wdog_status_flags_t

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

***kWDOG_RunningFlag*** Running flag, set when WDOG is enabled.
***kWDOG_TimeoutFlag*** Interrupt flag, set when an exception occurs.

## 46.6 Function Documentation

### 46.6.1 void WDOG_GetDefaultConfig ( wdog_config_t * *config* )

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
*    wdogConfig->enableWdog = true;
*    wdogConfig->clockSource = kWDOG_LpoClockSource;
*    wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
*    wdogConfig->workMode.enableWait = true;
*    wdogConfig->workMode.enableStop = false;
*    wdogConfig->workMode.enableDebug = false;
*    wdogConfig->enableUpdate = true;
*    wdogConfig->enableInterrupt = false;
*    wdogConfig->enableWindowMode = false;
*    wdogConfig->windowValue = 0;
*    wdogConfig->timeoutValue = 0xFFFFU;
*
```

Parameters

| | |
|---|---|
| *config* | Pointer to the WDOG configuration structure. |

See Also

wdog_config_t

### 46.6.2 void WDOG_Init ( WDOG_Type * *base*, const wdog_config_t * *config* )

This function initializes the WDOG. When called, the WDOG runs according to the configuration. To reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

This is an example.

```
*    wdog_config_t config;
*    WDOG_GetDefaultConfig(&config);
*    config.timeoutValue = 0x7ffU;
*    config.enableUpdate = true;
*    WDOG_Init(wdog_base,&config);
*
```

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |
| *config* | The configuration of WDOG |

### 46.6.3 void WDOG_Deinit ( WDOG_Type * *base* )

This function shuts down the WDOG. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which indicates that the register update is enabled.

### 46.6.4 void WDOG_SetTestModeConfig ( WDOG_Type * *base*, wdog_test_config_t * *config* )

This function is used to configure the WDOG functional test. When called, the WDOG goes into test mode and runs according to the configuration. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

This is an example.

```
*   wdog_test_config_t test_config;
*   test_config.testMode = kWDOG_QuickTest;
*   test_config.timeoutValue = 0xffffu;
*   WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |
| *config* | The functional test configuration of WDOG |

### 46.6.5 static void WDOG_Enable ( WDOG_Type * *base* ) [inline], [static]

This function write value into WDOG_STCTRLH register to enable the WDOG, it is a write-once register, make sure that the WCT window is still open and this register has not been written in this WCT while this function is called.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |

### 46.6.6 static void WDOG_Disable ( WDOG_Type * *base* ) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG. It is a write-once register. Ensure that the WCT window is still open and that register has not been written to in this WCT while the function is called.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |

### 46.6.7 static void WDOG_EnableInterrupts ( WDOG_Type * *base*, uint32_t *mask* ) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to enable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |
| *mask* | The interrupts to enable The parameter can be combination of the following source if defined.<br>• kWDOG_InterruptEnable |

### 46.6.8 static void WDOG_DisableInterrupts ( WDOG_Type * *base*, uint32_t *mask* ) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |
| *mask* | The interrupts to disable The parameter can be combination of the following source if defined.<br>• kWDOG_InterruptEnable |

### 46.6.9 uint32_t WDOG_GetStatusFlags ( WDOG_Type * *base* )

This function gets all status flags.

This is an example for getting the Running Flag.

```
*   uint32_t status;
*   status = WDOG_GetStatusFlags (wdog_base) &
      kWDOG_RunningFlag;
```

*

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

_wdog_status_flags_t
- true: a related status flag has been set.
- false: a related status flag is not set.

### 46.6.10 void WDOG_ClearStatusFlags ( WDOG_Type * *base*, uint32_t *mask* )

This function clears the WDOG status flag.

This is an example for clearing the timeout (interrupt) flag.

```
*   WDOG_ClearStatusFlags(wdog_base,kWDOG_TimeoutFlag);
*
```

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |
| *mask* | The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag |

### 46.6.11 static void WDOG_SetTimeoutValue ( WDOG_Type * *base*, uint32_t *timeoutCount* ) [inline], [static]

This function sets the timeout value. It should be ensured that the time-out value for the WDOG is always greater than 2xWCT time + 20 bus clock cycles. This function writes a value into WDOG_TOVALH and WDOG_TOVALL registers which are wirte-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |
| *timeoutCount* | WDOG timeout value; count of WDOG clock tick. |

### 46.6.12 static void WDOG_SetWindowValue ( WDOG_Type * *base*, uint32_t *windowValue* ) [inline], [static]

This function sets the WDOG window value. This function writes a value into WDOG_WINH and WDOG_WINL registers which are wirte-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |
| *windowValue* | WDOG window value. |

### 46.6.13 static void WDOG_Unlock ( WDOG_Type * *base* ) [inline], [static]

This function unlocks the WDOG register written. Before starting the unlock sequence and following configuration, disable the global interrupts. Otherwise, an interrupt may invalidate the unlocking sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |

### 46.6.14 void WDOG_Refresh ( WDOG_Type * *base* )

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |

### 46.6.15 static uint16_t WDOG_GetResetCount ( WDOG_Type * *base* ) [inline], [static]

This function gets the WDOG reset count value.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |

Returns

WDOG reset count value.

### 46.6.16 static void WDOG_ClearResetCount ( WDOG_Type * *base* ) [inline], [static]

This function clears the WDOG reset count value.

Parameters

| | |
|---|---|
| *base* | WDOG peripheral base address |

# Chapter 47
# XBARA: Inter-Peripheral Crossbar Switch

## 47.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARA) block of MCUXpresso SDK devices.

The XBARA peripheral driver configures the XBARA (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARA module.

XBARA driver has two parts:

- Signal connection interconnects input and output signals.
- Active edge feature - Some of the outputs provide an active edge detection. If an active edge occurs, an interrupt or a DMA request can be called. APIs handle user callbacks for the interrupts. The driver also includes API for clearing and reading status bits.

## 47.2 Function

### 47.2.1 XBARA Initialization

To initialize the XBARA driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARA module is initialized by calling the XBARA_Init() function.

### 47.2.2 Call diagram

1. Call the "XBARA_Init()" function to initialize the XBARA module.
2. Optionally, call the "XBARA_SetSignalsConnection()" function to Set connection between the selected XBARA_IN[∗] input and the XBARA_OUT[∗] output signal. It connects the XBARA input to the selected XBARA output. A configuration structure of the "xbara_input_signal_t" type and "xbara_output_signal_t" type is required.
3. Call the "XBARA_SetOutputSignalConfig" function to set the active edge features, such interrupts or DMA requests. A configuration structure of the "xbara_control_config_t" type is required to point to structure that keeps configuration of control register.
4. Finally, the XBARA works properly.

## 47.3 Typical use case

## Data Structures

- struct xbara_control_config_t
  *Defines the configuration structure of the XBARA control register. More...*

## Macros

- #define FSL_XBARA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  *Version 2.0.4.*

## Enumerations

- enum xbara_active_edge_t {
  kXBARA_EdgeNone = 0U,
  kXBARA_EdgeRising = 1U,
  kXBARA_EdgeFalling = 2U,
  kXBARA_EdgeRisingAndFalling = 3U }
  *XBARA active edge for detection.*
- enum xbara_request_t {
  kXBARA_RequestDisable = 0U,
  kXBARA_RequestDMAEnable = 1U,
  kXBARA_RequestInterruptEnalbe = 2U }
  *Defines the XBARA DMA and interrupt configurations.*
- enum xbara_status_flag_t {
  kXBARA_EdgeDetectionOut0,
  kXBARA_EdgeDetectionOut1,
  kXBARA_EdgeDetectionOut2,
  kXBARA_EdgeDetectionOut3 }
  *XBARA status flags.*

## XBARA functional Operation.

- void XBARA_Init (XBARA_Type *base)
  *Initializes the XBARA module.*
- void XBARA_Deinit (XBARA_Type *base)
  *Shuts down the XBARA module.*
- void XBARA_SetSignalsConnection (XBARA_Type *base, xbar_input_signal_t input, xbar_output_signal_t output)
  *Sets a connection between the selected XBARA_IN[*] input and the XBARA_OUT[*] output signal.*
- uint32_t XBARA_GetStatusFlags (XBARA_Type *base)
  *Gets the active edge detection status.*
- void XBARA_ClearStatusFlags (XBARA_Type *base, uint32_t mask)
  *Clears the edge detection status flags of relative mask.*
- void XBARA_SetOutputSignalConfig (XBARA_Type *base, xbar_output_signal_t output, const xbara_control_config_t *controlConfig)
  *Configures the XBARA control register.*

# 47.4 Data Structure Documentation

## 47.4.1 struct xbara_control_config_t

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.

### Data Fields

- xbara_active_edge_t activeEdge
  *Active edge to be detected.*
- xbara_request_t requestType
  *Selects DMA/Interrupt request.*

#### 47.4.1.0.0.65 Field Documentation

#### 47.4.1.0.0.65.1 xbara_active_edge_t xbara_control_config_t::activeEdge

#### 47.4.1.0.0.65.2 xbara_request_t xbara_control_config_t::requestType

## 47.5 Macro Definition Documentation

### 47.5.1 #define FSL_XBARA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

## 47.6 Enumeration Type Documentation

### 47.6.1 enum xbara_active_edge_t

Enumerator

***kXBARA_EdgeNone*** Edge detection status bit never asserts.
***kXBARA_EdgeRising*** Edge detection status bit asserts on rising edges.
***kXBARA_EdgeFalling*** Edge detection status bit asserts on falling edges.
***kXBARA_EdgeRisingAndFalling*** Edge detection status bit asserts on rising and falling edges.

### 47.6.2 enum xbara_request_t

Enumerator

***kXBARA_RequestDisable*** Interrupt and DMA are disabled.
***kXBARA_RequestDMAEnable*** DMA enabled, interrupt disabled.
***kXBARA_RequestInterruptEnalbe*** Interrupt enabled, DMA disabled.

### 47.6.3 enum xbara_status_flag_t

This provides constants for the XBARA status flags for use in the XBARA functions.

Enumerator

***kXBARA_EdgeDetectionOut0*** XBAR_OUT0 active edge interrupt flag, sets when active edge detected.

***kXBARA_EdgeDetectionOut1*** XBAR_OUT1 active edge interrupt flag, sets when active edge detected.

***kXBARA_EdgeDetectionOut2*** XBAR_OUT2 active edge interrupt flag, sets when active edge detected.

***kXBARA_EdgeDetectionOut3*** XBAR_OUT3 active edge interrupt flag, sets when active edge detected.

## 47.7 Function Documentation

### 47.7.1 void XBARA_Init ( XBARA_Type * *base* )

This function un-gates the XBARA clock.

Parameters

| | |
|---|---|
| *base* | XBARA peripheral address. |

### 47.7.2 void XBARA_Deinit ( XBARA_Type * *base* )

This function disables XBARA clock.

Parameters

| | |
|---|---|
| *base* | XBARA peripheral address. |

### 47.7.3 void XBARA_SetSignalsConnection ( XBARA_Type * *base*, xbar_input_signal_t *input*, xbar_output_signal_t *output* )

This function connects the XBARA input to the selected XBARA output. If more than one XBARA module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBARA_SetSignalsConnection(XBARA, kXBARA_InputPIT_TRG0, kXBARA_OutputDMAMUX18);
```

Parameters

| | |
|---|---|
| *base* | XBARA peripheral address. |
| *input* | XBARA input signal. |
| *output* | XBARA output signal. |

### 47.7.4 uint32_t XBARA_GetStatusFlags ( XBARA_Type * *base* )

This function gets the active edge detect status of all XBARA_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBARA_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

| | |
|---|---|
| *base* | XBARA peripheral address. |

Returns

the mask of these status flag bits.

### 47.7.5 void XBARA_ClearStatusFlags ( XBARA_Type * *base*, uint32_t *mask* )

Parameters

| | |
|---|---|
| *base* | XBARA peripheral address. |
| *mask* | the status flags to clear. |

### 47.7.6 void XBARA_SetOutputSignalConfig ( XBARA_Type * *base*, xbar_output_signal_t *output*, const xbara_control_config_t * *controlConfig* )

This function configures an XBARA control register. The active edge detection and the DMA/IRQ function on the corresponding XBARA output can be set.

Example:

```
xbara_control_config_t userConfig;
userConfig.activeEdge = kXBARA_EdgeRising;
userConfig.requestType = kXBARA_RequestInterruptEnalbe;
XBARA_SetOutputSignalConfig(XBARA, kXBARA_OutputDMAMUX18, &userConfig);
```

Parameters

| | |
|---|---|
| *base* | XBARA peripheral address. |
| *output* | XBARA output number. |
| *controlConfig* | Pointer to structure that keeps configuration of control register. |

# Chapter 48
# XBARB: Inter-Peripheral Crossbar Switch

## 48.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARB) block of MCUXpresso SDK devices.

The XBARB peripheral driver configures the XBARB (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARB module.

XBARB driver has two parts:

- Signal connection interconnects input and output signals.

## 48.2 Function groups

### 48.2.1 XBARB Initialization

To initialize the XBARB driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARB module is initialized by calling the XBARB_Init() function.

### 48.2.2 Call diagram

1. Call the "XBARB_Init()" function to initialize the XBARB module.
2. Optionally, call the "XBARB_SetSignalsConnection()" function to Set connection between the selected XBARB_IN[∗] input and the XBARB_OUT[∗] output signal. It connects the XBARB input to the selected XBARB output. A configuration structure of the "xbarb_input_signal_t" type and "xbarb_output_signal_t" type is required.
3. Finally, the XBARB works properly.

## 48.3 Typical use case

## Macros

- #define FSL_XBARB_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  *Version 2.0.1.*

## XBARB functional Operation.

- void XBARB_Init (XBARB_Type ∗base)
  *Initializes the XBARB module.*
- void XBARB_Deinit (XBARB_Type ∗base)

*Shuts down the XBARB module.*

- void XBARB_SetSignalsConnection (XBARB_Type *base, xbar_input_signal_t input, xbar_output_signal_t output)
  *Configures a connection between the selected XBARB_IN[*] input and the XBARB_OUT[*] output signal.*

## 48.4 Macro Definition Documentation

### 48.4.1 #define FSL_XBARB_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

## 48.5 Function Documentation

### 48.5.1 void XBARB_Init ( XBARB_Type * *base* )

This function un-gates the XBARB clock.

Parameters

| | |
|---|---|
| *base* | XBARB peripheral address. |

### 48.5.2 void XBARB_Deinit ( XBARB_Type * *base* )

This function disables XBARB clock.

Parameters

| | |
|---|---|
| *base* | XBARB peripheral address. |

### 48.5.3 void XBARB_SetSignalsConnection ( XBARB_Type * *base*, xbar_input_signal_t *input*, xbar_output_signal_t *output* )

This function configures which XBARB input is connected to the selected XBARB output. If more than one XBARB module is available, only the inputs and outputs from the same module can be connected.

Parameters

| | |
|---|---|
| *base* | XBARB peripheral address. |
| *input* | XBARB input signal. |

| | |
|---|---|
| *output* | XBARB output signal. |

# Chapter 49
# Clock Driver

## 49.1 Overview

The MCUXpresso SDK provides APIs for MCUXpresso SDK devices' clock operation.

### Files

- file fsl_clock.h

### Data Structures

- struct clock_arm_pll_config_t
  *PLL configuration for ARM. More...*
- struct clock_usb_pll_config_t
  *PLL configuration for USB. More...*
- struct clock_sys_pll_config_t
  *PLL configuration for System. More...*
- struct clock_audio_pll_config_t
  *PLL configuration for AUDIO and VIDEO. More...*
- struct clock_video_pll_config_t
  *PLL configuration for AUDIO and VIDEO. More...*
- struct clock_enet_pll_config_t
  *PLL configuration for ENET. More...*

### Macros

- #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0
  *Configure whether driver controls clock.*
- #define CCSR_OFFSET 0x0C
  *CCM registers offset.*
- #define PLL_ARM_OFFSET 0x00
  *CCM Analog registers offset.*
- #define CCM_ANALOG_TUPLE(reg, shift) (((reg & 0xFFFU) << 16U) | (shift))
  *CCM ANALOG tuple macros to map corresponding registers and bit fields.*
- #define CLKPN_FREQ 0U
  *clock1PN frequency.*
- #define ADC_CLOCKS
  *Clock ip name array for ADC.*
- #define AOI_CLOCKS
  *Clock ip name array for AOI.*
- #define BEE_CLOCKS
  *Clock ip name array for BEE.*
- #define CMP_CLOCKS
  *Clock ip name array for CMP.*
- #define CSI_CLOCKS

*Clock ip name array for CSI.*

- #define DCDC_CLOCKS
  *Clock ip name array for DCDC.*
- #define DCP_CLOCKS
  *Clock ip name array for DCP.*
- #define DMAMUX_CLOCKS
  *Clock ip name array for DMAMUX_CLOCKS.*
- #define EDMA_CLOCKS
  *Clock ip name array for DMA.*
- #define ENC_CLOCKS
  *Clock ip name array for ENC.*
- #define ENET_CLOCKS
  *Clock ip name array for ENET.*
- #define EWM_CLOCKS
  *Clock ip name array for EWM.*
- #define FLEXCAN_CLOCKS
  *Clock ip name array for FLEXCAN.*
- #define FLEXCAN_PERIPH_CLOCKS
  *Clock ip name array for FLEXCAN Peripheral clock.*
- #define FLEXIO_CLOCKS
  *Clock ip name array for FLEXIO.*
- #define FLEXRAM_CLOCKS
  *Clock ip name array for FLEXRAM.*
- #define FLEXSPI_CLOCKS
  *Clock ip name array for FLEXSPI.*
- #define FLEXSPI_EXSC_CLOCKS
  *Clock ip name array for FLEXSPI EXSC.*
- #define GPIO_CLOCKS
  *Clock ip name array for GPIO.*
- #define GPT_CLOCKS
  *Clock ip name array for GPT.*
- #define KPP_CLOCKS
  *Clock ip name array for KPP.*
- #define LCDIF_CLOCKS
  *Clock ip name array for LCDIF.*
- #define LCDIF_PERIPH_CLOCKS
  *Clock ip name array for LCDIF PIXEL.*
- #define LPI2C_CLOCKS
  *Clock ip name array for LPI2C.*
- #define LPSPI_CLOCKS
  *Clock ip name array for LPSPI.*
- #define LPUART_CLOCKS
  *Clock ip name array for LPUART.*
- #define MQS_CLOCKS
  *Clock ip name array for MQS.*
- #define OCRAM_EXSC_CLOCKS
  *Clock ip name array for OCRAM EXSC.*
- #define PIT_CLOCKS
  *Clock ip name array for PIT.*
- #define PWM_CLOCKS
  *Clock ip name array for PWM.*

- #define PXP_CLOCKS
  *Clock ip name array for PXP.*
- #define RTWDOG_CLOCKS
  *Clock ip name array for RTWDOG.*
- #define SAI_CLOCKS
  *Clock ip name array for SAI.*
- #define SEMC_CLOCKS
  *Clock ip name array for SEMC.*
- #define SEMC_EXSC_CLOCKS
  *Clock ip name array for SEMC EXSC.*
- #define TMR_CLOCKS
  *Clock ip name array for QTIMER.*
- #define TRNG_CLOCKS
  *Clock ip name array for TRNG.*
- #define TSC_CLOCKS
  *Clock ip name array for TSC.*
- #define WDOG_CLOCKS
  *Clock ip name array for WDOG.*
- #define USDHC_CLOCKS
  *Clock ip name array for USDHC.*
- #define SPDIF_CLOCKS
  *Clock ip name array for SPDIF.*
- #define XBARA_CLOCKS
  *Clock ip name array for XBARA.*
- #define XBARB_CLOCKS
  *Clock ip name array for XBARB.*
- #define kCLOCK_CoreSysClk kCLOCK_CpuClk
  *For compatible with other platforms without CCM.*
- #define CLOCK_GetCoreSysClkFreq CLOCK_GetCpuClkFreq
  *For compatible with other platforms without CCM.*

## Enumerations

- enum clock_name_t {
  kCLOCK_CpuClk = 0x0U,
  kCLOCK_AhbClk = 0x1U,
  kCLOCK_SemcClk = 0x2U,
  kCLOCK_IpgClk = 0x3U,
  kCLOCK_PerClk = 0x4U,
  kCLOCK_OscClk = 0x5U,
  kCLOCK_RtcClk = 0x6U,
  kCLOCK_ArmPllClk = 0x7U,
  kCLOCK_Usb1PllClk = 0x8U,
  kCLOCK_Usb1PllPfd0Clk = 0x9U,
  kCLOCK_Usb1PllPfd1Clk = 0xAU,
  kCLOCK_Usb1PllPfd2Clk = 0xBU,
  kCLOCK_Usb1PllPfd3Clk = 0xCU,
  kCLOCK_Usb2PllClk = 0xDU,
  kCLOCK_SysPllClk = 0xEU,
  kCLOCK_SysPllPfd0Clk = 0xFU,
  kCLOCK_SysPllPfd1Clk = 0x10U,
  kCLOCK_SysPllPfd2Clk = 0x11U,
  kCLOCK_SysPllPfd3Clk = 0x12U,
  kCLOCK_EnetPll0Clk = 0x13U,
  kCLOCK_EnetPll1Clk = 0x14U,
  kCLOCK_AudioPllClk = 0x15U,
  kCLOCK_VideoPllClk = 0x16U }
  *Clock name used to get clock frequency.*
- enum clock_ip_name_t { ,

kCLOCK_Aips_tz1 = (0U << 8U) | CCM_CCGR0_CG0_SHIFT,
kCLOCK_Aips_tz2 = (0U << 8U) | CCM_CCGR0_CG1_SHIFT,
kCLOCK_Mqs = (0U << 8U) | CCM_CCGR0_CG2_SHIFT,
kCLOCK_FlexSpiExsc = (0U << 8U) | CCM_CCGR0_CG3_SHIFT,
kCLOCK_Sim_M_Main = (0U << 8U) | CCM_CCGR0_CG4_SHIFT,
kCLOCK_Dcp = (0U << 8U) | CCM_CCGR0_CG5_SHIFT,
kCLOCK_Lpuart3 = (0U << 8U) | CCM_CCGR0_CG6_SHIFT,
kCLOCK_Can1 = (0U << 8U) | CCM_CCGR0_CG7_SHIFT,
kCLOCK_Can1S = (0U << 8U) | CCM_CCGR0_CG8_SHIFT,
kCLOCK_Can2 = (0U << 8U) | CCM_CCGR0_CG9_SHIFT,
kCLOCK_Can2S = (0U << 8U) | CCM_CCGR0_CG10_SHIFT,
kCLOCK_Trace = (0U << 8U) | CCM_CCGR0_CG11_SHIFT,
kCLOCK_Gpt2 = (0U << 8U) | CCM_CCGR0_CG12_SHIFT,
kCLOCK_Gpt2S = (0U << 8U) | CCM_CCGR0_CG13_SHIFT,
kCLOCK_Lpuart2 = (0U << 8U) | CCM_CCGR0_CG14_SHIFT,
kCLOCK_Gpio2 = (0U << 8U) | CCM_CCGR0_CG15_SHIFT,
kCLOCK_Lpspi1 = (1U << 8U) | CCM_CCGR1_CG0_SHIFT,
kCLOCK_Lpspi2 = (1U << 8U) | CCM_CCGR1_CG1_SHIFT,
kCLOCK_Lpspi3 = (1U << 8U) | CCM_CCGR1_CG2_SHIFT,
kCLOCK_Lpspi4 = (1U << 8U) | CCM_CCGR1_CG3_SHIFT,
kCLOCK_Adc2 = (1U << 8U) | CCM_CCGR1_CG4_SHIFT,
kCLOCK_Enet = (1U << 8U) | CCM_CCGR1_CG5_SHIFT,
kCLOCK_Pit = (1U << 8U) | CCM_CCGR1_CG6_SHIFT,
kCLOCK_Aoi2 = (1U << 8U) | CCM_CCGR1_CG7_SHIFT,
kCLOCK_Adc1 = (1U << 8U) | CCM_CCGR1_CG8_SHIFT,
kCLOCK_SemcExsc = (1U << 8U) | CCM_CCGR1_CG9_SHIFT,
kCLOCK_Gpt1 = (1U << 8U) | CCM_CCGR1_CG10_SHIFT,
kCLOCK_Gpt1S = (1U << 8U) | CCM_CCGR1_CG11_SHIFT,
kCLOCK_Lpuart4 = (1U << 8U) | CCM_CCGR1_CG12_SHIFT,
kCLOCK_Gpio1 = (1U << 8U) | CCM_CCGR1_CG13_SHIFT,
kCLOCK_Csu = (1U << 8U) | CCM_CCGR1_CG14_SHIFT,
kCLOCK_Gpio5 = (1U << 8U) | CCM_CCGR1_CG15_SHIFT,
kCLOCK_OcramExsc = (2U << 8U) | CCM_CCGR2_CG0_SHIFT,
kCLOCK_Csi = (2U << 8U) | CCM_CCGR2_CG1_SHIFT,
kCLOCK_IomuxcSnvs = (2U << 8U) | CCM_CCGR2_CG2_SHIFT,
kCLOCK_Lpi2c1 = (2U << 8U) | CCM_CCGR2_CG3_SHIFT,
kCLOCK_Lpi2c2 = (2U << 8U) | CCM_CCGR2_CG4_SHIFT,
kCLOCK_Lpi2c3 = (2U << 8U) | CCM_CCGR2_CG5_SHIFT,
kCLOCK_Ocotp = (2U << 8U) | CCM_CCGR2_CG6_SHIFT,
kCLOCK_Xbar3 = (2U << 8U) | CCM_CCGR2_CG7_SHIFT,
kCLOCK_Ipmux1 = (2U << 8U) | CCM_CCGR2_CG8_SHIFT,
kCLOCK_Ipmux2 = (2U << 8U) | CCM_CCGR2_CG9_SHIFT,
kCLOCK_Ipmux3 = (2U << 8U) | CCM_CCGR2_CG10_SHIFT,
kCLOCK_Xbar1 = (2U << 8U) | CCM_CCGR2_CG11_SHIFT,
kCLOCK_Xbar2 = (2U << 8U) | CCM_CCGR2_CG12_SHIFT,
kCLOCK_Gpio3 = (2U << 8U) | CCM_CCGR2_CG13_SHIFT,
kCLOCK_Lcd = (2U << 8U) | CCM_CCGR2_CG14_SHIFT,
kCLOCK_Pxp = (2U << 8U) | CCM_CCGR2_CG15_SHIFT,
kCLOCK_Flexio2 = (3U << 8U) | CCM_CCGR3_CG0_SHIFT,

kCLOCK_Timer3 = (6U << 8U) | CCM_CCGR6_CG15_SHIFT }
*CCM CCGR gate control for each module independently.*

- enum clock_osc_t {
  kCLOCK_RcOsc = 0U,
  kCLOCK_XtalOsc = 1U }
  *OSC 24M sorce select.*
- enum clock_gate_value_t {
  kCLOCK_ClockNotNeeded = 0U,
  kCLOCK_ClockNeededRun = 1U,
  kCLOCK_ClockNeededRunWait = 3U }
  *Clock gate value.*
- enum clock_mode_t {
  kCLOCK_ModeRun = 0U,
  kCLOCK_ModeWait = 1U,
  kCLOCK_ModeStop = 2U }
  *System clock mode.*
- enum clock_mux_t {
  kCLOCK_Pll3SwMux,
  kCLOCK_PeriphMux,
  kCLOCK_SemcAltMux,
  kCLOCK_SemcMux,
  kCLOCK_PrePeriphMux,
  kCLOCK_TraceMux,
  kCLOCK_PeriphClk2Mux,
  kCLOCK_LpspiMux,
  kCLOCK_FlexspiMux,
  kCLOCK_Usdhc2Mux,
  kCLOCK_Usdhc1Mux,
  kCLOCK_Sai3Mux,
  kCLOCK_Sai2Mux,
  kCLOCK_Sai1Mux,
  kCLOCK_PerclkMux,
  kCLOCK_Flexio2Mux,
  kCLOCK_CanMux,
  kCLOCK_UartMux,
  kCLOCK_SpdifMux,
  kCLOCK_Flexio1Mux,
  kCLOCK_Lpi2cMux,
  kCLOCK_LcdifPreMux,
  kCLOCK_CsiMux }
  *MUX control names for clock mux setting.*
- enum clock_div_t {

kCLOCK_ArmDiv,
kCLOCK_PeriphClk2Div,
kCLOCK_SemcDiv,
kCLOCK_AhbDiv,
kCLOCK_IpgDiv,
kCLOCK_LpspiDiv,
kCLOCK_LcdifDiv,
kCLOCK_FlexspiDiv,
kCLOCK_PerclkDiv,
kCLOCK_CanDiv,
kCLOCK_TraceDiv,
kCLOCK_Usdhc2Div,
kCLOCK_Usdhc1Div,
kCLOCK_UartDiv,
kCLOCK_Flexio2Div,
kCLOCK_Sai3PreDiv,
kCLOCK_Sai3Div,
kCLOCK_Flexio2PreDiv,
kCLOCK_Sai1PreDiv,
kCLOCK_Sai1Div,
kCLOCK_Sai2PreDiv,
kCLOCK_Sai2Div,
kCLOCK_Spdif0PreDiv,
kCLOCK_Spdif0Div,
kCLOCK_Flexio1PreDiv,
kCLOCK_Flexio1Div,
kCLOCK_Lpi2cDiv,
kCLOCK_LcdifPreDiv,
kCLOCK_CsiDiv }
*DIV control names for clock div setting.*

- enum clock_usb_src_t {
  kCLOCK_Usb480M = 0,
  kCLOCK_UsbSrcUnused = (int)0xFFFFFFFFU }
  *USB clock source definition.*
- enum clock_usb_phy_src_t { kCLOCK_Usbphy480M = 0 }
  *Source of the USB HS PHY.*
- enum _clock_pll_clk_src {
  kCLOCK_PllClkSrc24M = 0U,
  kCLOCK_PllSrcClkPN = 1U }
  *PLL clock source, bypass cloco source also.*
- enum clock_pll_t {
  kCLOCK_PllArm = CCM_ANALOG_TUPLE(PLL_ARM_OFFSET, CCM_ANALOG_PLL_ARM_ENABLE_SHIFT),
  kCLOCK_PllSys = CCM_ANALOG_TUPLE(PLL_SYS_OFFSET, CCM_ANALOG_PLL_SYS_ENABLE_SHIFT),
  kCLOCK_PllUsb1 = CCM_ANALOG_TUPLE(PLL_USB1_OFFSET, CCM_ANALOG_PLL_U-

SB1_ENABLE_SHIFT),
kCLOCK_PllAudio = CCM_ANALOG_TUPLE(PLL_AUDIO_OFFSET, CCM_ANALOG_PLL-_AUDIO_ENABLE_SHIFT),
kCLOCK_PllVideo = CCM_ANALOG_TUPLE(PLL_VIDEO_OFFSET, CCM_ANALOG_PLL_-VIDEO_ENABLE_SHIFT),
kCLOCK_PllEnet = CCM_ANALOG_TUPLE(PLL_ENET_OFFSET, CCM_ANALOG_PLL_E-NET_ENABLE_SHIFT),
kCLOCK_PllEnet25M = CCM_ANALOG_TUPLE(PLL_ENET_OFFSET, CCM_ANALOG_PL-L_ENET_ENET_25M_REF_EN_SHIFT),
kCLOCK_PllUsb2 = CCM_ANALOG_TUPLE(PLL_USB2_OFFSET, CCM_ANALOG_PLL_U-SB2_ENABLE_SHIFT) }
*PLL name.*

- enum clock_pfd_t {
  kCLOCK_Pfd0 = 0U,
  kCLOCK_Pfd1 = 1U,
  kCLOCK_Pfd2 = 2U,
  kCLOCK_Pfd3 = 3U }
  *PLL PFD name.*

## Functions

- static void CLOCK_SetMux (clock_mux_t mux, uint32_t value)
  *Set CCM MUX node to certain value.*
- static uint32_t CLOCK_GetMux (clock_mux_t mux)
  *Get CCM MUX value.*
- static void CLOCK_SetDiv (clock_div_t divider, uint32_t value)
  *Set CCM DIV node to certain value.*
- static uint32_t CLOCK_GetDiv (clock_div_t divider)
  *Get CCM DIV node value.*
- static void CLOCK_ControlGate (clock_ip_name_t name, clock_gate_value_t value)
  *Control the clock gate for specific IP.*
- static void CLOCK_EnableClock (clock_ip_name_t name)
  *Enable the clock for specific IP.*
- static void CLOCK_DisableClock (clock_ip_name_t name)
  *Disable the clock for specific IP.*
- static void CLOCK_SetMode (clock_mode_t mode)
  *Setting the low power mode that system will enter on next assertion of dsm_request signal.*
- static uint32_t CLOCK_GetOscFreq (void)
  *Gets the OSC clock frequency.*
- uint32_t CLOCK_GetAhbFreq (void)
  *Gets the AHB clock frequency.*
- uint32_t CLOCK_GetSemcFreq (void)
  *Gets the SEMC clock frequency.*
- uint32_t CLOCK_GetIpgFreq (void)
  *Gets the IPG clock frequency.*
- uint32_t CLOCK_GetPerClkFreq (void)
  *Gets the PER clock frequency.*
- uint32_t CLOCK_GetFreq (clock_name_t name)
  *Gets the clock frequency for a specific clock name.*

- static uint32_t CLOCK_GetCpuClkFreq (void)
  *Get the CCM CPU/core/system frequency.*
- bool CLOCK_EnableUsbhs0Clock (clock_usb_src_t src, uint32_t freq)
  *Enable USB HS clock.*
- bool CLOCK_EnableUsbhs1Clock (clock_usb_src_t src, uint32_t freq)
  *Enable USB HS clock.*

## Variables

- volatile uint32_t g_xtalFreq
  *External XTAL (24M OSC/SYSOSC) clock frequency.*
- volatile uint32_t g_rtcXtalFreq
  *External RTC XTAL (32K OSC) clock frequency.*

## Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  *CLOCK driver version 2.2.0.*
- #define **SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY** (600000000UL)
- #define **CCM_ANALOG_PLL_BYPASS_SHIFT** (16U)
- #define **CCM_ANALOG_PLL_BYPASS_CLK_SRC_MASK** (0xC000U)
- #define **CCM_ANALOG_PLL_BYPASS_CLK_SRC_SHIFT** (14U)

## OSC operations

- void CLOCK_InitExternalClk (bool bypassXtalOsc)
  *Initialize the external 24MHz clock.*
- void CLOCK_DeinitExternalClk (void)
  *Deinitialize the external 24MHz clock.*
- void CLOCK_SwitchOsc (clock_osc_t osc)
  *Switch the OSC.*
- static uint32_t CLOCK_GetRtcFreq (void)
  *Gets the RTC clock frequency.*
- static void CLOCK_SetXtalFreq (uint32_t freq)
  *Set the XTAL (24M OSC) frequency based on board setting.*
- static void CLOCK_SetRtcXtalFreq (uint32_t freq)
  *Set the RTC XTAL (32K OSC) frequency based on board setting.*
- void CLOCK_InitRcOsc24M (void)
  *Initialize the RC oscillator 24MHz clock.*
- void CLOCK_DeinitRcOsc24M (void)
  *Power down the RCOSC 24M clock.*

## PLL/PFD operations

- void CLOCK_DisableUsbhs1PhyPllClock (void)
  *Disable USB HS PHY PLL clock.*
- static void CLOCK_SetPllBypass (CCM_ANALOG_Type ∗base, clock_pll_t pll, bool bypass)
  *PLL bypass setting.*
- static bool CLOCK_IsPllBypassed (CCM_ANALOG_Type ∗base, clock_pll_t pll)
  *Check if PLL is bypassed.*
- static bool CLOCK_IsPllEnabled (CCM_ANALOG_Type ∗base, clock_pll_t pll)

*Check if PLL is enabled.*
- static void CLOCK_SetPllBypassRefClkSrc (CCM_ANALOG_Type *base, clock_pll_t pll, uint32_t src)
  *PLL bypass clock source setting.*
- static uint32_t CLOCK_GetPllBypassRefClk (CCM_ANALOG_Type *base, clock_pll_t pll)
  *Get PLL bypass clock value, it is PLL reference clock actually.*
- void CLOCK_InitArmPll (const clock_arm_pll_config_t *config)
  *Initialize the ARM PLL.*
- void CLOCK_DeinitArmPll (void)
  *De-initialize the ARM PLL.*
- void CLOCK_InitSysPll (const clock_sys_pll_config_t *config)
  *Initialize the System PLL.*
- void CLOCK_DeinitSysPll (void)
  *De-initialize the System PLL.*
- void CLOCK_InitUsb1Pll (const clock_usb_pll_config_t *config)
  *Initialize the USB1 PLL.*
- void CLOCK_DeinitUsb1Pll (void)
  *Deinitialize the USB1 PLL.*
- void CLOCK_InitUsb2Pll (const clock_usb_pll_config_t *config)
  *Initialize the USB2 PLL.*
- void CLOCK_DeinitUsb2Pll (void)
  *Deinitialize the USB2 PLL.*
- void CLOCK_InitAudioPll (const clock_audio_pll_config_t *config)
  *Initializes the Audio PLL.*
- void CLOCK_DeinitAudioPll (void)
  *De-initialize the Audio PLL.*
- void CLOCK_InitVideoPll (const clock_video_pll_config_t *config)
  *Initialize the video PLL.*
- void CLOCK_DeinitVideoPll (void)
  *De-initialize the Video PLL.*
- void CLOCK_InitEnetPll (const clock_enet_pll_config_t *config)
  *Initialize the ENET PLL.*
- void CLOCK_DeinitEnetPll (void)
  *Deinitialize the ENET PLL.*
- uint32_t CLOCK_GetPllFreq (clock_pll_t pll)
  *Get current PLL output frequency.*
- void CLOCK_InitSysPfd (clock_pfd_t pfd, uint8_t pfdFrac)
  *Initialize the System PLL PFD.*
- void CLOCK_DeinitSysPfd (clock_pfd_t pfd)
  *De-initialize the System PLL PFD.*
- void CLOCK_InitUsb1Pfd (clock_pfd_t pfd, uint8_t pfdFrac)
  *Initialize the USB1 PLL PFD.*
- void CLOCK_DeinitUsb1Pfd (clock_pfd_t pfd)
  *De-initialize the USB1 PLL PFD.*
- uint32_t CLOCK_GetSysPfdFreq (clock_pfd_t pfd)
  *Get current System PLL PFD output frequency.*
- uint32_t CLOCK_GetUsb1PfdFreq (clock_pfd_t pfd)
  *Get current USB1 PLL PFD output frequency.*
- bool CLOCK_EnableUsbhs0PhyPllClock (clock_usb_phy_src_t src, uint32_t freq)
  *Enable USB HS PHY PLL clock.*
- void CLOCK_DisableUsbhs0PhyPllClock (void)

*Disable USB HS PHY PLL clock.*
- bool CLOCK_EnableUsbhs1PhyPllClock (clock_usb_phy_src_t src, uint32_t freq)
  *Enable USB HS PHY PLL clock.*
- void SDK_DelayAtLeastUs (uint32_t delay_us)
  *Use DWT to delay at least for some time.*

## 49.2 Data Structure Documentation

### 49.2.1 struct clock_arm_pll_config_t

#### Data Fields

- uint32_t loopDivider
  *PLL loop divider.*
- uint8_t src
  *Pll clock source, reference _clock_pll_clk_src.*

##### 49.2.1.0.0.66 Field Documentation

##### 49.2.1.0.0.66.1 uint32_t clock_arm_pll_config_t::loopDivider

Valid range for divider value: 54-108. Fout=Fin∗loopDivider/2.

### 49.2.2 struct clock_usb_pll_config_t

#### Data Fields

- uint8_t loopDivider
  *PLL loop divider.*
- uint8_t src
  *Pll clock source, reference _clock_pll_clk_src.*

##### 49.2.2.0.0.67 Field Documentation

##### 49.2.2.0.0.67.1 uint8_t clock_usb_pll_config_t::loopDivider

0 - Fout=Fref∗20; 1 - Fout=Fref∗22

### 49.2.3 struct clock_sys_pll_config_t

#### Data Fields

- uint8_t loopDivider
  *PLL loop divider.*
- uint32_t numerator
  *30 bit numerator of fractional loop divider.*

- uint32_t denominator
  *30 bit denominator of fractional loop divider*
- uint8_t src
  *Pll clock source, reference _clock_pll_clk_src.*
- uint16_t ss_stop
  *Stop value to get frequency change.*
- uint8_t ss_enable
  *Enable spread spectrum modulation.*
- uint16_t ss_step
  *Step value to get frequency change step.*

#### 49.2.3.0.0.68 Field Documentation

##### 49.2.3.0.0.68.1 uint8_t clock_sys_pll_config_t::loopDivider

Intended to be 1 (528M). 0 - Fout=Fref∗20; 1 - Fout=Fref∗22

##### 49.2.3.0.0.68.2 uint32_t clock_sys_pll_config_t::numerator

##### 49.2.3.0.0.68.3 uint16_t clock_sys_pll_config_t::ss_stop

##### 49.2.3.0.0.68.4 uint16_t clock_sys_pll_config_t::ss_step

### 49.2.4 struct clock_audio_pll_config_t

## Data Fields

- uint8_t loopDivider
  *PLL loop divider.*
- uint8_t postDivider
  *Divider after the PLL, should only be 1, 2, 4, 8, 16.*
- uint32_t numerator
  *30 bit numerator of fractional loop divider.*
- uint32_t denominator
  *30 bit denominator of fractional loop divider*
- uint8_t src
  *Pll clock source, reference _clock_pll_clk_src.*

#### 49.2.4.0.0.69 Field Documentation

##### 49.2.4.0.0.69.1 uint8_t clock_audio_pll_config_t::loopDivider

Valid range for DIV_SELECT divider value: 27∼54.

#### 49.2.4.0.0.69.2 uint8_t clock_audio_pll_config_t::postDivider

#### 49.2.4.0.0.69.3 uint32_t clock_audio_pll_config_t::numerator

### 49.2.5 struct clock_video_pll_config_t

## Data Fields

- uint8_t loopDivider
  *PLL loop divider.*
- uint8_t postDivider
  *Divider after the PLL, should only be 1, 2, 4, 8, 16.*
- uint32_t numerator
  *30 bit numerator of fractional loop divider.*
- uint32_t denominator
  *30 bit denominator of fractional loop divider*
- uint8_t src
  *Pll clock source, reference _clock_pll_clk_src.*

#### 49.2.5.0.0.70 Field Documentation

#### 49.2.5.0.0.70.1 uint8_t clock_video_pll_config_t::loopDivider

Valid range for DIV_SELECT divider value: 27~54.

#### 49.2.5.0.0.70.2 uint8_t clock_video_pll_config_t::postDivider

#### 49.2.5.0.0.70.3 uint32_t clock_video_pll_config_t::numerator

### 49.2.6 struct clock_enet_pll_config_t

## Data Fields

- bool enableClkOutput
  *Power on and enable PLL clock output for ENET0 (ref_enetpll0).*
- bool enableClkOutput25M
  *Power on and enable PLL clock output for ENET1 (ref_enetpll1).*
- uint8_t loopDivider
  *Controls the frequency of the ENET0 reference clock.*
- uint8_t src
  *Pll clock source, reference _clock_pll_clk_src.*

#### 49.2.6.0.0.71 Field Documentation

#### 49.2.6.0.0.71.1 bool clock_enet_pll_config_t::enableClkOutput

#### 49.2.6.0.0.71.2 bool clock_enet_pll_config_t::enableClkOutput25M

#### 49.2.6.0.0.71.3 uint8_t clock_enet_pll_config_t::loopDivider

b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

## 49.3 Macro Definition Documentation

### 49.3.1 #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0

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.

### 49.3.2 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

### 49.3.3 #define ADC_CLOCKS

**Value:**

```
{                                                \
        kCLOCK_IpInvalid, kCLOCK_Adc1, kCLOCK_Adc2 \
    }
```

### 49.3.4 #define AOI_CLOCKS

**Value:**

```
{                                                \
        kCLOCK_IpInvalid, kCLOCK_Aoi1, kCLOCK_Aoi2 \
    }
```

### 49.3.5 #define BEE_CLOCKS

**Value:**

```
{               \
        kCLOCK_Bee \
    }
```

### 49.3.6 #define CMP_CLOCKS

**Value:**

```
{                                                                   \
        kCLOCK_IpInvalid, kCLOCK_Acmp1, kCLOCK_Acmp2,
      kCLOCK_Acmp3, kCLOCK_Acmp4 \
    }
```

### 49.3.7 #define CSI_CLOCKS

**Value:**

```
{               \
        kCLOCK_Csi \
    }
```

### 49.3.8 #define DCDC_CLOCKS

**Value:**

```
{                \
        kCLOCK_Dcdc \
    }
```

### 49.3.9 #define DCP_CLOCKS

**Value:**

```
{               \
        kCLOCK_Dcp \
    }
```

### 49.3.10 #define DMAMUX_CLOCKS

**Value:**

```
{                   \
        kCLOCK_Dma     \
    }
```

### 49.3.11 #define EDMA_CLOCKS

**Value:**

```
{                 \
        kCLOCK_Dma   \
    }
```

### 49.3.12 #define ENC_CLOCKS

**Value:**

```
{                                                                   \
        kCLOCK_IpInvalid, kCLOCK_Enc1, kCLOCK_Enc2,
      kCLOCK_Enc3, kCLOCK_Enc4 \
    }
```

### 49.3.13 #define ENET_CLOCKS

**Value:**

```
{                 \
        kCLOCK_Enet \
    }
```

### 49.3.14 #define EWM_CLOCKS

**Value:**

```
{                 \
        kCLOCK_Ewm0 \
    }
```

### 49.3.15 #define FLEXCAN_CLOCKS

**Value:**

```
{                                                \
        kCLOCK_IpInvalid, kCLOCK_Can1, kCLOCK_Can2 \
    }
```

### 49.3.16 #define FLEXCAN_PERIPH_CLOCKS

**Value:**

```
{                                                \
        kCLOCK_IpInvalid, kCLOCK_Can1S, kCLOCK_Can2S \
    }
```

### 49.3.17 #define FLEXIO_CLOCKS

**Value:**

```
{                                                \
        kCLOCK_IpInvalid, kCLOCK_Flexio1, kCLOCK_Flexio2 \
    }
```

### 49.3.18 #define FLEXRAM_CLOCKS

**Value:**

```
{                   \
        kCLOCK_FlexRam \
    }
```

### 49.3.19 #define FLEXSPI_CLOCKS

**Value:**

```
{                   \
        kCLOCK_FlexSpi \
    }
```

### 49.3.20 #define FLEXSPI_EXSC_CLOCKS

**Value:**

```
{                       \
        kCLOCK_FlexSpiExsc  \
    }
```

### 49.3.21 #define GPIO_CLOCKS

**Value:**

```
{                                                                                   \
        kCLOCK_IpInvalid, kCLOCK_Gpio1, kCLOCK_Gpio2,
      kCLOCK_Gpio3, kCLOCK_Gpio4, kCLOCK_Gpio5 \
    }
```

### 49.3.22 #define GPT_CLOCKS

**Value:**

```
{                                           \
        kCLOCK_IpInvalid, kCLOCK_Gpt1, kCLOCK_Gpt2 \
    }
```

### 49.3.23 #define KPP_CLOCKS

**Value:**

```
{               \
        kCLOCK_Kpp \
    }
```

### 49.3.24 #define LCDIF_CLOCKS

**Value:**

```
{                \
        kCLOCK_Lcd   \
    }
```

### 49.3.25 #define LCDIF_PERIPH_CLOCKS

**Value:**

```
{                       \
        kCLOCK_LcdPixel     \
    }
```

### 49.3.26 #define LPI2C_CLOCKS

**Value:**

```
{                                                                       \
        kCLOCK_IpInvalid, kCLOCK_Lpi2c1, kCLOCK_Lpi2c2,
      kCLOCK_Lpi2c3, kCLOCK_Lpi2c4 \
    }
```

### 49.3.27 #define LPSPI_CLOCKS

**Value:**

```
{                                                                       \
        kCLOCK_IpInvalid, kCLOCK_Lpspi1, kCLOCK_Lpspi2,
      kCLOCK_Lpspi3, kCLOCK_Lpspi4 \
    }
```

### 49.3.28 #define LPUART_CLOCKS

**Value:**

```
{                                                                                   \
        kCLOCK_IpInvalid, kCLOCK_Lpuart1, kCLOCK_Lpuart2,
      kCLOCK_Lpuart3, kCLOCK_Lpuart4, kCLOCK_Lpuart5,
      \
            kCLOCK_Lpuart6, kCLOCK_Lpuart7,
      kCLOCK_Lpuart8                                                \
    }
```

### 49.3.29 #define MQS_CLOCKS

**Value:**

```
{               \
        kCLOCK_Mqs \
    }
```

### 49.3.30 #define OCRAM_EXSC_CLOCKS

**Value:**

```
{                       \
        kCLOCK_OcramExsc  \
    }
```

### 49.3.31 #define PIT_CLOCKS

**Value:**

```
{               \
        kCLOCK_Pit \
    }
```

### 49.3.32 #define PWM_CLOCKS

**Value:**

```
{                                                                                   \
        {kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_IpInvalid}, \
            {kCLOCK_Pwm1, kCLOCK_Pwm1, kCLOCK_Pwm1, kCLOCK_Pwm1},
              \
            {kCLOCK_Pwm2, kCLOCK_Pwm2, kCLOCK_Pwm2, kCLOCK_Pwm2},
              \
            {kCLOCK_Pwm3, kCLOCK_Pwm3, kCLOCK_Pwm3, kCLOCK_Pwm3},
              \
        {
      \
              kCLOCK_Pwm4, kCLOCK_Pwm4,
      kCLOCK_Pwm4, kCLOCK_Pwm4                    \
        }                                                                                \
    }
```

### 49.3.33 #define PXP_CLOCKS

**Value:**

```
{               \
        kCLOCK_Pxp \
    }
```

### 49.3.34 #define RTWDOG_CLOCKS

**Value:**

```
{                   \
        kCLOCK_Wdog3  \
    }
```

### 49.3.35 #define SAI_CLOCKS

**Value:**

```
{                                                       \
        kCLOCK_IpInvalid, kCLOCK_Sai1, kCLOCK_Sai2,
      kCLOCK_Sai3 \
    }
```

### 49.3.36 #define SEMC_CLOCKS

**Value:**

```
{                 \
        kCLOCK_Semc \
    }
```

### 49.3.37 #define SEMC_EXSC_CLOCKS

**Value:**

```
{                     \
        kCLOCK_SemcExsc  \
    }
```

### 49.3.38 #define TMR_CLOCKS

**Value:**

```
{                                                                   \
        kCLOCK_IpInvalid, kCLOCK_Timer1, kCLOCK_Timer2,
      kCLOCK_Timer3, kCLOCK_Timer4 \
    }
```

### 49.3.39 #define TRNG_CLOCKS

**Value:**

```
{                \
        kCLOCK_Trng \
    }
```

### 49.3.40 #define TSC_CLOCKS

**Value:**

```
{               \
        kCLOCK_Tsc \
    }
```

### 49.3.41 #define WDOG_CLOCKS

**Value:**

```
{                                                  \
        kCLOCK_IpInvalid, kCLOCK_Wdog1, kCLOCK_Wdog2 \
    }
```

### 49.3.42 #define USDHC_CLOCKS

**Value:**

```
{                                                    \
        kCLOCK_IpInvalid, kCLOCK_Usdhc1, kCLOCK_Usdhc2 \
    }
```

### 49.3.43 #define SPDIF_CLOCKS

**Value:**

```
{                \
        kCLOCK_Spdif \
    }
```

### 49.3.44 #define XBARA_CLOCKS

**Value:**

```
{                \
        kCLOCK_Xbar1 \
    }
```

### 49.3.45 #define XBARB_CLOCKS

**Value:**

```
{                                                          \
        kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_Xbar2,
      kCLOCK_Xbar3 \
    }
```

### 49.3.46 #define kCLOCK_CoreSysClk kCLOCK_CpuClk

### 49.3.47 #define CLOCK_GetCoreSysClkFreq CLOCK_GetCpuClkFreq

## 49.4 Enumeration Type Documentation

### 49.4.1 enum clock_name_t

Enumerator

***kCLOCK_CpuClk*** CPU clock.
***kCLOCK_AhbClk*** AHB clock.
***kCLOCK_SemcClk*** SEMC clock.
***kCLOCK_IpgClk*** IPG clock.
***kCLOCK_PerClk*** PER clock.
***kCLOCK_OscClk*** OSC clock selected by PMU_LOWPWR_CTRL[OSC_SEL].
***kCLOCK_RtcClk*** RTC clock. (RTCCLK)
***kCLOCK_ArmPllClk*** ARMPLLCLK.
***kCLOCK_Usb1PllClk*** USB1PLLCLK.
***kCLOCK_Usb1PllPfd0Clk*** USB1PLLPDF0CLK.
***kCLOCK_Usb1PllPfd1Clk*** USB1PLLPFD1CLK.
***kCLOCK_Usb1PllPfd2Clk*** USB1PLLPFD2CLK.
***kCLOCK_Usb1PllPfd3Clk*** USB1PLLPFD3CLK.
***kCLOCK_Usb2PllClk*** USB2PLLCLK.
***kCLOCK_SysPllClk*** SYSPLLCLK.
***kCLOCK_SysPllPfd0Clk*** SYSPLLPDF0CLK.

*kCLOCK_SysPllPfd1Clk* SYSPLLPFD1CLK.
*kCLOCK_SysPllPfd2Clk* SYSPLLPFD2CLK.
*kCLOCK_SysPllPfd3Clk* SYSPLLPFD3CLK.
*kCLOCK_EnetPll0Clk* Enet PLLCLK ref_enetpll0.
*kCLOCK_EnetPll1Clk* Enet PLLCLK ref_enetpll1.
*kCLOCK_AudioPllClk* Audio PLLCLK.
*kCLOCK_VideoPllClk* Video PLLCLK.

### 49.4.2 enum clock_ip_name_t

Enumerator

*kCLOCK_Aips_tz1* CCGR0, CG0.
*kCLOCK_Aips_tz2* CCGR0, CG1.
*kCLOCK_Mqs* CCGR0, CG2.
*kCLOCK_FlexSpiExsc* CCGR0, CG3.
*kCLOCK_Sim_M_Main* CCGR0, CG4.
*kCLOCK_Dcp* CCGR0, CG5.
*kCLOCK_Lpuart3* CCGR0, CG6.
*kCLOCK_Can1* CCGR0, CG7.
*kCLOCK_Can1S* CCGR0, CG8.
*kCLOCK_Can2* CCGR0, CG9.
*kCLOCK_Can2S* CCGR0, CG10.
*kCLOCK_Trace* CCGR0, CG11.
*kCLOCK_Gpt2* CCGR0, CG12.
*kCLOCK_Gpt2S* CCGR0, CG13.
*kCLOCK_Lpuart2* CCGR0, CG14.
*kCLOCK_Gpio2* CCGR0, CG15.
*kCLOCK_Lpspi1* CCGR1, CG0.
*kCLOCK_Lpspi2* CCGR1, CG1.
*kCLOCK_Lpspi3* CCGR1, CG2.
*kCLOCK_Lpspi4* CCGR1, CG3.
*kCLOCK_Adc2* CCGR1, CG4.
*kCLOCK_Enet* CCGR1, CG5.
*kCLOCK_Pit* CCGR1, CG6.
*kCLOCK_Aoi2* CCGR1, CG7.
*kCLOCK_Adc1* CCGR1, CG8.
*kCLOCK_SemcExsc* CCGR1, CG9.
*kCLOCK_Gpt1* CCGR1, CG10.
*kCLOCK_Gpt1S* CCGR1, CG11.
*kCLOCK_Lpuart4* CCGR1, CG12.
*kCLOCK_Gpio1* CCGR1, CG13.
*kCLOCK_Csu* CCGR1, CG14.
*kCLOCK_Gpio5* CCGR1, CG15.

***kCLOCK_OcramExsc*** CCGR2, CG0.
***kCLOCK_Csi*** CCGR2, CG1.
***kCLOCK_IomuxcSnvs*** CCGR2, CG2.
***kCLOCK_Lpi2c1*** CCGR2, CG3.
***kCLOCK_Lpi2c2*** CCGR2, CG4.
***kCLOCK_Lpi2c3*** CCGR2, CG5.
***kCLOCK_Ocotp*** CCGR2, CG6.
***kCLOCK_Xbar3*** CCGR2, CG7.
***kCLOCK_Ipmux1*** CCGR2, CG8.
***kCLOCK_Ipmux2*** CCGR2, CG9.
***kCLOCK_Ipmux3*** CCGR2, CG10.
***kCLOCK_Xbar1*** CCGR2, CG11.
***kCLOCK_Xbar2*** CCGR2, CG12.
***kCLOCK_Gpio3*** CCGR2, CG13.
***kCLOCK_Lcd*** CCGR2, CG14.
***kCLOCK_Pxp*** CCGR2, CG15.
***kCLOCK_Flexio2*** CCGR3, CG0.
***kCLOCK_Lpuart5*** CCGR3, CG1.
***kCLOCK_Semc*** CCGR3, CG2.
***kCLOCK_Lpuart6*** CCGR3, CG3.
***kCLOCK_Aoi1*** CCGR3, CG4.
***kCLOCK_LcdPixel*** CCGR3, CG5.
***kCLOCK_Gpio4*** CCGR3, CG6.
***kCLOCK_Ewm0*** CCGR3, CG7.
***kCLOCK_Wdog1*** CCGR3, CG8.
***kCLOCK_FlexRam*** CCGR3, CG9.
***kCLOCK_Acmp1*** CCGR3, CG10.
***kCLOCK_Acmp2*** CCGR3, CG11.
***kCLOCK_Acmp3*** CCGR3, CG12.
***kCLOCK_Acmp4*** CCGR3, CG13.
***kCLOCK_Ocram*** CCGR3, CG14.
***kCLOCK_IomuxcSnvsGpr*** CCGR3, CG15.
***kCLOCK_Iomuxc*** CCGR4, CG1.
***kCLOCK_IomuxcGpr*** CCGR4, CG2.
***kCLOCK_Bee*** CCGR4, CG3.
***kCLOCK_SimM7*** CCGR4, CG4.
***kCLOCK_Tsc*** CCGR4, CG5.
***kCLOCK_SimM*** CCGR4, CG6.
***kCLOCK_SimEms*** CCGR4, CG7.
***kCLOCK_Pwm1*** CCGR4, CG8.
***kCLOCK_Pwm2*** CCGR4, CG9.
***kCLOCK_Pwm3*** CCGR4, CG10.
***kCLOCK_Pwm4*** CCGR4, CG11.
***kCLOCK_Enc1*** CCGR4, CG12.
***kCLOCK_Enc2*** CCGR4, CG13.

***kCLOCK_Enc3*** CCGR4, CG14.
***kCLOCK_Enc4*** CCGR4, CG15.
***kCLOCK_Rom*** CCGR5, CG0.
***kCLOCK_Flexio1*** CCGR5, CG1.
***kCLOCK_Wdog3*** CCGR5, CG2.
***kCLOCK_Dma*** CCGR5, CG3.
***kCLOCK_Kpp*** CCGR5, CG4.
***kCLOCK_Wdog2*** CCGR5, CG5.
***kCLOCK_Aips_tz4*** CCGR5, CG6.
***kCLOCK_Spdif*** CCGR5, CG7.
***kCLOCK_SimMain*** CCGR5, CG8.
***kCLOCK_Sai1*** CCGR5, CG9.
***kCLOCK_Sai2*** CCGR5, CG10.
***kCLOCK_Sai3*** CCGR5, CG11.
***kCLOCK_Lpuart1*** CCGR5, CG12.
***kCLOCK_Lpuart7*** CCGR5, CG13.
***kCLOCK_SnvsHp*** CCGR5, CG14.
***kCLOCK_SnvsLp*** CCGR5, CG15.
***kCLOCK_UsbOh3*** CCGR6, CG0.
***kCLOCK_Usdhc1*** CCGR6, CG1.
***kCLOCK_Usdhc2*** CCGR6, CG2.
***kCLOCK_Dcdc*** CCGR6, CG3.
***kCLOCK_Ipmux4*** CCGR6, CG4.
***kCLOCK_FlexSpi*** CCGR6, CG5.
***kCLOCK_Trng*** CCGR6, CG6.
***kCLOCK_Lpuart8*** CCGR6, CG7.
***kCLOCK_Timer4*** CCGR6, CG8.
***kCLOCK_Aips_tz3*** CCGR6, CG9.
***kCLOCK_SimPer*** CCGR6, CG10.
***kCLOCK_Anadig*** CCGR6, CG11.
***kCLOCK_Lpi2c4*** CCGR6, CG12.
***kCLOCK_Timer1*** CCGR6, CG13.
***kCLOCK_Timer2*** CCGR6, CG14.
***kCLOCK_Timer3*** CCGR6, CG15.

### 49.4.3 enum clock_osc_t

Enumerator

***kCLOCK_RcOsc*** On chip OSC.
***kCLOCK_XtalOsc*** 24M Xtal OSC

### 49.4.4 enum clock_gate_value_t

Enumerator

***kCLOCK_ClockNotNeeded*** Clock is off during all modes.
***kCLOCK_ClockNeededRun*** Clock is on in run mode, but off in WAIT and STOP modes.
***kCLOCK_ClockNeededRunWait*** Clock is on during all modes, except STOP mode.

### 49.4.5 enum clock_mode_t

Enumerator

***kCLOCK_ModeRun*** Remain in run mode.
***kCLOCK_ModeWait*** Transfer to wait mode.
***kCLOCK_ModeStop*** Transfer to stop mode.

### 49.4.6 enum clock_mux_t

These constants define the mux control names for clock mux setting.

- 0:7: REG offset to CCM_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

Enumerator

***kCLOCK_Pll3SwMux*** pll3_sw_clk mux name
***kCLOCK_PeriphMux*** periph mux name
***kCLOCK_SemcAltMux*** semc mux name
***kCLOCK_SemcMux*** semc mux name
***kCLOCK_PrePeriphMux*** pre-periph mux name
***kCLOCK_TraceMux*** trace mux name
***kCLOCK_PeriphClk2Mux*** periph clock2 mux name
***kCLOCK_LpspiMux*** lpspi mux name
***kCLOCK_FlexspiMux*** flexspi mux name
***kCLOCK_Usdhc2Mux*** usdhc2 mux name
***kCLOCK_Usdhc1Mux*** usdhc1 mux name
***kCLOCK_Sai3Mux*** sai3 mux name
***kCLOCK_Sai2Mux*** sai2 mux name
***kCLOCK_Sai1Mux*** sai1 mux name
***kCLOCK_PerclkMux*** perclk mux name
***kCLOCK_Flexio2Mux*** flexio2 mux name
***kCLOCK_CanMux*** can mux name

***kCLOCK_UartMux*** uart mux name
***kCLOCK_SpdifMux*** spdif mux name
***kCLOCK_Flexio1Mux*** flexio1 mux name
***kCLOCK_Lpi2cMux*** lpi2c mux name
***kCLOCK_LcdifPreMux*** lcdif pre mux name
***kCLOCK_CsiMux*** csi mux name

### 49.4.7 enum clock_div_t

These constants define div control names for clock div setting.

- 0:7: REG offset to CCM_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

Enumerator

***kCLOCK_ArmDiv*** core div name
***kCLOCK_PeriphClk2Div*** periph clock2 div name
***kCLOCK_SemcDiv*** semc div name
***kCLOCK_AhbDiv*** ahb div name
***kCLOCK_IpgDiv*** ipg div name
***kCLOCK_LpspiDiv*** lpspi div name
***kCLOCK_LcdifDiv*** lcdif div name
***kCLOCK_FlexspiDiv*** flexspi div name
***kCLOCK_PerclkDiv*** perclk div name
***kCLOCK_CanDiv*** can div name
***kCLOCK_TraceDiv*** trace div name
***kCLOCK_Usdhc2Div*** usdhc2 div name
***kCLOCK_Usdhc1Div*** usdhc1 div name
***kCLOCK_UartDiv*** uart div name
***kCLOCK_Flexio2Div*** flexio2 pre div name
***kCLOCK_Sai3PreDiv*** sai3 pre div name
***kCLOCK_Sai3Div*** sai3 div name
***kCLOCK_Flexio2PreDiv*** sai3 pre div name
***kCLOCK_Sai1PreDiv*** sai1 pre div name
***kCLOCK_Sai1Div*** sai1 div name
***kCLOCK_Sai2PreDiv*** sai2 pre div name
***kCLOCK_Sai2Div*** sai2 div name
***kCLOCK_Spdif0PreDiv*** spdif pre div name
***kCLOCK_Spdif0Div*** spdif div name
***kCLOCK_Flexio1PreDiv*** flexio1 pre div name
***kCLOCK_Flexio1Div*** flexio1 div name
***kCLOCK_Lpi2cDiv*** lpi2c div name

***kCLOCK_LcdifPreDiv*** lcdif pre div name
***kCLOCK_CsiDiv*** csi div name

### 49.4.8 enum clock_usb_src_t

Enumerator

***kCLOCK_Usb480M*** Use 480M.
***kCLOCK_UsbSrcUnused*** Used when the function does not care the clock source.

### 49.4.9 enum clock_usb_phy_src_t

Enumerator

***kCLOCK_Usbphy480M*** Use 480M.

### 49.4.10 enum _clock_pll_clk_src

Enumerator

***kCLOCK_PllClkSrc24M*** Pll clock source 24M.
***kCLOCK_PllSrcClkPN*** Pll clock source CLK1_P and CLK1_N.

### 49.4.11 enum clock_pll_t

Enumerator

***kCLOCK_PllArm*** PLL ARM.
***kCLOCK_PllSys*** PLL SYS.
***kCLOCK_PllUsb1*** PLL USB1.
***kCLOCK_PllAudio*** PLL Audio.
***kCLOCK_PllVideo*** PLL Video.
***kCLOCK_PllEnet*** PLL Enet0.
***kCLOCK_PllEnet25M*** PLL Enet1.
***kCLOCK_PllUsb2*** PLL USB2.

### 49.4.12 enum clock_pfd_t

Enumerator

***kCLOCK_Pfd0*** PLL PFD0.
***kCLOCK_Pfd1*** PLL PFD1.
***kCLOCK_Pfd2*** PLL PFD2.
***kCLOCK_Pfd3*** PLL PFD3.

## 49.5 Function Documentation

### 49.5.1 static void CLOCK_SetMux ( clock_mux_t *mux*, uint32_t *value* ) [inline], [static]

Parameters

| | |
|---|---|
| *mux* | Which mux node to set, see clock_mux_t. |
| *value* | Clock mux value to set, different mux has different value range. |

### 49.5.2 static uint32_t CLOCK_GetMux ( clock_mux_t *mux* ) [inline], [static]

Parameters

| | |
|---|---|
| *mux* | Which mux node to get, see clock_mux_t. |

Returns

Clock mux value.

### 49.5.3 static void CLOCK_SetDiv ( clock_div_t *divider*, uint32_t *value* ) [inline], [static]

Parameters

| | |
|---|---|
| *divider* | Which div node to set, see clock_div_t. |
| *value* | Clock div value to set, different divider has different value range. |

### 49.5.4 static uint32_t CLOCK_GetDiv ( clock_div_t *divider* ) [inline], [static]

Parameters

| | |
|---|---|
| *divider* | Which div node to get, see clock_div_t. |

### 49.5.5 static void CLOCK_ControlGate ( clock_ip_name_t *name,* clock_gate_value_t *value* ) [inline], [static]

Parameters

| | |
|---|---|
| *name* | Which clock to enable, see clock_ip_name_t. |
| *value* | Clock gate value to set, see clock_gate_value_t. |

### 49.5.6 static void CLOCK_EnableClock ( clock_ip_name_t *name* ) [inline], [static]

Parameters

| | |
|---|---|
| *name* | Which clock to enable, see clock_ip_name_t. |

### 49.5.7 static void CLOCK_DisableClock ( clock_ip_name_t *name* ) [inline], [static]

Parameters

| | |
|---|---|
| *name* | Which clock to disable, see clock_ip_name_t. |

### 49.5.8 static void CLOCK_SetMode ( clock_mode_t *mode* ) [inline], [static]

Parameters

| | |
|---|---|
| *mode* | Which mode to enter, see clock_mode_t. |

### 49.5.9 static uint32_t CLOCK_GetOscFreq ( void ) [inline], [static]

This function will return the external XTAL OSC frequency if it is selected as the source of OSC, otherwise internal 24MHz RC OSC frequency will be returned.

Parameters

| | |
|---|---|
| *osc* | OSC type to get frequency. |

Returns

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

### 49.5.10 uint32_t CLOCK_GetAhbFreq ( void )

Returns

The AHB clock frequency value in hertz.

### 49.5.11 uint32_t CLOCK_GetSemcFreq ( void )

Returns

The SEMC clock frequency value in hertz.

### 49.5.12 uint32_t CLOCK_GetIpgFreq ( void )

Returns

The IPG clock frequency value in hertz.

### 49.5.13 uint32_t CLOCK_GetPerClkFreq ( void )

Returns

The PER clock frequency value in hertz.

### 49.5.14 uint32_t CLOCK_GetFreq ( clock_name_t *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

### 49.5.15 static uint32_t CLOCK_GetCpuClkFreq ( void ) [inline], [static]

Returns

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

### 49.5.16 void CLOCK_InitExternalClk ( bool *bypassXtalOsc* )

This function supports two modes:

1. Use external crystal oscillator.
2. Bypass the external crystal oscillator, using input source clock directly.

After this function, please call CLOCK_SetXtal0Freq to inform clock driver the external clock frequency.

Parameters

| | |
|---|---|
| *bypassXtalOsc* | Pass in true to bypass the external crystal oscillator. |

Note

This device does not support bypass external crystal oscillator, so the input parameter should always be false.

### 49.5.17 void CLOCK_DeinitExternalClk ( void )

This function disables the external 24MHz clock.

After this function, please call CLOCK_SetXtal0Freq to set external clock frequency to 0.

### 49.5.18 void CLOCK_SwitchOsc ( clock_osc_t *osc* )

This function switches the OSC source for SoC.

Parameters

| | |
|---|---|
| *osc* | OSC source to switch to. |

### 49.5.19 static uint32_t CLOCK_GetRtcFreq ( void ) [inline], [static]

Returns

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

### 49.5.20 static void CLOCK_SetXtalFreq ( uint32_t *freq* ) [inline], [static]

Parameters

| | |
|---|---|
| *freq* | The XTAL input clock frequency in Hz. |

### 49.5.21 static void CLOCK_SetRtcXtalFreq ( uint32_t *freq* ) [inline], [static]

Parameters

| | |
|---|---|
| *freq* | The RTC XTAL input clock frequency in Hz. |

### 49.5.22 bool CLOCK_EnableUsbhs0Clock ( clock_usb_src_t *src*, uint32_t *freq* )

This function only enables the access to USB HS prepheral, upper layer should first call the CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

| | |
|---|---|
| *src* | USB HS does not care about the clock source, here must be kCLOCK_UsbSrcUnused. |

| | |
|---|---|
| *freq* | USB HS does not care about the clock source, so this parameter is ignored. |

Return values

| | |
|---|---|
| *true* | The clock is set successfully. |
| *false* | The clock source is invalid to get proper USB HS clock. |

### 49.5.23 bool CLOCK_EnableUsbhs1Clock ( clock_usb_src_t *src*, uint32_t *freq* )

This function only enables the access to USB HS prepheral, upper layer should first call the CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

| | |
|---|---|
| *src* | USB HS does not care about the clock source, here must be kCLOCK_UsbSrcUnused. |
| *freq* | USB HS does not care about the clock source, so this parameter is ignored. |

Return values

| | |
|---|---|
| *true* | The clock is set successfully. |
| *false* | The clock source is invalid to get proper USB HS clock. |

### 49.5.24 void CLOCK_DisableUsbhs1PhyPllClock ( void )

This function disables USB HS PHY PLL clock.

### 49.5.25 static void CLOCK_SetPllBypass ( CCM_ANALOG_Type * *base*, clock_pll_t *pll*, bool *bypass* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CCM_ANALOG base pointer. |
| *pll* | PLL control name (see ccm_analog_pll_control_t enumeration) |
| *bypass* | Bypass the PLL.<br>• true: Bypass the PLL.<br>• false:Not bypass the PLL. |

### 49.5.26 static bool CLOCK_IsPllBypassed ( CCM_ANALOG_Type * *base,* clock_pll_t *pll* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CCM_ANALOG base pointer. |
| *pll* | PLL control name (see ccm_analog_pll_control_t enumeration) |

Returns

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

### 49.5.27 static bool CLOCK_IsPllEnabled ( CCM_ANALOG_Type * *base,* clock_pll_t *pll* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | CCM_ANALOG base pointer. |
| *pll* | PLL control name (see ccm_analog_pll_control_t enumeration) |

Returns

PLL bypass status.
- true: The PLL is enabled.
- false: The PLL is not enabled.

### 49.5.28 static void CLOCK_SetPllBypassRefClkSrc ( CCM_ANALOG_Type * *base,* clock_pll_t *pll,* uint32_t *src* ) [inline], [static]

Note: change the bypass clock source also change the pll reference clock source.

Parameters

| | |
|---|---|
| *base* | CCM_ANALOG base pointer. |
| *pll* | PLL control name (see ccm_analog_pll_control_t enumeration) |

| | |
|---|---|
| *src* | Bypass clock source, reference _clock_pll_bypass_clk_src. |

### 49.5.29 static uint32_t CLOCK_GetPllBypassRefClk ( CCM_ANALOG_Type * *base*, clock_pll_t *pll* ) [inline], [static]

If CLOCK1_P,CLOCK1_N is choose as the pll bypass clock source, please implement the CLKPN_FREQ define, otherwise 0 will be returned.

Parameters

| | |
|---|---|
| *base* | CCM_ANALOG base pointer. |
| *pll* | PLL control name (see ccm_analog_pll_control_t enumeration) |

Return values

| | |
|---|---|
| *bypass* | reference clock frequency value. |

### 49.5.30 void CLOCK_InitArmPll ( const clock_arm_pll_config_t * *config* )

This function initialize the ARM PLL with specific settings

Parameters

| | |
|---|---|
| *config* | configuration to set to PLL. |

### 49.5.31 void CLOCK_InitSysPll ( const clock_sys_pll_config_t * *config* )

This function initializes the System PLL with specific settings

Parameters

| | |
|---|---|
| *config* | Configuration to set to PLL. |

### 49.5.32 void CLOCK_InitUsb1Pll ( const clock_usb_pll_config_t * *config* )

This function initializes the USB1 PLL with specific settings

Parameters

| | |
|---|---|
| *config* | Configuration to set to PLL. |

### 49.5.33 void CLOCK_InitUsb2Pll ( const clock_usb_pll_config_t * *config* )

This function initializes the USB2 PLL with specific settings

Parameters

| | |
|---|---|
| *config* | Configuration to set to PLL. |

### 49.5.34 void CLOCK_InitAudioPll ( const clock_audio_pll_config_t * *config* )

This function initializes the Audio PLL with specific settings

Parameters

| | |
|---|---|
| *config* | Configuration to set to PLL. |

### 49.5.35 void CLOCK_InitVideoPll ( const clock_video_pll_config_t * *config* )

This function configures the Video PLL with specific settings

Parameters

| | |
|---|---|
| *config* | configuration to set to PLL. |

### 49.5.36 void CLOCK_InitEnetPll ( const clock_enet_pll_config_t * *config* )

This function initializes the ENET PLL with specific settings.

Parameters

| | |
|---|---|
| *config* | Configuration to set to PLL. |

### 49.5.37 void CLOCK_DeinitEnetPll ( void )

This function disables the ENET PLL.

### 49.5.38 uint32_t CLOCK_GetPllFreq ( clock_pll_t *pll* )

This function get current output frequency of specific PLL

Parameters

| | |
|---|---|
| *pll* | pll name to get frequency. |

Returns

The PLL output frequency in hertz.

### 49.5.39 void CLOCK_InitSysPfd ( clock_pfd_t *pfd,* uint8_t *pfdFrac* )

This function initializes the System PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

Parameters

| | |
|---|---|
| *pfd* | Which PFD clock to enable. |
| *pfdFrac* | The PFD FRAC value. |

Note

It is recommended that PFD settings are kept between 12-35.

### 49.5.40 void CLOCK_DeinitSysPfd ( clock_pfd_t *pfd* )

This function disables the System PLL PFD.

Parameters

| | |
|---|---|
| *pfd* | Which PFD clock to disable. |

### 49.5.41 void CLOCK_InitUsb1Pfd ( clock_pfd_t *pfd,* uint8_t *pfdFrac* )

This function initializes the USB1 PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

Parameters

| | |
|---|---|
| *pfd* | Which PFD clock to enable. |
| *pfdFrac* | The PFD FRAC value. |

Note

It is recommended that PFD settings are kept between 12-35.

### 49.5.42 void CLOCK_DeinitUsb1Pfd ( clock_pfd_t *pfd* )

This function disables the USB1 PLL PFD.

Parameters

| | |
|---|---|
| *pfd* | Which PFD clock to disable. |

### 49.5.43 uint32_t CLOCK_GetSysPfdFreq ( clock_pfd_t *pfd* )

This function get current output frequency of specific System PLL PFD

Parameters

| | |
|---|---|
| *pfd* | pfd name to get frequency. |

Returns

The PFD output frequency in hertz.

### 49.5.44 uint32_t CLOCK_GetUsb1PfdFreq ( clock_pfd_t *pfd* )

This function get current output frequency of specific USB1 PLL PFD

Parameters

| | |
|---|---|
| *pfd* | pfd name to get frequency. |

Returns

The PFD output frequency in hertz.

### 49.5.45 bool CLOCK_EnableUsbhs0PhyPllClock ( clock_usb_phy_src_t *src*, uint32_t *freq* )

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

| | |
|---|---|
| *src* | USB HS PHY PLL clock source. |
| *freq* | The frequency specified by src. |

Return values

| | |
|---|---|
| *true* | The clock is set successfully. |
| *false* | The clock source is invalid to get proper USB HS clock. |

### 49.5.46 void CLOCK_DisableUsbhs0PhyPllClock ( void )

This function disables USB HS PHY PLL clock.

### 49.5.47 bool CLOCK_EnableUsbhs1PhyPllClock ( clock_usb_phy_src_t *src*, uint32_t *freq* )

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

| | |
|---|---|
| *src* | USB HS PHY PLL clock source. |
| *freq* | The frequency specified by src. |

Return values

| | |
|---|---|
| *true* | The clock is set successfully. |
| *false* | The clock source is invalid to get proper USB HS clock. |

### 49.5.48 void SDK_DelayAtLeastUs ( uint32_t *delay_us* )

Please note that, this API will calculate the microsecond period with the maximum supported CPU frequency, so this API will only delay for at least the given microseconds, if precise delay count was needed, please implement a new timer count to achieve this function.

Parameters

| | |
|---|---|
| *delay_us* | Delay time in unit of microsecond. |

## 49.6 Variable Documentation

### 49.6.1 volatile uint32_t g_xtalFreq

The XTAL (24M OSC/SYSOSC) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetXtalFreq to set the value in to clock driver. For example, if XTAL is 24MHz,

```
* CLOCK_InitExternalClk(false); // Setup the 24M OSC/SYSOSC
* CLOCK_SetXtalFreq(240000000); // Set the XTAL value to clock driver.
*
```

### 49.6.2 volatile uint32_t g_rtcXtalFreq

The RTC XTAL (32K OSC) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetRtcXtalFreq to set the value in to clock driver.

# Chapter 50
# IOMUXC: Input/Output Multiplexing Controller

## 50.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Input/Output Multiplexing Controller.

The IOMUXC, working in conjunction with the IOMUX, enables the chip to share one pad for multiple signals from different peripheral interfaces.

## Files

- file fsl_iomuxc.h

## Driver version

- #define FSL_IOMUXC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  *IOMUXC driver version 2.0.0.*

## Pin function ID

- enum **iomuxc_gpr_mode_t**
- enum **iomuxc_gpr_saimclk_t**
- enum **iomuxc_mqs_pwm_oversample_rate_t**
- #define IOMUXC_SNVS_WAKEUP_GPIO5_IO00 0x400A8000U, 0x5U, 0, 0, 0x400A8018U
  *The pin function ID is a tuple of <muxRegister muxmode="" inputregister="" inputdaisy="" configregister>=""*
- #define **IOMUXC_SNVS_WAKEUP_NMI_GLUE_NMI** 0x400A8000U, 0x7U, 0x401F8568U, 0x1U, 0x400A8018U
- #define **IOMUXC_SNVS_PMIC_ON_REQ_SNVS_LP_PMIC_ON_REQ** 0x400A8004U, 0x0U, 0, 0, 0x400A801CU
- #define **IOMUXC_SNVS_PMIC_ON_REQ_GPIO5_IO01** 0x400A8004U, 0x5U, 0, 0, 0x400A801CU
- #define **IOMUXC_SNVS_PMIC_STBY_REQ_CCM_PMIC_VSTBY_REQ** 0x400A8008U, 0x0U, 0, 0, 0x400A8020U
- #define **IOMUXC_SNVS_PMIC_STBY_REQ_GPIO5_IO02** 0x400A8008U, 0x5U, 0, 0, 0x400A8020U
- #define **IOMUXC_SNVS_TEST_MODE** 0, 0, 0, 0, 0x400A800CU
- #define **IOMUXC_SNVS_POR_B** 0, 0, 0, 0, 0x400A8010U
- #define **IOMUXC_SNVS_ONOFF** 0, 0, 0, 0, 0x400A8014U
- #define **IOMUXC_GPIO_EMC_00_SEMC_DATA00** 0x401F8014U, 0x0U, 0, 0, 0x401F8204U
- #define **IOMUXC_GPIO_EMC_00_FLEXPWM4_PWMA00** 0x401F8014U, 0x1U, 0x401F8494U, 0x0U, 0x401F8204U
- #define **IOMUXC_GPIO_EMC_00_LPSPI2_SCK** 0x401F8014U, 0x2U, 0x401F8500U, 0x1U, 0x401F8204U
- #define **IOMUXC_GPIO_EMC_00_XBAR1_XBAR_IN02** 0x401F8014U, 0x3U, 0x401F860CU, 0x0U, 0x401F8204U

- #define **IOMUXC_GPIO_EMC_00_FLEXIO1_FLEXIO00** 0x401F8014U, 0x4U, 0, 0, 0x401-F8204U
- #define **IOMUXC_GPIO_EMC_00_GPIO4_IO00** 0x401F8014U, 0x5U, 0, 0, 0x401F8204U
- #define **IOMUXC_GPIO_EMC_01_SEMC_DATA01** 0x401F8018U, 0x0U, 0, 0, 0x401F8208U
- #define **IOMUXC_GPIO_EMC_01_FLEXPWM4_PWMB00** 0x401F8018U, 0x1U, 0, 0, 0x401-F8208U
- #define **IOMUXC_GPIO_EMC_01_LPSPI2_PCS0** 0x401F8018U, 0x2U, 0x401F84FCU, 0x1U, 0x401F8208U
- #define **IOMUXC_GPIO_EMC_01_XBAR1_IN03** 0x401F8018U, 0x3U, 0x401F8610U, 0x0U, 0x401F8208U
- #define **IOMUXC_GPIO_EMC_01_FLEXIO1_FLEXIO01** 0x401F8018U, 0x4U, 0, 0, 0x401-F8208U
- #define **IOMUXC_GPIO_EMC_01_GPIO4_IO01** 0x401F8018U, 0x5U, 0, 0, 0x401F8208U
- #define **IOMUXC_GPIO_EMC_02_SEMC_DATA02** 0x401F801CU, 0x0U, 0, 0, 0x401F820CU
- #define **IOMUXC_GPIO_EMC_02_FLEXPWM4_PWMA01** 0x401F801CU, 0x1U, 0x401-F8498U, 0x0U, 0x401F820CU
- #define **IOMUXC_GPIO_EMC_02_LPSPI2_SDO** 0x401F801CU, 0x2U, 0x401F8508U, 0x1U, 0x401F820CU
- #define **IOMUXC_GPIO_EMC_02_XBAR1_INOUT04** 0x401F801CU, 0x3U, 0x401F8614U, 0x0U, 0x401F820CU
- #define **IOMUXC_GPIO_EMC_02_FLEXIO1_FLEXIO02** 0x401F801CU, 0x4U, 0, 0, 0x401-F820CU
- #define **IOMUXC_GPIO_EMC_02_GPIO4_IO02** 0x401F801CU, 0x5U, 0, 0, 0x401F820CU
- #define **IOMUXC_GPIO_EMC_03_SEMC_DATA03** 0x401F8020U, 0x0U, 0, 0, 0x401F8210U
- #define **IOMUXC_GPIO_EMC_03_FLEXPWM4_PWMB01** 0x401F8020U, 0x1U, 0, 0, 0x401-F8210U
- #define **IOMUXC_GPIO_EMC_03_LPSPI2_SDI** 0x401F8020U, 0x2U, 0x401F8504U, 0x1U, 0x401F8210U
- #define **IOMUXC_GPIO_EMC_03_XBAR1_INOUT05** 0x401F8020U, 0x3U, 0x401F8618U, 0x0U, 0x401F8210U
- #define **IOMUXC_GPIO_EMC_03_FLEXIO1_FLEXIO03** 0x401F8020U, 0x4U, 0, 0, 0x401-F8210U
- #define **IOMUXC_GPIO_EMC_03_GPIO4_IO03** 0x401F8020U, 0x5U, 0, 0, 0x401F8210U
- #define **IOMUXC_GPIO_EMC_04_SEMC_DATA04** 0x401F8024U, 0x0U, 0, 0, 0x401F8214U
- #define **IOMUXC_GPIO_EMC_04_FLEXPWM4_PWMA02** 0x401F8024U, 0x1U, 0x401-F849CU, 0x0U, 0x401F8214U
- #define **IOMUXC_GPIO_EMC_04_SAI2_TX_DATA** 0x401F8024U, 0x2U, 0, 0, 0x401F8214U
- #define **IOMUXC_GPIO_EMC_04_XBAR1_INOUT06** 0x401F8024U, 0x3U, 0x401F861CU, 0x0U, 0x401F8214U
- #define **IOMUXC_GPIO_EMC_04_FLEXIO1_FLEXIO04** 0x401F8024U, 0x4U, 0, 0, 0x401-F8214U
- #define **IOMUXC_GPIO_EMC_04_GPIO4_IO04** 0x401F8024U, 0x5U, 0, 0, 0x401F8214U
- #define **IOMUXC_GPIO_EMC_05_SEMC_DATA05** 0x401F8028U, 0x0U, 0, 0, 0x401F8218U
- #define **IOMUXC_GPIO_EMC_05_FLEXPWM4_PWMB02** 0x401F8028U, 0x1U, 0, 0, 0x401-F8218U
- #define **IOMUXC_GPIO_EMC_05_SAI2_TX_SYNC** 0x401F8028U, 0x2U, 0x401F85C4U, 0x0U, 0x401F8218U
- #define **IOMUXC_GPIO_EMC_05_XBAR1_INOUT07** 0x401F8028U, 0x3U, 0x401F8620U, 0x0U, 0x401F8218U
- #define **IOMUXC_GPIO_EMC_05_FLEXIO1_FLEXIO05** 0x401F8028U, 0x4U, 0, 0, 0x401-F8218U

- #define **IOMUXC_GPIO_EMC_05_GPIO4_IO05** 0x401F8028U, 0x5U, 0, 0, 0x401F8218U
- #define **IOMUXC_GPIO_EMC_06_SEMC_DATA06** 0x401F802CU, 0x0U, 0, 0, 0x401F821CU
- #define **IOMUXC_GPIO_EMC_06_FLEXPWM2_PWMA00** 0x401F802CU, 0x1U, 0x401F8478U, 0x0U, 0x401F821CU
- #define **IOMUXC_GPIO_EMC_06_SAI2_TX_BCLK** 0x401F802CU, 0x2U, 0x401F85C0U, 0x0U, 0x401F821CU
- #define **IOMUXC_GPIO_EMC_06_XBAR1_INOUT08** 0x401F802CU, 0x3U, 0x401F8624U, 0x0U, 0x401F821CU
- #define **IOMUXC_GPIO_EMC_06_FLEXIO1_FLEXIO06** 0x401F802CU, 0x4U, 0, 0, 0x401F821CU
- #define **IOMUXC_GPIO_EMC_06_GPIO4_IO06** 0x401F802CU, 0x5U, 0, 0, 0x401F821CU
- #define **IOMUXC_GPIO_EMC_07_SEMC_DATA07** 0x401F8030U, 0x0U, 0, 0, 0x401F8220U
- #define **IOMUXC_GPIO_EMC_07_FLEXPWM2_PWMB00** 0x401F8030U, 0x1U, 0x401F8488U, 0x0U, 0x401F8220U
- #define **IOMUXC_GPIO_EMC_07_SAI2_MCLK** 0x401F8030U, 0x2U, 0x401F85B0U, 0x0U, 0x401F8220U
- #define **IOMUXC_GPIO_EMC_07_XBAR1_INOUT09** 0x401F8030U, 0x3U, 0x401F8628U, 0x0U, 0x401F8220U
- #define **IOMUXC_GPIO_EMC_07_FLEXIO1_FLEXIO07** 0x401F8030U, 0x4U, 0, 0, 0x401F8220U
- #define **IOMUXC_GPIO_EMC_07_GPIO4_IO07** 0x401F8030U, 0x5U, 0, 0, 0x401F8220U
- #define **IOMUXC_GPIO_EMC_08_SEMC_DM00** 0x401F8034U, 0x0U, 0, 0, 0x401F8224U
- #define **IOMUXC_GPIO_EMC_08_FLEXPWM2_PWMA01** 0x401F8034U, 0x1U, 0x401F847CU, 0x0U, 0x401F8224U
- #define **IOMUXC_GPIO_EMC_08_SAI2_RX_DATA** 0x401F8034U, 0x2U, 0x401F85B8U, 0x0U, 0x401F8224U
- #define **IOMUXC_GPIO_EMC_08_XBAR1_INOUT17** 0x401F8034U, 0x3U, 0x401F862CU, 0x0U, 0x401F8224U
- #define **IOMUXC_GPIO_EMC_08_FLEXIO1_FLEXIO08** 0x401F8034U, 0x4U, 0, 0, 0x401F8224U
- #define **IOMUXC_GPIO_EMC_08_GPIO4_IO08** 0x401F8034U, 0x5U, 0, 0, 0x401F8224U
- #define **IOMUXC_GPIO_EMC_09_SEMC_ADDR00** 0x401F8038U, 0x0U, 0, 0, 0x401F8228U
- #define **IOMUXC_GPIO_EMC_09_FLEXPWM2_PWMB01** 0x401F8038U, 0x1U, 0x401F848CU, 0x0U, 0x401F8228U
- #define **IOMUXC_GPIO_EMC_09_SAI2_RX_SYNC** 0x401F8038U, 0x2U, 0x401F85BCU, 0x0U, 0x401F8228U
- #define **IOMUXC_GPIO_EMC_09_FLEXCAN2_TX** 0x401F8038U, 0x3U, 0, 0, 0x401F8228U
- #define **IOMUXC_GPIO_EMC_09_FLEXIO1_FLEXIO09** 0x401F8038U, 0x4U, 0, 0, 0x401F8228U
- #define **IOMUXC_GPIO_EMC_09_GPIO4_IO09** 0x401F8038U, 0x5U, 0, 0, 0x401F8228U
- #define **IOMUXC_GPIO_EMC_10_SEMC_ADDR01** 0x401F803CU, 0x0U, 0, 0, 0x401F822CU
- #define **IOMUXC_GPIO_EMC_10_FLEXPWM2_PWMA02** 0x401F803CU, 0x1U, 0x401F8480U, 0x0U, 0x401F822CU
- #define **IOMUXC_GPIO_EMC_10_SAI2_RX_BCLK** 0x401F803CU, 0x2U, 0x401F85B4U, 0x0U, 0x401F822CU
- #define **IOMUXC_GPIO_EMC_10_FLEXCAN2_RX** 0x401F803CU, 0x3U, 0x401F8450U, 0x0U, 0x401F822CU
- #define **IOMUXC_GPIO_EMC_10_FLEXIO1_FLEXIO10** 0x401F803CU, 0x4U, 0, 0, 0x401F822CU
- #define **IOMUXC_GPIO_EMC_10_GPIO4_IO10** 0x401F803CU, 0x5U, 0, 0, 0x401F822CU
- #define **IOMUXC_GPIO_EMC_11_SEMC_ADDR02** 0x401F8040U, 0x0U, 0, 0, 0x401F8230U

- #define **IOMUXC_GPIO_EMC_11_FLEXPWM2_PWMB02** 0x401F8040U, 0x1U, 0x401F8490U, 0x0U, 0x401F8230U
- #define **IOMUXC_GPIO_EMC_11_LPI2C4_SDA** 0x401F8040U, 0x2U, 0x401F84E8U, 0x0U, 0x401F8230U
- #define **IOMUXC_GPIO_EMC_11_USDHC2_RESET_B** 0x401F8040U, 0x3U, 0, 0, 0x401F8230U
- #define **IOMUXC_GPIO_EMC_11_FLEXIO1_FLEXIO11** 0x401F8040U, 0x4U, 0, 0, 0x401F8230U
- #define **IOMUXC_GPIO_EMC_11_GPIO4_IO11** 0x401F8040U, 0x5U, 0, 0, 0x401F8230U
- #define **IOMUXC_GPIO_EMC_12_SEMC_ADDR03** 0x401F8044U, 0x0U, 0, 0, 0x401F8234U
- #define **IOMUXC_GPIO_EMC_12_XBAR1_IN24** 0x401F8044U, 0x1U, 0x401F8640U, 0x0U, 0x401F8234U
- #define **IOMUXC_GPIO_EMC_12_LPI2C4_SCL** 0x401F8044U, 0x2U, 0x401F84E4U, 0x0U, 0x401F8234U
- #define **IOMUXC_GPIO_EMC_12_USDHC1_WP** 0x401F8044U, 0x3U, 0x401F85D8U, 0x0U, 0x401F8234U
- #define **IOMUXC_GPIO_EMC_12_FLEXPWM1_PWMA03** 0x401F8044U, 0x4U, 0x401F8454U, 0x1U, 0x401F8234U
- #define **IOMUXC_GPIO_EMC_12_GPIO4_IO12** 0x401F8044U, 0x5U, 0, 0, 0x401F8234U
- #define **IOMUXC_GPIO_EMC_13_SEMC_ADDR04** 0x401F8048U, 0x0U, 0, 0, 0x401F8238U
- #define **IOMUXC_GPIO_EMC_13_XBAR1_IN25** 0x401F8048U, 0x1U, 0x401F8650U, 0x1U, 0x401F8238U
- #define **IOMUXC_GPIO_EMC_13_LPUART3_TX** 0x401F8048U, 0x2U, 0x401F853CU, 0x1U, 0x401F8238U
- #define **IOMUXC_GPIO_EMC_13_MQS_RIGHT** 0x401F8048U, 0x3U, 0, 0, 0x401F8238U
- #define **IOMUXC_GPIO_EMC_13_FLEXPWM1_PWMB03** 0x401F8048U, 0x4U, 0x401F8464U, 0x1U, 0x401F8238U
- #define **IOMUXC_GPIO_EMC_13_GPIO4_IO13** 0x401F8048U, 0x5U, 0, 0, 0x401F8238U
- #define **IOMUXC_GPIO_EMC_14_SEMC_ADDR05** 0x401F804CU, 0x0U, 0, 0, 0x401F823CU
- #define **IOMUXC_GPIO_EMC_14_XBAR1_INOUT19** 0x401F804CU, 0x1U, 0x401F8654U, 0x0U, 0x401F823CU
- #define **IOMUXC_GPIO_EMC_14_LPUART3_RX** 0x401F804CU, 0x2U, 0x401F8538U, 0x1U, 0x401F823CU
- #define **IOMUXC_GPIO_EMC_14_MQS_LEFT** 0x401F804CU, 0x3U, 0, 0, 0x401F823CU
- #define **IOMUXC_GPIO_EMC_14_LPSPI2_PCS1** 0x401F804CU, 0x4U, 0, 0, 0x401F823CU
- #define **IOMUXC_GPIO_EMC_14_GPIO4_IO14** 0x401F804CU, 0x5U, 0, 0, 0x401F823CU
- #define **IOMUXC_GPIO_EMC_15_SEMC_ADDR06** 0x401F8050U, 0x0U, 0, 0, 0x401F8240U
- #define **IOMUXC_GPIO_EMC_15_XBAR1_IN20** 0x401F8050U, 0x1U, 0x401F8634U, 0x0U, 0x401F8240U
- #define **IOMUXC_GPIO_EMC_15_LPUART3_CTS_B** 0x401F8050U, 0x2U, 0x401F8534U, 0x0U, 0x401F8240U
- #define **IOMUXC_GPIO_EMC_15_SPDIF_OUT** 0x401F8050U, 0x3U, 0, 0, 0x401F8240U
- #define **IOMUXC_GPIO_EMC_15_QTIMER3_TIMER0** 0x401F8050U, 0x4U, 0x401F857CU, 0x0U, 0x401F8240U
- #define **IOMUXC_GPIO_EMC_15_GPIO4_IO15** 0x401F8050U, 0x5U, 0, 0, 0x401F8240U
- #define **IOMUXC_GPIO_EMC_16_SEMC_ADDR07** 0x401F8054U, 0x0U, 0, 0, 0x401F8244U
- #define **IOMUXC_GPIO_EMC_16_XBAR1_IN21** 0x401F8054U, 0x1U, 0x401F8658U, 0x0U, 0x401F8244U
- #define **IOMUXC_GPIO_EMC_16_LPUART3_RTS_B** 0x401F8054U, 0x2U, 0, 0, 0x401F8244U
- #define **IOMUXC_GPIO_EMC_16_SPDIF_IN** 0x401F8054U, 0x3U, 0x401F85C8U, 0x1U,

0x401F8244U
- #define **IOMUXC_GPIO_EMC_16_QTIMER3_TIMER1** 0x401F8054U, 0x4U, 0x401F8580U, 0x1U, 0x401F8244U
- #define **IOMUXC_GPIO_EMC_16_GPIO4_IO16** 0x401F8054U, 0x5U, 0, 0, 0x401F8244U
- #define **IOMUXC_GPIO_EMC_17_SEMC_ADDR08** 0x401F8058U, 0x0U, 0, 0, 0x401F8248U
- #define **IOMUXC_GPIO_EMC_17_FLEXPWM4_PWMA03** 0x401F8058U, 0x1U, 0x401F84A0U, 0x0U, 0x401F8248U
- #define **IOMUXC_GPIO_EMC_17_LPUART4_CTS_B** 0x401F8058U, 0x2U, 0, 0, 0x401F8248U
- #define **IOMUXC_GPIO_EMC_17_FLEXCAN1_TX** 0x401F8058U, 0x3U, 0, 0, 0x401F8248U
- #define **IOMUXC_GPIO_EMC_17_QTIMER3_TIMER2** 0x401F8058U, 0x4U, 0x401F8584U, 0x0U, 0x401F8248U
- #define **IOMUXC_GPIO_EMC_17_GPIO4_IO17** 0x401F8058U, 0x5U, 0, 0, 0x401F8248U
- #define **IOMUXC_GPIO_EMC_18_SEMC_ADDR09** 0x401F805CU, 0x0U, 0, 0, 0x401F824CU
- #define **IOMUXC_GPIO_EMC_18_FLEXPWM4_PWMB03** 0x401F805CU, 0x1U, 0, 0, 0x401F824CU
- #define **IOMUXC_GPIO_EMC_18_LPUART4_RTS_B** 0x401F805CU, 0x2U, 0, 0, 0x401F824CU
- #define **IOMUXC_GPIO_EMC_18_FLEXCAN1_RX** 0x401F805CU, 0x3U, 0x401F844CU, 0x1U, 0x401F824CU
- #define **IOMUXC_GPIO_EMC_18_QTIMER3_TIMER3** 0x401F805CU, 0x4U, 0x401F8588U, 0x0U, 0x401F824CU
- #define **IOMUXC_GPIO_EMC_18_GPIO4_IO18** 0x401F805CU, 0x5U, 0, 0, 0x401F824CU
- #define **IOMUXC_GPIO_EMC_18_SNVS_VIO_5_CTL** 0x401F805CU, 0x6U, 0, 0, 0x401F824CU
- #define **IOMUXC_GPIO_EMC_19_SEMC_ADDR11** 0x401F8060U, 0x0U, 0, 0, 0x401F8250U
- #define **IOMUXC_GPIO_EMC_19_FLEXPWM2_PWMA03** 0x401F8060U, 0x1U, 0x401F8474U, 0x1U, 0x401F8250U
- #define **IOMUXC_GPIO_EMC_19_LPUART4_TX** 0x401F8060U, 0x2U, 0x401F8544U, 0x1U, 0x401F8250U
- #define **IOMUXC_GPIO_EMC_19_ENET_RDATA01** 0x401F8060U, 0x3U, 0x401F8438U, 0x0U, 0x401F8250U
- #define **IOMUXC_GPIO_EMC_19_QTIMER2_TIMER0** 0x401F8060U, 0x4U, 0x401F856CU, 0x0U, 0x401F8250U
- #define **IOMUXC_GPIO_EMC_19_GPIO4_IO19** 0x401F8060U, 0x5U, 0, 0, 0x401F8250U
- #define **IOMUXC_GPIO_EMC_19_SNVS_VIO_5** 0x401F8060U, 0x6U, 0, 0, 0x401F8250U
- #define **IOMUXC_GPIO_EMC_20_SEMC_ADDR12** 0x401F8064U, 0x0U, 0, 0, 0x401F8254U
- #define **IOMUXC_GPIO_EMC_20_FLEXPWM2_PWMB03** 0x401F8064U, 0x1U, 0x401F8484U, 0x1U, 0x401F8254U
- #define **IOMUXC_GPIO_EMC_20_LPUART4_RX** 0x401F8064U, 0x2U, 0x401F8540U, 0x1U, 0x401F8254U
- #define **IOMUXC_GPIO_EMC_20_ENET_RDATA00** 0x401F8064U, 0x3U, 0x401F8434U, 0x0U, 0x401F8254U
- #define **IOMUXC_GPIO_EMC_20_QTIMER2_TIMER1** 0x401F8064U, 0x4U, 0x401F8570U, 0x0U, 0x401F8254U
- #define **IOMUXC_GPIO_EMC_20_GPIO4_IO20** 0x401F8064U, 0x5U, 0, 0, 0x401F8254U
- #define **IOMUXC_GPIO_EMC_21_SEMC_BA0** 0x401F8068U, 0x0U, 0, 0, 0x401F8258U
- #define **IOMUXC_GPIO_EMC_21_FLEXPWM3_PWMA03** 0x401F8068U, 0x1U, 0, 0, 0x401F8258U
- #define **IOMUXC_GPIO_EMC_21_LPI2C3_SDA** 0x401F8068U, 0x2U, 0x401F84E0U, 0x0U, 0x401F8258U

- #define **IOMUXC_GPIO_EMC_21_ENET_TDATA01** 0x401F8068U, 0x3U, 0, 0, 0x401F8258U
- #define **IOMUXC_GPIO_EMC_21_QTIMER2_TIMER2** 0x401F8068U, 0x4U, 0x401F8574U, 0x0U, 0x401F8258U
- #define **IOMUXC_GPIO_EMC_21_GPIO4_IO21** 0x401F8068U, 0x5U, 0, 0, 0x401F8258U
- #define **IOMUXC_GPIO_EMC_22_SEMC_BA1** 0x401F806CU, 0x0U, 0, 0, 0x401F825CU
- #define **IOMUXC_GPIO_EMC_22_FLEXPWM3_PWMB03** 0x401F806CU, 0x1U, 0, 0, 0x401F825CU
- #define **IOMUXC_GPIO_EMC_22_LPI2C3_SCL** 0x401F806CU, 0x2U, 0x401F84DCU, 0x0U, 0x401F825CU
- #define **IOMUXC_GPIO_EMC_22_ENET_TDATA00** 0x401F806CU, 0x3U, 0, 0, 0x401F825CU
- #define **IOMUXC_GPIO_EMC_22_QTIMER2_TIMER3** 0x401F806CU, 0x4U, 0x401F8578U, 0x0U, 0x401F825CU
- #define **IOMUXC_GPIO_EMC_22_GPIO4_IO22** 0x401F806CU, 0x5U, 0, 0, 0x401F825CU
- #define **IOMUXC_GPIO_EMC_23_SEMC_ADDR10** 0x401F8070U, 0x0U, 0, 0, 0x401F8260U
- #define **IOMUXC_GPIO_EMC_23_FLEXPWM1_PWMA00** 0x401F8070U, 0x1U, 0x401F8458U, 0x0U, 0x401F8260U
- #define **IOMUXC_GPIO_EMC_23_LPUART5_TX** 0x401F8070U, 0x2U, 0x401F854CU, 0x0U, 0x401F8260U
- #define **IOMUXC_GPIO_EMC_23_ENET_RX_EN** 0x401F8070U, 0x3U, 0x401F843CU, 0x0U, 0x401F8260U
- #define **IOMUXC_GPIO_EMC_23_GPT1_CAPTURE2** 0x401F8070U, 0x4U, 0, 0, 0x401F8260U
- #define **IOMUXC_GPIO_EMC_23_GPIO4_IO23** 0x401F8070U, 0x5U, 0, 0, 0x401F8260U
- #define **IOMUXC_GPIO_EMC_24_SEMC_CAS** 0x401F8074U, 0x0U, 0, 0, 0x401F8264U
- #define **IOMUXC_GPIO_EMC_24_FLEXPWM1_PWMB00** 0x401F8074U, 0x1U, 0x401F8468U, 0x0U, 0x401F8264U
- #define **IOMUXC_GPIO_EMC_24_LPUART5_RX** 0x401F8074U, 0x2U, 0x401F8548U, 0x0U, 0x401F8264U
- #define **IOMUXC_GPIO_EMC_24_ENET_TX_EN** 0x401F8074U, 0x3U, 0, 0, 0x401F8264U
- #define **IOMUXC_GPIO_EMC_24_GPT1_CAPTURE1** 0x401F8074U, 0x4U, 0, 0, 0x401F8264U
- #define **IOMUXC_GPIO_EMC_24_GPIO4_IO24** 0x401F8074U, 0x5U, 0, 0, 0x401F8264U
- #define **IOMUXC_GPIO_EMC_25_SEMC_RAS** 0x401F8078U, 0x0U, 0, 0, 0x401F8268U
- #define **IOMUXC_GPIO_EMC_25_FLEXPWM1_PWMA01** 0x401F8078U, 0x1U, 0x401F845CU, 0x0U, 0x401F8268U
- #define **IOMUXC_GPIO_EMC_25_LPUART6_TX** 0x401F8078U, 0x2U, 0x401F8554U, 0x0U, 0x401F8268U
- #define **IOMUXC_GPIO_EMC_25_ENET_TX_CLK** 0x401F8078U, 0x3U, 0x401F8448U, 0x0U, 0x401F8268U
- #define **IOMUXC_GPIO_EMC_25_ENET_REF_CLK** 0x401F8078U, 0x4U, 0x401F842CU, 0x0U, 0x401F8268U
- #define **IOMUXC_GPIO_EMC_25_GPIO4_IO25** 0x401F8078U, 0x5U, 0, 0, 0x401F8268U
- #define **IOMUXC_GPIO_EMC_26_SEMC_CLK** 0x401F807CU, 0x0U, 0, 0, 0x401F826CU
- #define **IOMUXC_GPIO_EMC_26_FLEXPWM1_PWMB01** 0x401F807CU, 0x1U, 0x401F846CU, 0x0U, 0x401F826CU
- #define **IOMUXC_GPIO_EMC_26_LPUART6_RX** 0x401F807CU, 0x2U, 0x401F8550U, 0x0U, 0x401F826CU
- #define **IOMUXC_GPIO_EMC_26_ENET_RX_ER** 0x401F807CU, 0x3U, 0x401F8440U, 0x0U, 0x401F826CU

- #define **IOMUXC_GPIO_EMC_26_FLEXIO1_FLEXIO12** 0x401F807CU, 0x4U, 0, 0, 0x401F826CU
- #define **IOMUXC_GPIO_EMC_26_GPIO4_IO26** 0x401F807CU, 0x5U, 0, 0, 0x401F826CU
- #define **IOMUXC_GPIO_EMC_27_SEMC_CKE** 0x401F8080U, 0x0U, 0, 0, 0x401F8270U
- #define **IOMUXC_GPIO_EMC_27_FLEXPWM1_PWMA02** 0x401F8080U, 0x1U, 0x401F8460U, 0x0U, 0x401F8270U
- #define **IOMUXC_GPIO_EMC_27_LPUART5_RTS_B** 0x401F8080U, 0x2U, 0, 0, 0x401F8270U
- #define **IOMUXC_GPIO_EMC_27_LPSPI1_SCK** 0x401F8080U, 0x3U, 0x401F84F0U, 0x0U, 0x401F8270U
- #define **IOMUXC_GPIO_EMC_27_FLEXIO1_FLEXIO13** 0x401F8080U, 0x4U, 0, 0, 0x401F8270U
- #define **IOMUXC_GPIO_EMC_27_GPIO4_IO27** 0x401F8080U, 0x5U, 0, 0, 0x401F8270U
- #define **IOMUXC_GPIO_EMC_28_SEMC_WE** 0x401F8084U, 0x0U, 0, 0, 0x401F8274U
- #define **IOMUXC_GPIO_EMC_28_FLEXPWM1_PWMB02** 0x401F8084U, 0x1U, 0x401F8470U, 0x0U, 0x401F8274U
- #define **IOMUXC_GPIO_EMC_28_LPUART5_CTS_B** 0x401F8084U, 0x2U, 0, 0, 0x401F8274U
- #define **IOMUXC_GPIO_EMC_28_LPSPI1_SDO** 0x401F8084U, 0x3U, 0x401F84F8U, 0x0U, 0x401F8274U
- #define **IOMUXC_GPIO_EMC_28_FLEXIO1_FLEXIO14** 0x401F8084U, 0x4U, 0, 0, 0x401F8274U
- #define **IOMUXC_GPIO_EMC_28_GPIO4_IO28** 0x401F8084U, 0x5U, 0, 0, 0x401F8274U
- #define **IOMUXC_GPIO_EMC_29_SEMC_CS0** 0x401F8088U, 0x0U, 0, 0, 0x401F8278U
- #define **IOMUXC_GPIO_EMC_29_FLEXPWM3_PWMA00** 0x401F8088U, 0x1U, 0, 0, 0x401F8278U
- #define **IOMUXC_GPIO_EMC_29_LPUART6_RTS_B** 0x401F8088U, 0x2U, 0, 0, 0x401F8278U
- #define **IOMUXC_GPIO_EMC_29_LPSPI1_SDI** 0x401F8088U, 0x3U, 0x401F84F4U, 0x0U, 0x401F8278U
- #define **IOMUXC_GPIO_EMC_29_FLEXIO1_FLEXIO15** 0x401F8088U, 0x4U, 0, 0, 0x401F8278U
- #define **IOMUXC_GPIO_EMC_29_GPIO4_IO29** 0x401F8088U, 0x5U, 0, 0, 0x401F8278U
- #define **IOMUXC_GPIO_EMC_30_SEMC_DATA08** 0x401F808CU, 0x0U, 0, 0, 0x401F827CU
- #define **IOMUXC_GPIO_EMC_30_FLEXPWM3_PWMB00** 0x401F808CU, 0x1U, 0, 0, 0x401F827CU
- #define **IOMUXC_GPIO_EMC_30_LPUART6_CTS_B** 0x401F808CU, 0x2U, 0, 0, 0x401F827CU
- #define **IOMUXC_GPIO_EMC_30_LPSPI1_PCS0** 0x401F808CU, 0x3U, 0x401F84ECU, 0x1U, 0x401F827CU
- #define **IOMUXC_GPIO_EMC_30_CSI_DATA23** 0x401F808CU, 0x4U, 0, 0, 0x401F827CU
- #define **IOMUXC_GPIO_EMC_30_GPIO4_IO30** 0x401F808CU, 0x5U, 0, 0, 0x401F827CU
- #define **IOMUXC_GPIO_EMC_31_SEMC_DATA09** 0x401F8090U, 0x0U, 0, 0, 0x401F8280U
- #define **IOMUXC_GPIO_EMC_31_FLEXPWM3_PWMA01** 0x401F8090U, 0x1U, 0, 0, 0x401F8280U
- #define **IOMUXC_GPIO_EMC_31_LPUART7_TX** 0x401F8090U, 0x2U, 0x401F855CU, 0x1U, 0x401F8280U
- #define **IOMUXC_GPIO_EMC_31_LPSPI1_PCS1** 0x401F8090U, 0x3U, 0, 0, 0x401F8280U
- #define **IOMUXC_GPIO_EMC_31_CSI_DATA22** 0x401F8090U, 0x4U, 0, 0, 0x401F8280U
- #define **IOMUXC_GPIO_EMC_31_GPIO4_IO31** 0x401F8090U, 0x5U, 0, 0, 0x401F8280U
- #define **IOMUXC_GPIO_EMC_32_SEMC_DATA10** 0x401F8094U, 0x0U, 0, 0, 0x401F8284U

- #define **IOMUXC_GPIO_EMC_32_FLEXPWM3_PWMB01** 0x401F8094U, 0x1U, 0, 0, 0x401F8284U
- #define **IOMUXC_GPIO_EMC_32_LPUART7_RX** 0x401F8094U, 0x2U, 0x401F8558U, 0x1U, 0x401F8284U
- #define **IOMUXC_GPIO_EMC_32_CCM_PMIC_RDY** 0x401F8094U, 0x3U, 0x401F83FCU, 0x4U, 0x401F8284U
- #define **IOMUXC_GPIO_EMC_32_CSI_DATA21** 0x401F8094U, 0x4U, 0, 0, 0x401F8284U
- #define **IOMUXC_GPIO_EMC_32_GPIO3_IO18** 0x401F8094U, 0x5U, 0, 0, 0x401F8284U
- #define **IOMUXC_GPIO_EMC_33_SEMC_DATA11** 0x401F8098U, 0x0U, 0, 0, 0x401F8288U
- #define **IOMUXC_GPIO_EMC_33_FLEXPWM3_PWMA02** 0x401F8098U, 0x1U, 0, 0, 0x401F8288U
- #define **IOMUXC_GPIO_EMC_33_USDHC1_RESET_B** 0x401F8098U, 0x2U, 0, 0, 0x401F8288U
- #define **IOMUXC_GPIO_EMC_33_SAI3_RX_DATA** 0x401F8098U, 0x3U, 0, 0, 0x401F8288U
- #define **IOMUXC_GPIO_EMC_33_CSI_DATA20** 0x401F8098U, 0x4U, 0, 0, 0x401F8288U
- #define **IOMUXC_GPIO_EMC_33_GPIO3_IO19** 0x401F8098U, 0x5U, 0, 0, 0x401F8288U
- #define **IOMUXC_GPIO_EMC_34_SEMC_DATA12** 0x401F809CU, 0x0U, 0, 0, 0x401F828CU
- #define **IOMUXC_GPIO_EMC_34_FLEXPWM3_PWMB02** 0x401F809CU, 0x1U, 0, 0, 0x401F828CU
- #define **IOMUXC_GPIO_EMC_34_USDHC1_VSELECT** 0x401F809CU, 0x2U, 0, 0, 0x401F828CU
- #define **IOMUXC_GPIO_EMC_34_SAI3_RX_SYNC** 0x401F809CU, 0x3U, 0, 0, 0x401F828CU
- #define **IOMUXC_GPIO_EMC_34_CSI_DATA19** 0x401F809CU, 0x4U, 0, 0, 0x401F828CU
- #define **IOMUXC_GPIO_EMC_34_GPIO3_IO20** 0x401F809CU, 0x5U, 0, 0, 0x401F828CU
- #define **IOMUXC_GPIO_EMC_35_SEMC_DATA13** 0x401F80A0U, 0x0U, 0, 0, 0x401F8290U
- #define **IOMUXC_GPIO_EMC_35_XBAR1_INOUT18** 0x401F80A0U, 0x1U, 0x401F8630U, 0x0U, 0x401F8290U
- #define **IOMUXC_GPIO_EMC_35_GPT1_COMPARE1** 0x401F80A0U, 0x2U, 0, 0, 0x401F8290U
- #define **IOMUXC_GPIO_EMC_35_SAI3_RX_BCLK** 0x401F80A0U, 0x3U, 0, 0, 0x401F8290U
- #define **IOMUXC_GPIO_EMC_35_CSI_DATA18** 0x401F80A0U, 0x4U, 0, 0, 0x401F8290U
- #define **IOMUXC_GPIO_EMC_35_GPIO3_IO21** 0x401F80A0U, 0x5U, 0, 0, 0x401F8290U
- #define **IOMUXC_GPIO_EMC_35_USDHC1_CD_B** 0x401F80A0U, 0x6U, 0x401F85D4U, 0x0U, 0x401F8290U
- #define **IOMUXC_GPIO_EMC_36_SEMC_DATA14** 0x401F80A4U, 0x0U, 0, 0, 0x401F8294U
- #define **IOMUXC_GPIO_EMC_36_XBAR1_IN22** 0x401F80A4U, 0x1U, 0x401F8638U, 0x0U, 0x401F8294U
- #define **IOMUXC_GPIO_EMC_36_GPT1_COMPARE2** 0x401F80A4U, 0x2U, 0, 0, 0x401F8294U
- #define **IOMUXC_GPIO_EMC_36_SAI3_TX_DATA** 0x401F80A4U, 0x3U, 0, 0, 0x401F8294U
- #define **IOMUXC_GPIO_EMC_36_CSI_DATA17** 0x401F80A4U, 0x4U, 0, 0, 0x401F8294U
- #define **IOMUXC_GPIO_EMC_36_GPIO3_IO22** 0x401F80A4U, 0x5U, 0, 0, 0x401F8294U
- #define **IOMUXC_GPIO_EMC_36_USDHC1_WP** 0x401F80A4U, 0x6U, 0x401F85D8U, 0x1U, 0x401F8294U
- #define **IOMUXC_GPIO_EMC_37_SEMC_DATA15** 0x401F80A8U, 0x0U, 0, 0, 0x401F8298U
- #define **IOMUXC_GPIO_EMC_37_XBAR1_IN23** 0x401F80A8U, 0x1U, 0x401F863CU, 0x0U, 0x401F8298U
- #define **IOMUXC_GPIO_EMC_37_GPT1_COMPARE3** 0x401F80A8U, 0x2U, 0, 0, 0x401F8298U
- #define **IOMUXC_GPIO_EMC_37_SAI3_MCLK** 0x401F80A8U, 0x3U, 0, 0, 0x401F8298U

- #define **IOMUXC_GPIO_EMC_37_CSI_DATA16** 0x401F80A8U, 0x4U, 0, 0, 0x401F8298U
- #define **IOMUXC_GPIO_EMC_37_GPIO3_IO23** 0x401F80A8U, 0x5U, 0, 0, 0x401F8298U
- #define **IOMUXC_GPIO_EMC_37_USDHC2_WP** 0x401F80A8U, 0x6U, 0x401F8608U, 0x0U, 0x401F8298U
- #define **IOMUXC_GPIO_EMC_38_SEMC_DM01** 0x401F80ACU, 0x0U, 0, 0, 0x401F829CU
- #define **IOMUXC_GPIO_EMC_38_FLEXPWM1_PWMA03** 0x401F80ACU, 0x1U, 0x401F8454U, 0x2U, 0x401F829CU
- #define **IOMUXC_GPIO_EMC_38_LPUART8_TX** 0x401F80ACU, 0x2U, 0x401F8564U, 0x2U, 0x401F829CU
- #define **IOMUXC_GPIO_EMC_38_SAI3_TX_BCLK** 0x401F80ACU, 0x3U, 0, 0, 0x401F829CU
- #define **IOMUXC_GPIO_EMC_38_CSI_FIELD** 0x401F80ACU, 0x4U, 0, 0, 0x401F829CU
- #define **IOMUXC_GPIO_EMC_38_GPIO3_IO24** 0x401F80ACU, 0x5U, 0, 0, 0x401F829CU
- #define **IOMUXC_GPIO_EMC_38_USDHC2_VSELECT** 0x401F80ACU, 0x6U, 0, 0, 0x401F829CU
- #define **IOMUXC_GPIO_EMC_39_SEMC_DQS** 0x401F80B0U, 0x0U, 0, 0, 0x401F82A0U
- #define **IOMUXC_GPIO_EMC_39_FLEXPWM1_PWMB03** 0x401F80B0U, 0x1U, 0x401F8464U, 0x2U, 0x401F82A0U
- #define **IOMUXC_GPIO_EMC_39_LPUART8_RX** 0x401F80B0U, 0x2U, 0x401F8560U, 0x2U, 0x401F82A0U
- #define **IOMUXC_GPIO_EMC_39_SAI3_TX_SYNC** 0x401F80B0U, 0x3U, 0, 0, 0x401F82A0U
- #define **IOMUXC_GPIO_EMC_39_WDOG1_WDOG_B** 0x401F80B0U, 0x4U, 0, 0, 0x401F82A0U
- #define **IOMUXC_GPIO_EMC_39_GPIO3_IO25** 0x401F80B0U, 0x5U, 0, 0, 0x401F82A0U
- #define **IOMUXC_GPIO_EMC_39_USDHC2_CD_B** 0x401F80B0U, 0x6U, 0x401F85E0U, 0x1U, 0x401F82A0U
- #define **IOMUXC_GPIO_EMC_40_SEMC_RDY** 0x401F80B4U, 0x0U, 0, 0, 0x401F82A4U
- #define **IOMUXC_GPIO_EMC_40_GPT2_CAPTURE2** 0x401F80B4U, 0x1U, 0, 0, 0x401F82A4U
- #define **IOMUXC_GPIO_EMC_40_LPSPI1_PCS2** 0x401F80B4U, 0x2U, 0, 0, 0x401F82A4U
- #define **IOMUXC_GPIO_EMC_40_USB_OTG2_OC** 0x401F80B4U, 0x3U, 0x401F85CCU, 0x1U, 0x401F82A4U
- #define **IOMUXC_GPIO_EMC_40_ENET_MDC** 0x401F80B4U, 0x4U, 0, 0, 0x401F82A4U
- #define **IOMUXC_GPIO_EMC_40_GPIO3_IO26** 0x401F80B4U, 0x5U, 0, 0, 0x401F82A4U
- #define **IOMUXC_GPIO_EMC_40_USDHC2_RESET_B** 0x401F80B4U, 0x6U, 0, 0, 0x401F82A4U
- #define **IOMUXC_GPIO_EMC_41_SEMC_CSX00** 0x401F80B8U, 0x0U, 0, 0, 0x401F82A8U
- #define **IOMUXC_GPIO_EMC_41_GPT2_CAPTURE1** 0x401F80B8U, 0x1U, 0, 0, 0x401F82A8U
- #define **IOMUXC_GPIO_EMC_41_LPSPI1_PCS3** 0x401F80B8U, 0x2U, 0, 0, 0x401F82A8U
- #define **IOMUXC_GPIO_EMC_41_USB_OTG2_PWR** 0x401F80B8U, 0x3U, 0, 0, 0x401F82A8U
- #define **IOMUXC_GPIO_EMC_41_ENET_MDIO** 0x401F80B8U, 0x4U, 0x401F8430U, 0x1U, 0x401F82A8U
- #define **IOMUXC_GPIO_EMC_41_GPIO3_IO27** 0x401F80B8U, 0x5U, 0, 0, 0x401F82A8U
- #define **IOMUXC_GPIO_EMC_41_USDHC1_VSELECT** 0x401F80B8U, 0x6U, 0, 0, 0x401F82A8U
- #define **IOMUXC_GPIO_AD_B0_00_FLEXPWM2_PWMA03** 0x401F80BCU, 0x0U, 0x401F8474U, 0x2U, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_00_XBAR1_INOUT14** 0x401F80BCU, 0x1U, 0x401F8644U,

0x0U, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_00_REF_CLK_32K** 0x401F80BCU, 0x2U, 0, 0, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_00_USB_OTG2_ID** 0x401F80BCU, 0x3U, 0x401F83F8U, 0x0U, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_00_LPI2C1_SCLS** 0x401F80BCU, 0x4U, 0, 0, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_00_GPIO1_IO00** 0x401F80BCU, 0x5U, 0, 0, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_00_USDHC1_RESET_B** 0x401F80BCU, 0x6U, 0, 0, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_00_LPSPI3_SCK** 0x401F80BCU, 0x7U, 0x401F8510U, 0x0U, 0x401F82ACU
- #define **IOMUXC_GPIO_AD_B0_01_FLEXPWM2_PWMB03** 0x401F80C0U, 0x0U, 0x401F8484U, 0x2U, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_01_XBAR1_INOUT15** 0x401F80C0U, 0x1U, 0x401F8648U, 0x0U, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_01_REF_CLK_24M** 0x401F80C0U, 0x2U, 0, 0, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_01_USB_OTG1_ID** 0x401F80C0U, 0x3U, 0x401F83F4U, 0x0U, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_01_LPI2C1_SDAS** 0x401F80C0U, 0x4U, 0, 0, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_01_GPIO1_IO01** 0x401F80C0U, 0x5U, 0, 0, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_01_EWM_OUT_B** 0x401F80C0U, 0x6U, 0, 0, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_01_LPSPI3_SDO** 0x401F80C0U, 0x7U, 0x401F8518U, 0x0U, 0x401F82B0U
- #define **IOMUXC_GPIO_AD_B0_02_FLEXCAN2_TX** 0x401F80C4U, 0x0U, 0, 0, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_02_XBAR1_INOUT16** 0x401F80C4U, 0x1U, 0x401F864CU, 0x0U, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_02_LPUART6_TX** 0x401F80C4U, 0x2U, 0x401F8554U, 0x1U, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_02_USB_OTG1_PWR** 0x401F80C4U, 0x3U, 0, 0, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_02_FLEXPWM1_PWMX00** 0x401F80C4U, 0x4U, 0, 0, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_02_GPIO1_IO02** 0x401F80C4U, 0x5U, 0, 0, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_02_LPI2C1_HREQ** 0x401F80C4U, 0x6U, 0, 0, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_02_LPSPI3_SDI** 0x401F80C4U, 0x7U, 0x401F8514U, 0x0U, 0x401F82B4U
- #define **IOMUXC_GPIO_AD_B0_03_FLEXCAN2_RX** 0x401F80C8U, 0x0U, 0x401F8450U, 0x1U, 0x401F82B8U
- #define **IOMUXC_GPIO_AD_B0_03_XBAR1_INOUT17** 0x401F80C8U, 0x1U, 0x401F862CU, 0x1U, 0x401F82B8U
- #define **IOMUXC_GPIO_AD_B0_03_LPUART6_RX** 0x401F80C8U, 0x2U, 0x401F8550U, 0x1U, 0x401F82B8U
- #define **IOMUXC_GPIO_AD_B0_03_USB_OTG1_OC** 0x401F80C8U, 0x3U, 0x401F85D0U, 0x0U, 0x401F82B8U
- #define **IOMUXC_GPIO_AD_B0_03_FLEXPWM1_PWMX01** 0x401F80C8U, 0x4U, 0, 0, 0x401F82B8U

- #define **IOMUXC_GPIO_AD_B0_03_GPIO1_IO03** 0x401F80C8U, 0x5U, 0, 0, 0x401F82B8U
- #define **IOMUXC_GPIO_AD_B0_03_REF_CLK_24M** 0x401F80C8U, 0x6U, 0, 0, 0x401F82B8U
- #define **IOMUXC_GPIO_AD_B0_03_LPSPI3_PCS0** 0x401F80C8U, 0x7U, 0x401F850CU, 0x0U, 0x401F82B8U
- #define **IOMUXC_GPIO_AD_B0_04_SRC_BOOT_MODE00** 0x401F80CCU, 0x0U, 0, 0, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_04_MQS_RIGHT** 0x401F80CCU, 0x1U, 0, 0, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_04_ENET_TX_DATA03** 0x401F80CCU, 0x2U, 0, 0, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_04_SAI2_TX_SYNC** 0x401F80CCU, 0x3U, 0x401F85C4U, 0x1U, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_04_CSI_DATA09** 0x401F80CCU, 0x4U, 0x401F841CU, 0x1U, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_04_GPIO1_IO04** 0x401F80CCU, 0x5U, 0, 0, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_04_PIT_TRIGGER00** 0x401F80CCU, 0x6U, 0, 0, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_04_LPSPI3_PCS1** 0x401F80CCU, 0x7U, 0, 0, 0x401F82BCU
- #define **IOMUXC_GPIO_AD_B0_05_SRC_BOOT_MODE01** 0x401F80D0U, 0x0U, 0, 0, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_05_MQS_LEFT** 0x401F80D0U, 0x1U, 0, 0, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_05_ENET_TX_DATA02** 0x401F80D0U, 0x2U, 0, 0, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_05_SAI2_TX_BCLK** 0x401F80D0U, 0x3U, 0x401F85C0U, 0x1U, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_05_CSI_DATA08** 0x401F80D0U, 0x4U, 0x401F8418U, 0x1U, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_05_GPIO1_IO05** 0x401F80D0U, 0x5U, 0, 0, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_05_XBAR1_INOUT17** 0x401F80D0U, 0x6U, 0x401F862CU, 0x2U, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_05_LPSPI3_PCS2** 0x401F80D0U, 0x7U, 0, 0, 0x401F82C0U
- #define **IOMUXC_GPIO_AD_B0_06_JTAG_TMS** 0x401F80D4U, 0x0U, 0, 0, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_06_GPT2_COMPARE1** 0x401F80D4U, 0x1U, 0, 0, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_06_ENET_RX_CLK** 0x401F80D4U, 0x2U, 0, 0, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_06_SAI2_RX_BCLK** 0x401F80D4U, 0x3U, 0x401F85B4U, 0x1U, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_06_CSI_DATA07** 0x401F80D4U, 0x4U, 0x401F8414U, 0x1U, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_06_GPIO1_IO06** 0x401F80D4U, 0x5U, 0, 0, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_06_XBAR1_INOUT18** 0x401F80D4U, 0x6U, 0x401F8630U, 0x1U, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_06_LPSPI3_PCS3** 0x401F80D4U, 0x7U, 0, 0, 0x401F82C4U
- #define **IOMUXC_GPIO_AD_B0_07_JTAG_TCK** 0x401F80D8U, 0x0U, 0, 0, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_07_GPT2_COMPARE2** 0x401F80D8U, 0x1U, 0, 0, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_07_ENET_TX_ER** 0x401F80D8U, 0x2U, 0, 0, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_07_SAI2_RX_SYNC** 0x401F80D8U, 0x3U, 0x401F85BCU,

0x1U, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_07_CSI_DATA06** 0x401F80D8U, 0x4U, 0x401F8410U, 0x1U, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_07_GPIO1_IO07** 0x401F80D8U, 0x5U, 0, 0, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_07_XBAR1_INOUT19** 0x401F80D8U, 0x6U, 0x401F8654U, 0x1U, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_07_ENET_1588_EVENT3_OUT** 0x401F80D8U, 0x7U, 0, 0, 0x401F82C8U
- #define **IOMUXC_GPIO_AD_B0_08_JTAG_MOD** 0x401F80DCU, 0x0U, 0, 0, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_08_GPT2_COMPARE3** 0x401F80DCU, 0x1U, 0, 0, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_08_ENET_RX_DATA03** 0x401F80DCU, 0x2U, 0, 0, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_08_SAI2_RX_DATA** 0x401F80DCU, 0x3U, 0x401F85B8U, 0x1U, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_08_CSI_DATA05** 0x401F80DCU, 0x4U, 0x401F840CU, 0x1U, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_08_GPIO1_IO08** 0x401F80DCU, 0x5U, 0, 0, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_08_XBAR1_IN20** 0x401F80DCU, 0x6U, 0x401F8634U, 0x1U, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_08_ENET_1588_EVENT3_IN** 0x401F80DCU, 0x7U, 0, 0, 0x401F82CCU
- #define **IOMUXC_GPIO_AD_B0_09_JTAG_TDI** 0x401F80E0U, 0x0U, 0, 0, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_09_FLEXPWM2_PWMA03** 0x401F80E0U, 0x1U, 0x401F8474U, 0x3U, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_09_ENET_RX_DATA02** 0x401F80E0U, 0x2U, 0, 0, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_09_SAI2_TX_DATA** 0x401F80E0U, 0x3U, 0, 0, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_09_CSI_DATA04** 0x401F80E0U, 0x4U, 0x401F8408U, 0x1U, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_09_GPIO1_IO09** 0x401F80E0U, 0x5U, 0, 0, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_09_XBAR1_IN21** 0x401F80E0U, 0x6U, 0x401F8658U, 0x1U, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_09_GPT2_CLK** 0x401F80E0U, 0x7U, 0, 0, 0x401F82D0U
- #define **IOMUXC_GPIO_AD_B0_10_JTAG_TDO** 0x401F80E4U, 0x0U, 0, 0, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_10_FLEXPWM1_PWMA03** 0x401F80E4U, 0x1U, 0x401F8454U, 0x3U, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_10_ENET_CRS** 0x401F80E4U, 0x2U, 0, 0, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_10_SAI2_MCLK** 0x401F80E4U, 0x3U, 0x401F85B0U, 0x1U, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_10_CSI_DATA03** 0x401F80E4U, 0x4U, 0x401F8404U, 0x1U, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_10_GPIO1_IO10** 0x401F80E4U, 0x5U, 0, 0, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_10_XBAR1_IN22** 0x401F80E4U, 0x6U, 0x401F8638U, 0x1U, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_10_ENET_1588_EVENT0_OUT** 0x401F80E4U, 0x7U, 0, 0, 0x401F82D4U
- #define **IOMUXC_GPIO_AD_B0_11_JTAG_TRSTB** 0x401F80E8U, 0x0U, 0, 0, 0x401F82D8U
- #define **IOMUXC_GPIO_AD_B0_11_FLEXPWM1_PWMB03** 0x401F80E8U, 0x1U, 0x401F8464U, 0x3U, 0x401F82D8U

- #define **IOMUXC_GPIO_AD_B0_11_ENET_COL** 0x401F80E8U, 0x2U, 0, 0, 0x401F82D8U
- #define **IOMUXC_GPIO_AD_B0_11_WDOG1_WDOG_B** 0x401F80E8U, 0x3U, 0, 0, 0x401-F82D8U
- #define **IOMUXC_GPIO_AD_B0_11_CSI_DATA02** 0x401F80E8U, 0x4U, 0x401F8400U, 0x1-U, 0x401F82D8U
- #define **IOMUXC_GPIO_AD_B0_11_GPIO1_IO11** 0x401F80E8U, 0x5U, 0, 0, 0x401F82D8U
- #define **IOMUXC_GPIO_AD_B0_11_XBAR1_IN23** 0x401F80E8U, 0x6U, 0x401F863CU, 0x1-U, 0x401F82D8U
- #define **IOMUXC_GPIO_AD_B0_11_ENET_1588_EVENT0_IN** 0x401F80E8U, 0x7U, 0x401-F8444U, 0x1U, 0x401F82D8U
- #define **IOMUXC_GPIO_AD_B0_12_LPI2C4_SCL** 0x401F80ECU, 0x0U, 0x401F84E4U, 0x1-U, 0x401F82DCU
- #define **IOMUXC_GPIO_AD_B0_12_CCM_PMIC_READY** 0x401F80ECU, 0x1U, 0x401F83-FCU, 0x1U, 0x401F82DCU
- #define **IOMUXC_GPIO_AD_B0_12_LPUART1_TX** 0x401F80ECU, 0x2U, 0, 0, 0x401F82D-CU
- #define **IOMUXC_GPIO_AD_B0_12_WDOG2_WDOG_B** 0x401F80ECU, 0x3U, 0, 0, 0x401-F82DCU
- #define **IOMUXC_GPIO_AD_B0_12_FLEXPWM1_PWMX02** 0x401F80ECU, 0x4U, 0, 0, 0x401F82DCU
- #define **IOMUXC_GPIO_AD_B0_12_GPIO1_IO12** 0x401F80ECU, 0x5U, 0, 0, 0x401F82DCU
- #define **IOMUXC_GPIO_AD_B0_12_ENET_1588_EVENT1_OUT** 0x401F80ECU, 0x6U, 0, 0, 0x401F82DCU
- #define **IOMUXC_GPIO_AD_B0_12_NMI_GLUE_NMI** 0x401F80ECU, 0x7U, 0x401F8568U, 0x0U, 0x401F82DCU
- #define **IOMUXC_GPIO_AD_B0_13_LPI2C4_SDA** 0x401F80F0U, 0x0U, 0x401F84E8U, 0x1-U, 0x401F82E0U
- #define **IOMUXC_GPIO_AD_B0_13_GPT1_CLK** 0x401F80F0U, 0x1U, 0, 0, 0x401F82E0U
- #define **IOMUXC_GPIO_AD_B0_13_LPUART1_RX** 0x401F80F0U, 0x2U, 0, 0, 0x401F82E0U
- #define **IOMUXC_GPIO_AD_B0_13_EWM_OUT_B** 0x401F80F0U, 0x3U, 0, 0, 0x401F82E0U
- #define **IOMUXC_GPIO_AD_B0_13_FLEXPWM1_PWMX03** 0x401F80F0U, 0x4U, 0, 0, 0x401F82E0U
- #define **IOMUXC_GPIO_AD_B0_13_GPIO1_IO13** 0x401F80F0U, 0x5U, 0, 0, 0x401F82E0U
- #define **IOMUXC_GPIO_AD_B0_13_ENET_1588_EVENT1_IN** 0x401F80F0U, 0x6U, 0, 0, 0x401F82E0U
- #define **IOMUXC_GPIO_AD_B0_13_REF_CLK_24M** 0x401F80F0U, 0x7U, 0, 0, 0x401F82-E0U
- #define **IOMUXC_GPIO_AD_B0_14_USB_OTG2_OC** 0x401F80F4U, 0x0U, 0x401F85CCU, 0x0U, 0x401F82E4U
- #define **IOMUXC_GPIO_AD_B0_14_XBAR1_IN24** 0x401F80F4U, 0x1U, 0x401F8640U, 0x1-U, 0x401F82E4U
- #define **IOMUXC_GPIO_AD_B0_14_LPUART1_CTS_B** 0x401F80F4U, 0x2U, 0, 0, 0x401-F82E4U
- #define **IOMUXC_GPIO_AD_B0_14_ENET_1588_EVENT0_OUT** 0x401F80F4U, 0x3U, 0, 0, 0x401F82E4U
- #define **IOMUXC_GPIO_AD_B0_14_CSI_VSYNC** 0x401F80F4U, 0x4U, 0x401F8428U, 0x0U, 0x401F82E4U
- #define **IOMUXC_GPIO_AD_B0_14_GPIO1_IO14** 0x401F80F4U, 0x5U, 0, 0, 0x401F82E4U
- #define **IOMUXC_GPIO_AD_B0_14_FLEXCAN2_TX** 0x401F80F4U, 0x6U, 0, 0, 0x401F82-E4U
- #define **IOMUXC_GPIO_AD_B0_15_USB_OTG2_PWR** 0x401F80F8U, 0x0U, 0, 0, 0x401F82-

E8U
- #define **IOMUXC_GPIO_AD_B0_15_XBAR1_IN25** 0x401F80F8U, 0x1U, 0x401F8650U, 0x0-U, 0x401F82E8U
- #define **IOMUXC_GPIO_AD_B0_15_LPUART1_RTS_B** 0x401F80F8U, 0x2U, 0, 0, 0x401-F82E8U
- #define **IOMUXC_GPIO_AD_B0_15_ENET_1588_EVENT0_IN** 0x401F80F8U, 0x3U, 0x401-F8444U, 0x0U, 0x401F82E8U
- #define **IOMUXC_GPIO_AD_B0_15_CSI_HSYNC** 0x401F80F8U, 0x4U, 0x401F8420U, 0x0U, 0x401F82E8U
- #define **IOMUXC_GPIO_AD_B0_15_GPIO1_IO15** 0x401F80F8U, 0x5U, 0, 0, 0x401F82E8U
- #define **IOMUXC_GPIO_AD_B0_15_FLEXCAN2_RX** 0x401F80F8U, 0x6U, 0x401F8450U, 0x2U, 0x401F82E8U
- #define **IOMUXC_GPIO_AD_B0_15_WDOG1_WDOG_RST_B_DEB** 0x401F80F8U, 0x7U, 0, 0, 0x401F82E8U
- #define **IOMUXC_GPIO_AD_B1_00_USB_OTG2_ID** 0x401F80FCU, 0x0U, 0x401F83F8U, 0x1U, 0x401F82ECU
- #define **IOMUXC_GPIO_AD_B1_00_QTIMER3_TIMER0** 0x401F80FCU, 0x1U, 0x401F857-CU, 0x1U, 0x401F82ECU
- #define **IOMUXC_GPIO_AD_B1_00_LPUART2_CTS_B** 0x401F80FCU, 0x2U, 0, 0, 0x401-F82ECU
- #define **IOMUXC_GPIO_AD_B1_00_LPI2C1_SCL** 0x401F80FCU, 0x3U, 0x401F84CCU, 0x1-U, 0x401F82ECU
- #define **IOMUXC_GPIO_AD_B1_00_WDOG1_B** 0x401F80FCU, 0x4U, 0, 0, 0x401F82ECU
- #define **IOMUXC_GPIO_AD_B1_00_GPIO1_IO16** 0x401F80FCU, 0x5U, 0, 0, 0x401F82ECU
- #define **IOMUXC_GPIO_AD_B1_00_USDHC1_WP** 0x401F80FCU, 0x6U, 0x401F85D8U, 0x2U, 0x401F82ECU
- #define **IOMUXC_GPIO_AD_B1_00_KPP_ROW07** 0x401F80FCU, 0x7U, 0, 0, 0x401F82ECU
- #define **IOMUXC_GPIO_AD_B1_01_USB_OTG1_PWR** 0x401F8100U, 0x0U, 0, 0, 0x401F82-F0U
- #define **IOMUXC_GPIO_AD_B1_01_QTIMER3_TIMER1** 0x401F8100U, 0x1U, 0x401F8580-U, 0x0U, 0x401F82F0U
- #define **IOMUXC_GPIO_AD_B1_01_LPUART2_RTS_B** 0x401F8100U, 0x2U, 0, 0, 0x401F82-F0U
- #define **IOMUXC_GPIO_AD_B1_01_LPI2C1_SDA** 0x401F8100U, 0x3U, 0x401F84D0U, 0x1-U, 0x401F82F0U
- #define **IOMUXC_GPIO_AD_B1_01_CCM_PMIC_READY** 0x401F8100U, 0x4U, 0x401F83-FCU, 0x2U, 0x401F82F0U
- #define **IOMUXC_GPIO_AD_B1_01_GPIO1_IO17** 0x401F8100U, 0x5U, 0, 0, 0x401F82F0U
- #define **IOMUXC_GPIO_AD_B1_01_USDHC1_VSELECT** 0x401F8100U, 0x6U, 0, 0, 0x401-F82F0U
- #define **IOMUXC_GPIO_AD_B1_01_KPP_COL07** 0x401F8100U, 0x7U, 0, 0, 0x401F82F0U
- #define **IOMUXC_GPIO_AD_B1_02_USB_OTG1_ID** 0x401F8104U, 0x0U, 0x401F83F4U, 0x1U, 0x401F82F4U
- #define **IOMUXC_GPIO_AD_B1_02_QTIMER3_TIMER2** 0x401F8104U, 0x1U, 0x401F8584-U, 0x1U, 0x401F82F4U
- #define **IOMUXC_GPIO_AD_B1_02_LPUART2_TX** 0x401F8104U, 0x2U, 0x401F8530U, 0x1-U, 0x401F82F4U
- #define **IOMUXC_GPIO_AD_B1_02_SPDIF_OUT** 0x401F8104U, 0x3U, 0, 0, 0x401F82F4U
- #define **IOMUXC_GPIO_AD_B1_02_ENET_1588_EVENT2_OUT** 0x401F8104U, 0x4U, 0, 0, 0x401F82F4U
- #define **IOMUXC_GPIO_AD_B1_02_GPIO1_IO18** 0x401F8104U, 0x5U, 0, 0, 0x401F82F4U

- #define **IOMUXC_GPIO_AD_B1_02_USDHC1_CD_B** 0x401F8104U, 0x6U, 0x401F85D4U, 0x1U, 0x401F82F4U
- #define **IOMUXC_GPIO_AD_B1_02_KPP_ROW06** 0x401F8104U, 0x7U, 0, 0, 0x401F82F4U
- #define **IOMUXC_GPIO_AD_B1_03_USB_OTG1_OC** 0x401F8108U, 0x0U, 0x401F85D0U, 0x1U, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_03_QTIMER3_TIMER3** 0x401F8108U, 0x1U, 0x401F8588U, 0x1U, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_03_LPUART2_RX** 0x401F8108U, 0x2U, 0x401F852CU, 0x1U, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_03_SPDIF_IN** 0x401F8108U, 0x3U, 0x401F85C8U, 0x0U, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_03_ENET_1588_EVENT2_IN** 0x401F8108U, 0x4U, 0, 0, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_03_GPIO1_IO19** 0x401F8108U, 0x5U, 0, 0, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_03_USDHC2_CD_B** 0x401F8108U, 0x6U, 0x401F85E0U, 0x0U, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_03_KPP_COL06** 0x401F8108U, 0x7U, 0, 0, 0x401F82F8U
- #define **IOMUXC_GPIO_AD_B1_04_FLEXSPIB_DATA03** 0x401F810CU, 0x0U, 0x401F84C4U, 0x1U, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_04_ENET_MDC** 0x401F810CU, 0x1U, 0, 0, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_04_LPUART3_CTS_B** 0x401F810CU, 0x2U, 0x401F8534U, 0x1U, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_04_SPDIF_SR_CLK** 0x401F810CU, 0x3U, 0, 0, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_04_CSI_PIXCLK** 0x401F810CU, 0x4U, 0x401F8424U, 0x0U, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_04_GPIO1_IO20** 0x401F810CU, 0x5U, 0, 0, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_04_USDHC2_DATA0** 0x401F810CU, 0x6U, 0x401F85E8U, 0x1U, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_04_KPP_ROW05** 0x401F810CU, 0x7U, 0, 0, 0x401F82FCU
- #define **IOMUXC_GPIO_AD_B1_05_FLEXSPIB_DATA02** 0x401F8110U, 0x0U, 0x401F84C0U, 0x1U, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_05_ENET_MDIO** 0x401F8110U, 0x1U, 0x401F8430U, 0x0U, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_05_LPUART3_RTS_B** 0x401F8110U, 0x2U, 0, 0, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_05_SPDIF_OUT** 0x401F8110U, 0x3U, 0, 0, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_05_CSI_MCLK** 0x401F8110U, 0x4U, 0, 0, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_05_GPIO1_IO21** 0x401F8110U, 0x5U, 0, 0, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_05_USDHC2_DATA1** 0x401F8110U, 0x6U, 0x401F85ECU, 0x1U, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_05_KPP_COL05** 0x401F8110U, 0x7U, 0, 0, 0x401F8300U
- #define **IOMUXC_GPIO_AD_B1_06_FLEXSPIB_DATA01** 0x401F8114U, 0x0U, 0x401F84BCU, 0x1U, 0x401F8304U
- #define **IOMUXC_GPIO_AD_B1_06_LPI2C3_SDA** 0x401F8114U, 0x1U, 0x401F84E0U, 0x2U, 0x401F8304U
- #define **IOMUXC_GPIO_AD_B1_06_LPUART3_TX** 0x401F8114U, 0x2U, 0x401F853CU, 0x0U, 0x401F8304U
- #define **IOMUXC_GPIO_AD_B1_06_SPDIF_LOCK** 0x401F8114U, 0x3U, 0, 0, 0x401F8304U
- #define **IOMUXC_GPIO_AD_B1_06_CSI_VSYNC** 0x401F8114U, 0x4U, 0x401F8428U, 0x1U, 0x401F8304U

- #define **IOMUXC_GPIO_AD_B1_06_GPIO1_IO22** 0x401F8114U, 0x5U, 0, 0, 0x401F8304U
- #define **IOMUXC_GPIO_AD_B1_06_USDHC2_DATA2** 0x401F8114U, 0x6U, 0x401F85F0U, 0x1U, 0x401F8304U
- #define **IOMUXC_GPIO_AD_B1_06_KPP_ROW04** 0x401F8114U, 0x7U, 0, 0, 0x401F8304U
- #define **IOMUXC_GPIO_AD_B1_07_FLEXSPIB_DATA00** 0x401F8118U, 0x0U, 0x401F84B8U, 0x1U, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_07_LPI2C3_SCL** 0x401F8118U, 0x1U, 0x401F84DCU, 0x2U, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_07_LPUART3_RX** 0x401F8118U, 0x2U, 0x401F8538U, 0x0U, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_07_SPDIF_EXT_CLK** 0x401F8118U, 0x3U, 0, 0, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_07_CSI_HSYNC** 0x401F8118U, 0x4U, 0x401F8420U, 0x1U, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_07_GPIO1_IO23** 0x401F8118U, 0x5U, 0, 0, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_07_USDHC2_DATA3** 0x401F8118U, 0x6U, 0x401F85F4U, 0x1U, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_07_KPP_COL04** 0x401F8118U, 0x7U, 0, 0, 0x401F8308U
- #define **IOMUXC_GPIO_AD_B1_08_FLEXSPIA_SS1_B** 0x401F811CU, 0x0U, 0, 0, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_08_FLEXPWM4_PWMA00** 0x401F811CU, 0x1U, 0x401F8494U, 0x1U, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_08_FLEXCAN1_TX** 0x401F811CU, 0x2U, 0, 0, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_08_CCM_PMIC_READY** 0x401F811CU, 0x3U, 0x401F83FCU, 0x3U, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_08_CSI_DATA09** 0x401F811CU, 0x4U, 0x401F841CU, 0x0U, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_08_GPIO1_IO24** 0x401F811CU, 0x5U, 0, 0, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_08_USDHC2_CMD** 0x401F811CU, 0x6U, 0x401F85E4U, 0x1U, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_08_KPP_ROW03** 0x401F811CU, 0x7U, 0, 0, 0x401F830CU
- #define **IOMUXC_GPIO_AD_B1_09_FLEXSPIA_DQS** 0x401F8120U, 0x0U, 0x401F84A4U, 0x1U, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_09_FLEXPWM4_PWMA01** 0x401F8120U, 0x1U, 0x401F8498U, 0x1U, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_09_FLEXCAN1_RX** 0x401F8120U, 0x2U, 0x401F844CU, 0x2U, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_09_SAI1_MCLK** 0x401F8120U, 0x3U, 0x401F858CU, 0x1U, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_09_CSI_DATA08** 0x401F8120U, 0x4U, 0x401F8418U, 0x0U, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_09_GPIO1_IO25** 0x401F8120U, 0x5U, 0, 0, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_09_USDHC2_CLK** 0x401F8120U, 0x6U, 0x401F85DCU, 0x1U, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_09_KPP_COL03** 0x401F8120U, 0x7U, 0, 0, 0x401F8310U
- #define **IOMUXC_GPIO_AD_B1_10_FLEXSPIA_DATA03** 0x401F8124U, 0x0U, 0x401F84B4U, 0x1U, 0x401F8314U
- #define **IOMUXC_GPIO_AD_B1_10_WDOG1_B** 0x401F8124U, 0x1U, 0, 0, 0x401F8314U
- #define **IOMUXC_GPIO_AD_B1_10_LPUART8_TX** 0x401F8124U, 0x2U, 0x401F8564U, 0x1U, 0x401F8314U

- #define **IOMUXC_GPIO_AD_B1_10_SAI1_RX_SYNC** 0x401F8124U, 0x3U, 0x401F85A4U, 0x1U, 0x401F8314U
- #define **IOMUXC_GPIO_AD_B1_10_CSI_DATA07** 0x401F8124U, 0x4U, 0x401F8414U, 0x0U, 0x401F8314U
- #define **IOMUXC_GPIO_AD_B1_10_GPIO1_IO26** 0x401F8124U, 0x5U, 0, 0, 0x401F8314U
- #define **IOMUXC_GPIO_AD_B1_10_USDHC2_WP** 0x401F8124U, 0x6U, 0x401F8608U, 0x1U, 0x401F8314U
- #define **IOMUXC_GPIO_AD_B1_10_KPP_ROW02** 0x401F8124U, 0x7U, 0, 0, 0x401F8314U
- #define **IOMUXC_GPIO_AD_B1_11_FLEXSPIA_DATA02** 0x401F8128U, 0x0U, 0x401F84B0U, 0x1U, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_11_EWM_OUT_B** 0x401F8128U, 0x1U, 0, 0, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_11_LPUART8_RX** 0x401F8128U, 0x2U, 0x401F8560U, 0x1U, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_11_SAI1_RX_BCLK** 0x401F8128U, 0x3U, 0x401F8590U, 0x1U, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_11_CSI_DATA06** 0x401F8128U, 0x4U, 0x401F8410U, 0x0U, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_11_GPIO1_IO27** 0x401F8128U, 0x5U, 0, 0, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_11_USDHC2_RESET_B** 0x401F8128U, 0x6U, 0, 0, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_11_KPP_COL02** 0x401F8128U, 0x7U, 0, 0, 0x401F8318U
- #define **IOMUXC_GPIO_AD_B1_12_FLEXSPIA_DATA01** 0x401F812CU, 0x0U, 0x401F84ACU, 0x1U, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_12_ACMP_OUT00** 0x401F812CU, 0x1U, 0, 0, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_12_LPSPI3_PCS0** 0x401F812CU, 0x2U, 0x401F850CU, 0x1U, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_12_SAI1_RX_DATA00** 0x401F812CU, 0x3U, 0x401F8594U, 0x1U, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_12_CSI_DATA05** 0x401F812CU, 0x4U, 0x401F840CU, 0x0U, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_12_GPIO1_IO28** 0x401F812CU, 0x5U, 0, 0, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_12_USDHC2_DATA4** 0x401F812CU, 0x6U, 0x401F85F8U, 0x1U, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_12_KPP_ROW01** 0x401F812CU, 0x7U, 0, 0, 0x401F831CU
- #define **IOMUXC_GPIO_AD_B1_13_FLEXSPIA_DATA00** 0x401F8130U, 0x0U, 0x401F84A8U, 0x1U, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_13_ACMP_OUT01** 0x401F8130U, 0x1U, 0, 0, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_13_LPSPI3_SDI** 0x401F8130U, 0x2U, 0x401F8514U, 0x1U, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_13_SAI1_TX_DATA00** 0x401F8130U, 0x3U, 0, 0, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_13_CSI_DATA04** 0x401F8130U, 0x4U, 0x401F8408U, 0x0U, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_13_GPIO1_IO29** 0x401F8130U, 0x5U, 0, 0, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_13_USDHC2_DATA5** 0x401F8130U, 0x6U, 0x401F85FCU, 0x1U, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_13_KPP_COL01** 0x401F8130U, 0x7U, 0, 0, 0x401F8320U
- #define **IOMUXC_GPIO_AD_B1_14_FLEXSPIA_SCLK** 0x401F8134U, 0x0U, 0x401F84C8U, 0x1U, 0x401F8324U
- #define **IOMUXC_GPIO_AD_B1_14_ACMP_OUT02** 0x401F8134U, 0x1U, 0, 0, 0x401F8324U

- #define **IOMUXC_GPIO_AD_B1_14_LPSPI3_SDO** 0x401F8134U, 0x2U, 0x401F8518U, 0x1U, 0x401F8324U
- #define **IOMUXC_GPIO_AD_B1_14_SAI1_TX_BCLK** 0x401F8134U, 0x3U, 0x401F85A8U, 0x1U, 0x401F8324U
- #define **IOMUXC_GPIO_AD_B1_14_CSI_DATA03** 0x401F8134U, 0x4U, 0x401F8404U, 0x0U, 0x401F8324U
- #define **IOMUXC_GPIO_AD_B1_14_GPIO1_IO30** 0x401F8134U, 0x5U, 0, 0, 0x401F8324U
- #define **IOMUXC_GPIO_AD_B1_14_USDHC2_DATA6** 0x401F8134U, 0x6U, 0x401F8600U, 0x1U, 0x401F8324U
- #define **IOMUXC_GPIO_AD_B1_14_KPP_ROW00** 0x401F8134U, 0x7U, 0, 0, 0x401F8324U
- #define **IOMUXC_GPIO_AD_B1_15_FLEXSPIA_SS0_B** 0x401F8138U, 0x0U, 0, 0, 0x401F8328U
- #define **IOMUXC_GPIO_AD_B1_15_ACMP_OUT03** 0x401F8138U, 0x1U, 0, 0, 0x401F8328U
- #define **IOMUXC_GPIO_AD_B1_15_LPSPI3_SCK** 0x401F8138U, 0x2U, 0, 0, 0x401F8328U
- #define **IOMUXC_GPIO_AD_B1_15_SAI1_TX_SYNC** 0x401F8138U, 0x3U, 0x401F85ACU, 0x1U, 0x401F8328U
- #define **IOMUXC_GPIO_AD_B1_15_CSI_DATA02** 0x401F8138U, 0x4U, 0x401F8400U, 0x0U, 0x401F8328U
- #define **IOMUXC_GPIO_AD_B1_15_GPIO1_IO31** 0x401F8138U, 0x5U, 0, 0, 0x401F8328U
- #define **IOMUXC_GPIO_AD_B1_15_USDHC2_DATA7** 0x401F8138U, 0x6U, 0x401F8604U, 0x1U, 0x401F8328U
- #define **IOMUXC_GPIO_AD_B1_15_KPP_COL00** 0x401F8138U, 0x7U, 0, 0, 0x401F8328U
- #define **IOMUXC_GPIO_B0_00_LCD_CLK** 0x401F813CU, 0x0U, 0, 0, 0x401F832CU
- #define **IOMUXC_GPIO_B0_00_QTIMER1_TIMER0** 0x401F813CU, 0x1U, 0, 0, 0x401F832CU
- #define **IOMUXC_GPIO_B0_00_MQS_RIGHT** 0x401F813CU, 0x2U, 0, 0, 0x401F832CU
- #define **IOMUXC_GPIO_B0_00_LPSPI4_PCS0** 0x401F813CU, 0x3U, 0x401F851CU, 0x0U, 0x401F832CU
- #define **IOMUXC_GPIO_B0_00_FLEXIO2_FLEXIO00** 0x401F813CU, 0x4U, 0, 0, 0x401F832CU
- #define **IOMUXC_GPIO_B0_00_GPIO2_IO00** 0x401F813CU, 0x5U, 0, 0, 0x401F832CU
- #define **IOMUXC_GPIO_B0_00_SEMC_CSX01** 0x401F813CU, 0x6U, 0, 0, 0x401F832CU
- #define **IOMUXC_GPIO_B0_01_LCD_ENABLE** 0x401F8140U, 0x0U, 0, 0, 0x401F8330U
- #define **IOMUXC_GPIO_B0_01_QTIMER1_TIMER1** 0x401F8140U, 0x1U, 0, 0, 0x401F8330U
- #define **IOMUXC_GPIO_B0_01_MQS_LEFT** 0x401F8140U, 0x2U, 0, 0, 0x401F8330U
- #define **IOMUXC_GPIO_B0_01_LPSPI4_SDI** 0x401F8140U, 0x3U, 0x401F8524U, 0x0U, 0x401F8330U
- #define **IOMUXC_GPIO_B0_01_FLEXIO2_FLEXIO01** 0x401F8140U, 0x4U, 0, 0, 0x401F8330U
- #define **IOMUXC_GPIO_B0_01_GPIO2_IO01** 0x401F8140U, 0x5U, 0, 0, 0x401F8330U
- #define **IOMUXC_GPIO_B0_01_SEMC_CSX02** 0x401F8140U, 0x6U, 0, 0, 0x401F8330U
- #define **IOMUXC_GPIO_B0_02_LCD_HSYNC** 0x401F8144U, 0x0U, 0, 0, 0x401F8334U
- #define **IOMUXC_GPIO_B0_02_QTIMER1_TIMER2** 0x401F8144U, 0x1U, 0, 0, 0x401F8334U
- #define **IOMUXC_GPIO_B0_02_FLEXCAN1_TX** 0x401F8144U, 0x2U, 0, 0, 0x401F8334U
- #define **IOMUXC_GPIO_B0_02_LPSPI4_SDO** 0x401F8144U, 0x3U, 0x401F8528U, 0x0U, 0x401F8334U
- #define **IOMUXC_GPIO_B0_02_FLEXIO2_FLEXIO02** 0x401F8144U, 0x4U, 0, 0, 0x401F8334U
- #define **IOMUXC_GPIO_B0_02_GPIO2_IO02** 0x401F8144U, 0x5U, 0, 0, 0x401F8334U
- #define **IOMUXC_GPIO_B0_02_SEMC_CSX03** 0x401F8144U, 0x6U, 0, 0, 0x401F8334U

- #define **IOMUXC_GPIO_B0_03_LCD_VSYNC** 0x401F8148U, 0x0U, 0, 0, 0x401F8338U
- #define **IOMUXC_GPIO_B0_03_QTIMER2_TIMER0** 0x401F8148U, 0x1U, 0x401F856CU, 0x1U, 0x401F8338U
- #define **IOMUXC_GPIO_B0_03_FLEXCAN1_RX** 0x401F8148U, 0x2U, 0x401F844CU, 0x3U, 0x401F8338U
- #define **IOMUXC_GPIO_B0_03_LPSPI4_SCK** 0x401F8148U, 0x3U, 0x401F8520U, 0x0U, 0x401F8338U
- #define **IOMUXC_GPIO_B0_03_FLEXIO2_FLEXIO03** 0x401F8148U, 0x4U, 0, 0, 0x401F8338U
- #define **IOMUXC_GPIO_B0_03_GPIO2_IO03** 0x401F8148U, 0x5U, 0, 0, 0x401F8338U
- #define **IOMUXC_GPIO_B0_03_WDOG2_RESET_B_DEB** 0x401F8148U, 0x6U, 0, 0, 0x401F8338U
- #define **IOMUXC_GPIO_B0_04_LCD_DATA00** 0x401F814CU, 0x0U, 0, 0, 0x401F833CU
- #define **IOMUXC_GPIO_B0_04_QTIMER2_TIMER1** 0x401F814CU, 0x1U, 0x401F8570U, 0x1U, 0x401F833CU
- #define **IOMUXC_GPIO_B0_04_LPI2C2_SCL** 0x401F814CU, 0x2U, 0x401F84D4U, 0x1U, 0x401F833CU
- #define **IOMUXC_GPIO_B0_04_ARM_CM7_TRACE00** 0x401F814CU, 0x3U, 0, 0, 0x401F833CU
- #define **IOMUXC_GPIO_B0_04_FLEXIO2_FLEXIO04** 0x401F814CU, 0x4U, 0, 0, 0x401F833CU
- #define **IOMUXC_GPIO_B0_04_GPIO2_IO04** 0x401F814CU, 0x5U, 0, 0, 0x401F833CU
- #define **IOMUXC_GPIO_B0_04_SRC_BOOT_CFG00** 0x401F814CU, 0x6U, 0, 0, 0x401F833CU
- #define **IOMUXC_GPIO_B0_05_LCD_DATA01** 0x401F8150U, 0x0U, 0, 0, 0x401F8340U
- #define **IOMUXC_GPIO_B0_05_QTIMER2_TIMER2** 0x401F8150U, 0x1U, 0x401F8574U, 0x1U, 0x401F8340U
- #define **IOMUXC_GPIO_B0_05_LPI2C2_SDA** 0x401F8150U, 0x2U, 0x401F84D8U, 0x1U, 0x401F8340U
- #define **IOMUXC_GPIO_B0_05_ARM_CM7_TRACE01** 0x401F8150U, 0x3U, 0, 0, 0x401F8340U
- #define **IOMUXC_GPIO_B0_05_FLEXIO2_FLEXIO05** 0x401F8150U, 0x4U, 0, 0, 0x401F8340U
- #define **IOMUXC_GPIO_B0_05_GPIO2_IO05** 0x401F8150U, 0x5U, 0, 0, 0x401F8340U
- #define **IOMUXC_GPIO_B0_05_SRC_BOOT_CFG01** 0x401F8150U, 0x6U, 0, 0, 0x401F8340U
- #define **IOMUXC_GPIO_B0_06_LCD_DATA02** 0x401F8154U, 0x0U, 0, 0, 0x401F8344U
- #define **IOMUXC_GPIO_B0_06_QTIMER3_TIMER0** 0x401F8154U, 0x1U, 0x401F857CU, 0x2U, 0x401F8344U
- #define **IOMUXC_GPIO_B0_06_FLEXPWM2_PWMA00** 0x401F8154U, 0x2U, 0x401F8478U, 0x1U, 0x401F8344U
- #define **IOMUXC_GPIO_B0_06_ARM_CM7_TRACE02** 0x401F8154U, 0x3U, 0, 0, 0x401F8344U
- #define **IOMUXC_GPIO_B0_06_FLEXIO2_FLEXIO06** 0x401F8154U, 0x4U, 0, 0, 0x401F8344U
- #define **IOMUXC_GPIO_B0_06_GPIO2_IO06** 0x401F8154U, 0x5U, 0, 0, 0x401F8344U
- #define **IOMUXC_GPIO_B0_06_SRC_BOOT_CFG02** 0x401F8154U, 0x6U, 0, 0, 0x401F8344U
- #define **IOMUXC_GPIO_B0_07_LCD_DATA03** 0x401F8158U, 0x0U, 0, 0, 0x401F8348U
- #define **IOMUXC_GPIO_B0_07_QTIMER3_TIMER1** 0x401F8158U, 0x1U, 0x401F8580U, 0x2U, 0x401F8348U

- #define **IOMUXC_GPIO_B0_07_FLEXPWM2_PWMB00** 0x401F8158U, 0x2U, 0x401F8488U, 0x1U, 0x401F8348U
- #define **IOMUXC_GPIO_B0_07_ARM_CM7_TRACE03** 0x401F8158U, 0x3U, 0, 0, 0x401F8348U
- #define **IOMUXC_GPIO_B0_07_FLEXIO2_FLEXIO07** 0x401F8158U, 0x4U, 0, 0, 0x401F8348U
- #define **IOMUXC_GPIO_B0_07_GPIO2_IO07** 0x401F8158U, 0x5U, 0, 0, 0x401F8348U
- #define **IOMUXC_GPIO_B0_07_SRC_BOOT_CFG03** 0x401F8158U, 0x6U, 0, 0, 0x401F8348U
- #define **IOMUXC_GPIO_B0_08_LCD_DATA04** 0x401F815CU, 0x0U, 0, 0, 0x401F834CU
- #define **IOMUXC_GPIO_B0_08_QTIMER3_TIMER2** 0x401F815CU, 0x1U, 0x401F8584U, 0x2U, 0x401F834CU
- #define **IOMUXC_GPIO_B0_08_FLEXPWM2_PWMA01** 0x401F815CU, 0x2U, 0x401F847CU, 0x1U, 0x401F834CU
- #define **IOMUXC_GPIO_B0_08_LPUART3_TX** 0x401F815CU, 0x3U, 0x401F853CU, 0x2U, 0x401F834CU
- #define **IOMUXC_GPIO_B0_08_FLEXIO2_FLEXIO08** 0x401F815CU, 0x4U, 0, 0, 0x401F834CU
- #define **IOMUXC_GPIO_B0_08_GPIO2_IO08** 0x401F815CU, 0x5U, 0, 0, 0x401F834CU
- #define **IOMUXC_GPIO_B0_08_SRC_BOOT_CFG04** 0x401F815CU, 0x6U, 0, 0, 0x401F834CU
- #define **IOMUXC_GPIO_B0_09_LCD_DATA05** 0x401F8160U, 0x0U, 0, 0, 0x401F8350U
- #define **IOMUXC_GPIO_B0_09_QTIMER4_TIMER0** 0x401F8160U, 0x1U, 0, 0, 0x401F8350U
- #define **IOMUXC_GPIO_B0_09_FLEXPWM2_PWMB01** 0x401F8160U, 0x2U, 0x401F848CU, 0x1U, 0x401F8350U
- #define **IOMUXC_GPIO_B0_09_LPUART3_RX** 0x401F8160U, 0x3U, 0x401F8538U, 0x2U, 0x401F8350U
- #define **IOMUXC_GPIO_B0_09_FLEXIO2_FLEXIO09** 0x401F8160U, 0x4U, 0, 0, 0x401F8350U
- #define **IOMUXC_GPIO_B0_09_GPIO2_IO09** 0x401F8160U, 0x5U, 0, 0, 0x401F8350U
- #define **IOMUXC_GPIO_B0_09_SRC_BOOT_CFG05** 0x401F8160U, 0x6U, 0, 0, 0x401F8350U
- #define **IOMUXC_GPIO_B0_10_LCD_DATA06** 0x401F8164U, 0x0U, 0, 0, 0x401F8354U
- #define **IOMUXC_GPIO_B0_10_QTIMER4_TIMER1** 0x401F8164U, 0x1U, 0, 0, 0x401F8354U
- #define **IOMUXC_GPIO_B0_10_FLEXPWM2_PWMA02** 0x401F8164U, 0x2U, 0x401F8480U, 0x1U, 0x401F8354U
- #define **IOMUXC_GPIO_B0_10_SAI1_TX_DATA03** 0x401F8164U, 0x3U, 0x401F8598U, 0x1U, 0x401F8354U
- #define **IOMUXC_GPIO_B0_10_FLEXIO2_FLEXIO10** 0x401F8164U, 0x4U, 0, 0, 0x401F8354U
- #define **IOMUXC_GPIO_B0_10_GPIO2_IO10** 0x401F8164U, 0x5U, 0, 0, 0x401F8354U
- #define **IOMUXC_GPIO_B0_10_SRC_BOOT_CFG06** 0x401F8164U, 0x6U, 0, 0, 0x401F8354U
- #define **IOMUXC_GPIO_B0_11_LCD_DATA07** 0x401F8168U, 0x0U, 0, 0, 0x401F8358U
- #define **IOMUXC_GPIO_B0_11_QTIMER4_TIMER2** 0x401F8168U, 0x1U, 0, 0, 0x401F8358U
- #define **IOMUXC_GPIO_B0_11_FLEXPWM2_PWMB02** 0x401F8168U, 0x2U, 0x401F8490U, 0x1U, 0x401F8358U
- #define **IOMUXC_GPIO_B0_11_SAI1_TX_DATA02** 0x401F8168U, 0x3U, 0x401F859CU,

0x1U, 0x401F8358U
- #define **IOMUXC_GPIO_B0_11_FLEXIO2_FLEXIO11** 0x401F8168U, 0x4U, 0, 0, 0x401F8358U
- #define **IOMUXC_GPIO_B0_11_GPIO2_IO11** 0x401F8168U, 0x5U, 0, 0, 0x401F8358U
- #define **IOMUXC_GPIO_B0_11_SRC_BOOT_CFG07** 0x401F8168U, 0x6U, 0, 0, 0x401F8358U
- #define **IOMUXC_GPIO_B0_12_LCD_DATA08** 0x401F816CU, 0x0U, 0, 0, 0x401F835CU
- #define **IOMUXC_GPIO_B0_12_XBAR1_INOUT10** 0x401F816CU, 0x1U, 0, 0, 0x401F835CU
- #define **IOMUXC_GPIO_B0_12_ARM_CM7_TRACE_CLK** 0x401F816CU, 0x2U, 0, 0, 0x401F835CU
- #define **IOMUXC_GPIO_B0_12_SAI1_TX_DATA01** 0x401F816CU, 0x3U, 0x401F85A0U, 0x1U, 0x401F835CU
- #define **IOMUXC_GPIO_B0_12_FLEXIO2_FLEXIO12** 0x401F816CU, 0x4U, 0, 0, 0x401F835CU
- #define **IOMUXC_GPIO_B0_12_GPIO2_IO12** 0x401F816CU, 0x5U, 0, 0, 0x401F835CU
- #define **IOMUXC_GPIO_B0_12_SRC_BOOT_CFG08** 0x401F816CU, 0x6U, 0, 0, 0x401F835CU
- #define **IOMUXC_GPIO_B0_13_LCD_DATA09** 0x401F8170U, 0x0U, 0, 0, 0x401F8360U
- #define **IOMUXC_GPIO_B0_13_XBAR1_INOUT11** 0x401F8170U, 0x1U, 0, 0, 0x401F8360U
- #define **IOMUXC_GPIO_B0_13_ARM_CM7_TRACE_SWO** 0x401F8170U, 0x2U, 0, 0, 0x401F8360U
- #define **IOMUXC_GPIO_B0_13_SAI1_MCLK** 0x401F8170U, 0x3U, 0x401F858CU, 0x2U, 0x401F8360U
- #define **IOMUXC_GPIO_B0_13_FLEXIO2_FLEXIO13** 0x401F8170U, 0x4U, 0, 0, 0x401F8360U
- #define **IOMUXC_GPIO_B0_13_GPIO2_IO13** 0x401F8170U, 0x5U, 0, 0, 0x401F8360U
- #define **IOMUXC_GPIO_B0_13_SRC_BOOT_CFG09** 0x401F8170U, 0x6U, 0, 0, 0x401F8360U
- #define **IOMUXC_GPIO_B0_14_LCD_DATA10** 0x401F8174U, 0x0U, 0, 0, 0x401F8364U
- #define **IOMUXC_GPIO_B0_14_XBAR1_INOUT12** 0x401F8174U, 0x1U, 0, 0, 0x401F8364U
- #define **IOMUXC_GPIO_B0_14_ARM_CM7_TXEV** 0x401F8174U, 0x2U, 0, 0, 0x401F8364U
- #define **IOMUXC_GPIO_B0_14_SAI1_RX_SYNC** 0x401F8174U, 0x3U, 0x401F85A4U, 0x2U, 0x401F8364U
- #define **IOMUXC_GPIO_B0_14_FLEXIO2_FLEXIO14** 0x401F8174U, 0x4U, 0, 0, 0x401F8364U
- #define **IOMUXC_GPIO_B0_14_GPIO2_IO14** 0x401F8174U, 0x5U, 0, 0, 0x401F8364U
- #define **IOMUXC_GPIO_B0_14_SRC_BOOT_CFG10** 0x401F8174U, 0x6U, 0, 0, 0x401F8364U
- #define **IOMUXC_GPIO_B0_15_LCD_DATA11** 0x401F8178U, 0x0U, 0, 0, 0x401F8368U
- #define **IOMUXC_GPIO_B0_15_XBAR1_INOUT13** 0x401F8178U, 0x1U, 0, 0, 0x401F8368U
- #define **IOMUXC_GPIO_B0_15_ARM_CM7_RXEV** 0x401F8178U, 0x2U, 0, 0, 0x401F8368U
- #define **IOMUXC_GPIO_B0_15_SAI1_RX_BCLK** 0x401F8178U, 0x3U, 0x401F8590U, 0x2U, 0x401F8368U
- #define **IOMUXC_GPIO_B0_15_FLEXIO2_FLEXIO15** 0x401F8178U, 0x4U, 0, 0, 0x401F8368U
- #define **IOMUXC_GPIO_B0_15_GPIO2_IO15** 0x401F8178U, 0x5U, 0, 0, 0x401F8368U
- #define **IOMUXC_GPIO_B0_15_SRC_BOOT_CFG11** 0x401F8178U, 0x6U, 0, 0, 0x401F8368U
- #define **IOMUXC_GPIO_B1_00_LCD_DATA12** 0x401F817CU, 0x0U, 0, 0, 0x401F836CU
- #define **IOMUXC_GPIO_B1_00_XBAR1_INOUT14** 0x401F817CU, 0x1U, 0x401F8644U, 0x1U, 0x401F836CU
- #define **IOMUXC_GPIO_B1_00_LPUART4_TX** 0x401F817CU, 0x2U, 0x401F8544U, 0x2U,

0x401F836CU
- #define **IOMUXC_GPIO_B1_00_SAI1_RX_DATA00** 0x401F817CU, 0x3U, 0x401F8594U, 0x2U, 0x401F836CU
- #define **IOMUXC_GPIO_B1_00_FLEXIO2_FLEXIO16** 0x401F817CU, 0x4U, 0, 0, 0x401F836CU
- #define **IOMUXC_GPIO_B1_00_GPIO2_IO16** 0x401F817CU, 0x5U, 0, 0, 0x401F836CU
- #define **IOMUXC_GPIO_B1_00_FLEXPWM1_PWMA03** 0x401F817CU, 0x6U, 0x401F8454U, 0x4U, 0x401F836CU
- #define **IOMUXC_GPIO_B1_01_LCD_DATA13** 0x401F8180U, 0x0U, 0, 0, 0x401F8370U
- #define **IOMUXC_GPIO_B1_01_XBAR1_INOUT15** 0x401F8180U, 0x1U, 0x401F8648U, 0x1U, 0x401F8370U
- #define **IOMUXC_GPIO_B1_01_LPUART4_RX** 0x401F8180U, 0x2U, 0x401F8540U, 0x2U, 0x401F8370U
- #define **IOMUXC_GPIO_B1_01_SAI1_TX_DATA00** 0x401F8180U, 0x3U, 0, 0, 0x401F8370U
- #define **IOMUXC_GPIO_B1_01_FLEXIO2_FLEXIO17** 0x401F8180U, 0x4U, 0, 0, 0x401F8370U
- #define **IOMUXC_GPIO_B1_01_GPIO2_IO17** 0x401F8180U, 0x5U, 0, 0, 0x401F8370U
- #define **IOMUXC_GPIO_B1_01_FLEXPWM1_PWMB03** 0x401F8180U, 0x6U, 0x401F8464U, 0x4U, 0x401F8370U
- #define **IOMUXC_GPIO_B1_02_LCD_DATA14** 0x401F8184U, 0x0U, 0, 0, 0x401F8374U
- #define **IOMUXC_GPIO_B1_02_XBAR1_INOUT16** 0x401F8184U, 0x1U, 0x401F864CU, 0x1U, 0x401F8374U
- #define **IOMUXC_GPIO_B1_02_LPSPI4_PCS2** 0x401F8184U, 0x2U, 0, 0, 0x401F8374U
- #define **IOMUXC_GPIO_B1_02_SAI1_TX_BCLK** 0x401F8184U, 0x3U, 0x401F85A8U, 0x2U, 0x401F8374U
- #define **IOMUXC_GPIO_B1_02_FLEXIO2_FLEXIO18** 0x401F8184U, 0x4U, 0, 0, 0x401F8374U
- #define **IOMUXC_GPIO_B1_02_GPIO2_IO18** 0x401F8184U, 0x5U, 0, 0, 0x401F8374U
- #define **IOMUXC_GPIO_B1_02_FLEXPWM2_PWMA03** 0x401F8184U, 0x6U, 0x401F8474U, 0x4U, 0x401F8374U
- #define **IOMUXC_GPIO_B1_03_LCD_DATA15** 0x401F8188U, 0x0U, 0, 0, 0x401F8378U
- #define **IOMUXC_GPIO_B1_03_XBAR1_INOUT17** 0x401F8188U, 0x1U, 0x401F862CU, 0x3U, 0x401F8378U
- #define **IOMUXC_GPIO_B1_03_LPSPI4_PCS1** 0x401F8188U, 0x2U, 0, 0, 0x401F8378U
- #define **IOMUXC_GPIO_B1_03_SAI1_TX_SYNC** 0x401F8188U, 0x3U, 0x401F85ACU, 0x2U, 0x401F8378U
- #define **IOMUXC_GPIO_B1_03_FLEXIO2_FLEXIO19** 0x401F8188U, 0x4U, 0, 0, 0x401F8378U
- #define **IOMUXC_GPIO_B1_03_GPIO2_IO19** 0x401F8188U, 0x5U, 0, 0, 0x401F8378U
- #define **IOMUXC_GPIO_B1_03_FLEXPWM2_PWMB03** 0x401F8188U, 0x6U, 0x401F8484U, 0x3U, 0x401F8378U
- #define **IOMUXC_GPIO_B1_04_LCD_DATA16** 0x401F818CU, 0x0U, 0, 0, 0x401F837CU
- #define **IOMUXC_GPIO_B1_04_LPSPI4_PCS0** 0x401F818CU, 0x1U, 0x401F851CU, 0x1U, 0x401F837CU
- #define **IOMUXC_GPIO_B1_04_CSI_DATA15** 0x401F818CU, 0x2U, 0, 0, 0x401F837CU
- #define **IOMUXC_GPIO_B1_04_ENET_RX_DATA00** 0x401F818CU, 0x3U, 0x401F8434U, 0x1U, 0x401F837CU
- #define **IOMUXC_GPIO_B1_04_FLEXIO2_FLEXIO20** 0x401F818CU, 0x4U, 0, 0, 0x401F837CU
- #define **IOMUXC_GPIO_B1_04_GPIO2_IO20** 0x401F818CU, 0x5U, 0, 0, 0x401F837CU
- #define **IOMUXC_GPIO_B1_05_LCD_DATA17** 0x401F8190U, 0x0U, 0, 0, 0x401F8380U

- #define **IOMUXC_GPIO_B1_05_LPSPI4_SDI** 0x401F8190U, 0x1U, 0x401F8524U, 0x1U, 0x401F8380U
- #define **IOMUXC_GPIO_B1_05_CSI_DATA14** 0x401F8190U, 0x2U, 0, 0, 0x401F8380U
- #define **IOMUXC_GPIO_B1_05_ENET_RX_DATA01** 0x401F8190U, 0x3U, 0x401F8438U, 0x1U, 0x401F8380U
- #define **IOMUXC_GPIO_B1_05_FLEXIO2_FLEXIO21** 0x401F8190U, 0x4U, 0, 0, 0x401F8380U
- #define **IOMUXC_GPIO_B1_05_GPIO2_IO21** 0x401F8190U, 0x5U, 0, 0, 0x401F8380U
- #define **IOMUXC_GPIO_B1_06_LCD_DATA18** 0x401F8194U, 0x0U, 0, 0, 0x401F8384U
- #define **IOMUXC_GPIO_B1_06_LPSPI4_SDO** 0x401F8194U, 0x1U, 0x401F8528U, 0x1U, 0x401F8384U
- #define **IOMUXC_GPIO_B1_06_CSI_DATA13** 0x401F8194U, 0x2U, 0, 0, 0x401F8384U
- #define **IOMUXC_GPIO_B1_06_ENET_RX_EN** 0x401F8194U, 0x3U, 0x401F843CU, 0x1U, 0x401F8384U
- #define **IOMUXC_GPIO_B1_06_FLEXIO2_FLEXIO22** 0x401F8194U, 0x4U, 0, 0, 0x401F8384U
- #define **IOMUXC_GPIO_B1_06_GPIO2_IO22** 0x401F8194U, 0x5U, 0, 0, 0x401F8384U
- #define **IOMUXC_GPIO_B1_07_LCD_DATA19** 0x401F8198U, 0x0U, 0, 0, 0x401F8388U
- #define **IOMUXC_GPIO_B1_07_LPSPI4_SCK** 0x401F8198U, 0x1U, 0x401F8520U, 0x1U, 0x401F8388U
- #define **IOMUXC_GPIO_B1_07_CSI_DATA12** 0x401F8198U, 0x2U, 0, 0, 0x401F8388U
- #define **IOMUXC_GPIO_B1_07_ENET_TX_DATA00** 0x401F8198U, 0x3U, 0, 0, 0x401F8388U
- #define **IOMUXC_GPIO_B1_07_FLEXIO2_FLEXIO23** 0x401F8198U, 0x4U, 0, 0, 0x401F8388U
- #define **IOMUXC_GPIO_B1_07_GPIO2_IO23** 0x401F8198U, 0x5U, 0, 0, 0x401F8388U
- #define **IOMUXC_GPIO_B1_08_LCD_DATA20** 0x401F819CU, 0x0U, 0, 0, 0x401F838CU
- #define **IOMUXC_GPIO_B1_08_QTIMER1_TIMER3** 0x401F819CU, 0x1U, 0, 0, 0x401F838CU
- #define **IOMUXC_GPIO_B1_08_CSI_DATA11** 0x401F819CU, 0x2U, 0, 0, 0x401F838CU
- #define **IOMUXC_GPIO_B1_08_ENET_TX_DATA01** 0x401F819CU, 0x3U, 0, 0, 0x401F838CU
- #define **IOMUXC_GPIO_B1_08_FLEXIO2_FLEXIO24** 0x401F819CU, 0x4U, 0, 0, 0x401F838CU
- #define **IOMUXC_GPIO_B1_08_GPIO2_IO24** 0x401F819CU, 0x5U, 0, 0, 0x401F838CU
- #define **IOMUXC_GPIO_B1_08_FLEXCAN2_TX** 0x401F819CU, 0x6U, 0, 0, 0x401F838CU
- #define **IOMUXC_GPIO_B1_09_LCD_DATA21** 0x401F81A0U, 0x0U, 0, 0, 0x401F8390U
- #define **IOMUXC_GPIO_B1_09_QTIMER2_TIMER3** 0x401F81A0U, 0x1U, 0x401F8578U, 0x1U, 0x401F8390U
- #define **IOMUXC_GPIO_B1_09_CSI_DATA10** 0x401F81A0U, 0x2U, 0, 0, 0x401F8390U
- #define **IOMUXC_GPIO_B1_09_ENET_TX_EN** 0x401F81A0U, 0x3U, 0, 0, 0x401F8390U
- #define **IOMUXC_GPIO_B1_09_FLEXIO2_FLEXIO25** 0x401F81A0U, 0x4U, 0, 0, 0x401F8390U
- #define **IOMUXC_GPIO_B1_09_GPIO2_IO25** 0x401F81A0U, 0x5U, 0, 0, 0x401F8390U
- #define **IOMUXC_GPIO_B1_09_FLEXCAN2_RX** 0x401F81A0U, 0x6U, 0x401F8450U, 0x3U, 0x401F8390U
- #define **IOMUXC_GPIO_B1_10_LCD_DATA22** 0x401F81A4U, 0x0U, 0, 0, 0x401F8394U
- #define **IOMUXC_GPIO_B1_10_QTIMER3_TIMER3** 0x401F81A4U, 0x1U, 0x401F8588U, 0x2U, 0x401F8394U
- #define **IOMUXC_GPIO_B1_10_CSI_DATA00** 0x401F81A4U, 0x2U, 0, 0, 0x401F8394U
- #define **IOMUXC_GPIO_B1_10_ENET_TX_CLK** 0x401F81A4U, 0x3U, 0x401F8448U, 0x1U, 0x401F8394U

- #define **IOMUXC_GPIO_B1_10_FLEXIO2_FLEXIO26** 0x401F81A4U, 0x4U, 0, 0, 0x401F8394U
- #define **IOMUXC_GPIO_B1_10_GPIO2_IO26** 0x401F81A4U, 0x5U, 0, 0, 0x401F8394U
- #define **IOMUXC_GPIO_B1_10_ENET_REF_CLK** 0x401F81A4U, 0x6U, 0x401F842CU, 0x1U, 0x401F8394U
- #define **IOMUXC_GPIO_B1_11_LCD_DATA23** 0x401F81A8U, 0x0U, 0, 0, 0x401F8398U
- #define **IOMUXC_GPIO_B1_11_QTIMER4_TIMER3** 0x401F81A8U, 0x1U, 0, 0, 0x401F8398U
- #define **IOMUXC_GPIO_B1_11_CSI_DATA01** 0x401F81A8U, 0x2U, 0, 0, 0x401F8398U
- #define **IOMUXC_GPIO_B1_11_ENET_RX_ER** 0x401F81A8U, 0x3U, 0x401F8440U, 0x1U, 0x401F8398U
- #define **IOMUXC_GPIO_B1_11_FLEXIO2_FLEXIO27** 0x401F81A8U, 0x4U, 0, 0, 0x401F8398U
- #define **IOMUXC_GPIO_B1_11_GPIO2_IO27** 0x401F81A8U, 0x5U, 0, 0, 0x401F8398U
- #define **IOMUXC_GPIO_B1_11_LPSPI4_PCS3** 0x401F81A8U, 0x6U, 0, 0, 0x401F8398U
- #define **IOMUXC_GPIO_B1_12_LPUART5_TX** 0x401F81ACU, 0x1U, 0x401F854CU, 0x1U, 0x401F839CU
- #define **IOMUXC_GPIO_B1_12_CSI_PIXCLK** 0x401F81ACU, 0x2U, 0x401F8424U, 0x1U, 0x401F839CU
- #define **IOMUXC_GPIO_B1_12_ENET_1588_EVENT0_IN** 0x401F81ACU, 0x3U, 0x401F8444U, 0x2U, 0x401F839CU
- #define **IOMUXC_GPIO_B1_12_FLEXIO2_FLEXIO28** 0x401F81ACU, 0x4U, 0, 0, 0x401F839CU
- #define **IOMUXC_GPIO_B1_12_GPIO2_IO28** 0x401F81ACU, 0x5U, 0, 0, 0x401F839CU
- #define **IOMUXC_GPIO_B1_12_USDHC1_CD_B** 0x401F81ACU, 0x6U, 0x401F85D4U, 0x2U, 0x401F839CU
- #define **IOMUXC_GPIO_B1_13_WDOG1_B** 0x401F81B0U, 0x0U, 0, 0, 0x401F83A0U
- #define **IOMUXC_GPIO_B1_13_LPUART5_RX** 0x401F81B0U, 0x1U, 0x401F8548U, 0x1U, 0x401F83A0U
- #define **IOMUXC_GPIO_B1_13_CSI_VSYNC** 0x401F81B0U, 0x2U, 0x401F8428U, 0x2U, 0x401F83A0U
- #define **IOMUXC_GPIO_B1_13_ENET_1588_EVENT0_OUT** 0x401F81B0U, 0x3U, 0, 0, 0x401F83A0U
- #define **IOMUXC_GPIO_B1_13_FLEXIO2_FLEXIO29** 0x401F81B0U, 0x4U, 0, 0, 0x401F83A0U
- #define **IOMUXC_GPIO_B1_13_GPIO2_IO29** 0x401F81B0U, 0x5U, 0, 0, 0x401F83A0U
- #define **IOMUXC_GPIO_B1_13_USDHC1_WP** 0x401F81B0U, 0x6U, 0x401F85D8U, 0x3U, 0x401F83A0U
- #define **IOMUXC_GPIO_B1_14_ENET_MDC** 0x401F81B4U, 0x0U, 0, 0, 0x401F83A4U
- #define **IOMUXC_GPIO_B1_14_FLEXPWM4_PWMA02** 0x401F81B4U, 0x1U, 0x401F849CU, 0x1U, 0x401F83A4U
- #define **IOMUXC_GPIO_B1_14_CSI_HSYNC** 0x401F81B4U, 0x2U, 0x401F8420U, 0x2U, 0x401F83A4U
- #define **IOMUXC_GPIO_B1_14_XBAR1_IN02** 0x401F81B4U, 0x3U, 0x401F860CU, 0x1U, 0x401F83A4U
- #define **IOMUXC_GPIO_B1_14_FLEXIO2_FLEXIO30** 0x401F81B4U, 0x4U, 0, 0, 0x401F83A4U
- #define **IOMUXC_GPIO_B1_14_GPIO2_IO30** 0x401F81B4U, 0x5U, 0, 0, 0x401F83A4U
- #define **IOMUXC_GPIO_B1_14_USDHC1_VSELECT** 0x401F81B4U, 0x6U, 0, 0, 0x401F83A4U
- #define **IOMUXC_GPIO_B1_15_ENET_MDIO** 0x401F81B8U, 0x0U, 0x401F8430U, 0x2U,

0x401F83A8U
- #define **IOMUXC_GPIO_B1_15_FLEXPWM4_PWMA03** 0x401F81B8U, 0x1U, 0x401F84A0U, 0x1U, 0x401F83A8U
- #define **IOMUXC_GPIO_B1_15_CSI_MCLK** 0x401F81B8U, 0x2U, 0, 0, 0x401F83A8U
- #define **IOMUXC_GPIO_B1_15_XBAR1_IN03** 0x401F81B8U, 0x3U, 0x401F8610U, 0x1U, 0x401F83A8U
- #define **IOMUXC_GPIO_B1_15_FLEXIO2_FLEXIO31** 0x401F81B8U, 0x4U, 0, 0, 0x401F83A8U
- #define **IOMUXC_GPIO_B1_15_GPIO2_IO31** 0x401F81B8U, 0x5U, 0, 0, 0x401F83A8U
- #define **IOMUXC_GPIO_B1_15_USDHC1_RESET_B** 0x401F81B8U, 0x6U, 0, 0, 0x401F83A8U
- #define **IOMUXC_GPIO_SD_B0_00_USDHC1_CMD** 0x401F81BCU, 0x0U, 0, 0, 0x401F83ACU
- #define **IOMUXC_GPIO_SD_B0_00_FLEXPWM1_PWMA00** 0x401F81BCU, 0x1U, 0x401F8458U, 0x1U, 0x401F83ACU
- #define **IOMUXC_GPIO_SD_B0_00_LPI2C3_SCL** 0x401F81BCU, 0x2U, 0x401F84DCU, 0x1U, 0x401F83ACU
- #define **IOMUXC_GPIO_SD_B0_00_XBAR1_INOUT04** 0x401F81BCU, 0x3U, 0x401F8614U, 0x1U, 0x401F83ACU
- #define **IOMUXC_GPIO_SD_B0_00_LPSPI1_SCK** 0x401F81BCU, 0x4U, 0x401F84F0U, 0x1U, 0x401F83ACU
- #define **IOMUXC_GPIO_SD_B0_00_GPIO3_IO12** 0x401F81BCU, 0x5U, 0, 0, 0x401F83ACU
- #define **IOMUXC_GPIO_SD_B0_00_FLEXSPIA_SS1_B** 0x401F81BCU, 0x6U, 0, 0, 0x401F83ACU
- #define **IOMUXC_GPIO_SD_B0_01_USDHC1_CLK** 0x401F81C0U, 0x0U, 0, 0, 0x401F83B0U
- #define **IOMUXC_GPIO_SD_B0_01_FLEXPWM1_PWMB00** 0x401F81C0U, 0x1U, 0x401F8468U, 0x1U, 0x401F83B0U
- #define **IOMUXC_GPIO_SD_B0_01_LPI2C3_SDA** 0x401F81C0U, 0x2U, 0x401F84E0U, 0x1U, 0x401F83B0U
- #define **IOMUXC_GPIO_SD_B0_01_XBAR1_INOUT05** 0x401F81C0U, 0x3U, 0x401F8618U, 0x1U, 0x401F83B0U
- #define **IOMUXC_GPIO_SD_B0_01_LPSPI1_PCS0** 0x401F81C0U, 0x4U, 0x401F84ECU, 0x0U, 0x401F83B0U
- #define **IOMUXC_GPIO_SD_B0_01_GPIO3_IO13** 0x401F81C0U, 0x5U, 0, 0, 0x401F83B0U
- #define **IOMUXC_GPIO_SD_B0_01_FLEXSPIB_SS1_B** 0x401F81C0U, 0x6U, 0, 0, 0x401F83B0U
- #define **IOMUXC_GPIO_SD_B0_02_USDHC1_DATA0** 0x401F81C4U, 0x0U, 0, 0, 0x401F83B4U
- #define **IOMUXC_GPIO_SD_B0_02_FLEXPWM1_PWMA01** 0x401F81C4U, 0x1U, 0x401F845CU, 0x1U, 0x401F83B4U
- #define **IOMUXC_GPIO_SD_B0_02_LPUART8_CTS_B** 0x401F81C4U, 0x2U, 0, 0, 0x401F83B4U
- #define **IOMUXC_GPIO_SD_B0_02_XBAR1_INOUT06** 0x401F81C4U, 0x3U, 0x401F861CU, 0x1U, 0x401F83B4U
- #define **IOMUXC_GPIO_SD_B0_02_LPSPI1_SDO** 0x401F81C4U, 0x4U, 0x401F84F8U, 0x1U, 0x401F83B4U
- #define **IOMUXC_GPIO_SD_B0_02_GPIO3_IO14** 0x401F81C4U, 0x5U, 0, 0, 0x401F83B4U
- #define **IOMUXC_GPIO_SD_B0_03_USDHC1_DATA1** 0x401F81C8U, 0x0U, 0, 0, 0x401F83B8U
- #define **IOMUXC_GPIO_SD_B0_03_FLEXPWM1_PWMB01** 0x401F81C8U, 0x1U, 0x401-

F846CU, 0x1U, 0x401F83B8U
- #define **IOMUXC_GPIO_SD_B0_03_LPUART8_RTS_B** 0x401F81C8U, 0x2U, 0, 0, 0x401F83B8U
- #define **IOMUXC_GPIO_SD_B0_03_XBAR1_INOUT07** 0x401F81C8U, 0x3U, 0x401F8620U, 0x1U, 0x401F83B8U
- #define **IOMUXC_GPIO_SD_B0_03_LPSPI1_SDI** 0x401F81C8U, 0x4U, 0x401F84F4U, 0x1U, 0x401F83B8U
- #define **IOMUXC_GPIO_SD_B0_03_GPIO3_IO15** 0x401F81C8U, 0x5U, 0, 0, 0x401F83B8U
- #define **IOMUXC_GPIO_SD_B0_04_USDHC1_DATA2** 0x401F81CCU, 0x0U, 0, 0, 0x401F83BCU
- #define **IOMUXC_GPIO_SD_B0_04_FLEXPWM1_PWMA02** 0x401F81CCU, 0x1U, 0x401F8460U, 0x1U, 0x401F83BCU
- #define **IOMUXC_GPIO_SD_B0_04_LPUART8_TX** 0x401F81CCU, 0x2U, 0x401F8564U, 0x0U, 0x401F83BCU
- #define **IOMUXC_GPIO_SD_B0_04_XBAR1_INOUT08** 0x401F81CCU, 0x3U, 0x401F8624U, 0x1U, 0x401F83BCU
- #define **IOMUXC_GPIO_SD_B0_04_FLEXSPIB_SS0_B** 0x401F81CCU, 0x4U, 0, 0, 0x401F83BCU
- #define **IOMUXC_GPIO_SD_B0_04_GPIO3_IO16** 0x401F81CCU, 0x5U, 0, 0, 0x401F83BCU
- #define **IOMUXC_GPIO_SD_B0_04_CCM_CLKO1** 0x401F81CCU, 0x6U, 0, 0, 0x401F83BCU
- #define **IOMUXC_GPIO_SD_B0_05_USDHC1_DATA3** 0x401F81D0U, 0x0U, 0, 0, 0x401F83C0U
- #define **IOMUXC_GPIO_SD_B0_05_FLEXPWM1_PWMB02** 0x401F81D0U, 0x1U, 0x401F8470U, 0x1U, 0x401F83C0U
- #define **IOMUXC_GPIO_SD_B0_05_LPUART8_RX** 0x401F81D0U, 0x2U, 0x401F8560U, 0x0U, 0x401F83C0U
- #define **IOMUXC_GPIO_SD_B0_05_XBAR1_INOUT09** 0x401F81D0U, 0x3U, 0x401F8628U, 0x1U, 0x401F83C0U
- #define **IOMUXC_GPIO_SD_B0_05_FLEXSPIB_DQS** 0x401F81D0U, 0x4U, 0, 0, 0x401F83C0U
- #define **IOMUXC_GPIO_SD_B0_05_GPIO3_IO17** 0x401F81D0U, 0x5U, 0, 0, 0x401F83C0U
- #define **IOMUXC_GPIO_SD_B0_05_CCM_CLKO2** 0x401F81D0U, 0x6U, 0, 0, 0x401F83C0U
- #define **IOMUXC_GPIO_SD_B1_00_USDHC2_DATA3** 0x401F81D4U, 0x0U, 0x401F85F4U, 0x0U, 0x401F83C4U
- #define **IOMUXC_GPIO_SD_B1_00_FLEXSPIB_DATA03** 0x401F81D4U, 0x1U, 0x401F84C4U, 0x0U, 0x401F83C4U
- #define **IOMUXC_GPIO_SD_B1_00_FLEXPWM1_PWMA03** 0x401F81D4U, 0x2U, 0x401F8454U, 0x0U, 0x401F83C4U
- #define **IOMUXC_GPIO_SD_B1_00_SAI1_TX_DATA03** 0x401F81D4U, 0x3U, 0x401F8598U, 0x0U, 0x401F83C4U
- #define **IOMUXC_GPIO_SD_B1_00_LPUART4_TX** 0x401F81D4U, 0x4U, 0x401F8544U, 0x0U, 0x401F83C4U
- #define **IOMUXC_GPIO_SD_B1_00_GPIO3_IO00** 0x401F81D4U, 0x5U, 0, 0, 0x401F83C4U
- #define **IOMUXC_GPIO_SD_B1_01_USDHC2_DATA2** 0x401F81D8U, 0x0U, 0x401F85F0U, 0x0U, 0x401F83C8U
- #define **IOMUXC_GPIO_SD_B1_01_FLEXSPIB_DATA02** 0x401F81D8U, 0x1U, 0x401F84C0U, 0x0U, 0x401F83C8U
- #define **IOMUXC_GPIO_SD_B1_01_FLEXPWM1_PWMB03** 0x401F81D8U, 0x2U, 0x401F8464U, 0x0U, 0x401F83C8U
- #define **IOMUXC_GPIO_SD_B1_01_SAI1_TX_DATA02** 0x401F81D8U, 0x3U, 0x401F859C-

U, 0x0U, 0x401F83C8U
- #define **IOMUXC_GPIO_SD_B1_01_LPUART4_RX** 0x401F81D8U, 0x4U, 0x401F8540U, 0x0U, 0x401F83C8U
- #define **IOMUXC_GPIO_SD_B1_01_GPIO3_IO01** 0x401F81D8U, 0x5U, 0, 0, 0x401F83C8U
- #define **IOMUXC_GPIO_SD_B1_02_USDHC2_DATA1** 0x401F81DCU, 0x0U, 0x401F85ECU, 0x0U, 0x401F83CCU
- #define **IOMUXC_GPIO_SD_B1_02_FLEXSPIB_DATA01** 0x401F81DCU, 0x1U, 0x401F84BCU, 0x0U, 0x401F83CCU
- #define **IOMUXC_GPIO_SD_B1_02_FLEXPWM2_PWMA03** 0x401F81DCU, 0x2U, 0x401F8474U, 0x0U, 0x401F83CCU
- #define **IOMUXC_GPIO_SD_B1_02_SAI1_TX_DATA01** 0x401F81DCU, 0x3U, 0x401F85A0U, 0x0U, 0x401F83CCU
- #define **IOMUXC_GPIO_SD_B1_02_FLEXCAN1_TX** 0x401F81DCU, 0x4U, 0, 0, 0x401F83CCU
- #define **IOMUXC_GPIO_SD_B1_02_GPIO3_IO02** 0x401F81DCU, 0x5U, 0, 0, 0x401F83CCU
- #define **IOMUXC_GPIO_SD_B1_02_CCM_WAIT** 0x401F81DCU, 0x6U, 0, 0, 0x401F83CCU
- #define **IOMUXC_GPIO_SD_B1_03_USDHC2_DATA0** 0x401F81E0U, 0x0U, 0x401F85E8U, 0x0U, 0x401F83D0U
- #define **IOMUXC_GPIO_SD_B1_03_FLEXSPIB_DATA00** 0x401F81E0U, 0x1U, 0x401F84B8U, 0x0U, 0x401F83D0U
- #define **IOMUXC_GPIO_SD_B1_03_FLEXPWM2_PWMB03** 0x401F81E0U, 0x2U, 0x401F8484U, 0x0U, 0x401F83D0U
- #define **IOMUXC_GPIO_SD_B1_03_SAI1_MCLK** 0x401F81E0U, 0x3U, 0x401F858CU, 0x0U, 0x401F83D0U
- #define **IOMUXC_GPIO_SD_B1_03_FLEXCAN1_RX** 0x401F81E0U, 0x4U, 0x401F844CU, 0x0U, 0x401F83D0U
- #define **IOMUXC_GPIO_SD_B1_03_GPIO3_IO03** 0x401F81E0U, 0x5U, 0, 0, 0x401F83D0U
- #define **IOMUXC_GPIO_SD_B1_03_CCM_PMIC_READY** 0x401F81E0U, 0x6U, 0x401F83FCU, 0x0U, 0x401F83D0U
- #define **IOMUXC_GPIO_SD_B1_04_USDHC2_CLK** 0x401F81E4U, 0x0U, 0x401F85DCU, 0x0U, 0x401F83D4U
- #define **IOMUXC_GPIO_SD_B1_04_FLEXSPIB_SCLK** 0x401F81E4U, 0x1U, 0, 0, 0x401F83D4U
- #define **IOMUXC_GPIO_SD_B1_04_LPI2C1_SCL** 0x401F81E4U, 0x2U, 0x401F84CCU, 0x0U, 0x401F83D4U
- #define **IOMUXC_GPIO_SD_B1_04_SAI1_RX_SYNC** 0x401F81E4U, 0x3U, 0x401F85A4U, 0x0U, 0x401F83D4U
- #define **IOMUXC_GPIO_SD_B1_04_FLEXSPIA_SS1_B** 0x401F81E4U, 0x4U, 0, 0, 0x401F83D4U
- #define **IOMUXC_GPIO_SD_B1_04_GPIO3_IO04** 0x401F81E4U, 0x5U, 0, 0, 0x401F83D4U
- #define **IOMUXC_GPIO_SD_B1_04_CCM_STOP** 0x401F81E4U, 0x6U, 0, 0, 0x401F83D4U
- #define **IOMUXC_GPIO_SD_B1_05_USDHC2_CMD** 0x401F81E8U, 0x0U, 0x401F85E4U, 0x0U, 0x401F83D8U
- #define **IOMUXC_GPIO_SD_B1_05_FLEXSPIA_DQS** 0x401F81E8U, 0x1U, 0x401F84A4U, 0x0U, 0x401F83D8U
- #define **IOMUXC_GPIO_SD_B1_05_LPI2C1_SDA** 0x401F81E8U, 0x2U, 0x401F84D0U, 0x0U, 0x401F83D8U
- #define **IOMUXC_GPIO_SD_B1_05_SAI1_RX_BCLK** 0x401F81E8U, 0x3U, 0x401F8590U, 0x0U, 0x401F83D8U
- #define **IOMUXC_GPIO_SD_B1_05_FLEXSPIB_SS0_B** 0x401F81E8U, 0x4U, 0, 0, 0x401F83-

D8U
- #define **IOMUXC_GPIO_SD_B1_05_GPIO3_IO05** 0x401F81E8U, 0x5U, 0, 0, 0x401F83D8U
- #define **IOMUXC_GPIO_SD_B1_06_USDHC2_RESET_B** 0x401F81ECU, 0x0U, 0, 0, 0x401-F83DCU
- #define **IOMUXC_GPIO_SD_B1_06_FLEXSPIA_SS0_B** 0x401F81ECU, 0x1U, 0, 0, 0x401-F83DCU
- #define **IOMUXC_GPIO_SD_B1_06_LPUART7_CTS_B** 0x401F81ECU, 0x2U, 0, 0, 0x401-F83DCU
- #define **IOMUXC_GPIO_SD_B1_06_SAI1_RX_DATA00** 0x401F81ECU, 0x3U, 0x401F8594-U, 0x0U, 0x401F83DCU
- #define **IOMUXC_GPIO_SD_B1_06_LPSPI2_PCS0** 0x401F81ECU, 0x4U, 0x401F84FCU, 0x0U, 0x401F83DCU
- #define **IOMUXC_GPIO_SD_B1_06_GPIO3_IO06** 0x401F81ECU, 0x5U, 0, 0, 0x401F83DCU
- #define **IOMUXC_GPIO_SD_B1_07_SEMC_CSX01** 0x401F81F0U, 0x0U, 0, 0, 0x401F83E0U
- #define **IOMUXC_GPIO_SD_B1_07_FLEXSPIA_SCLK** 0x401F81F0U, 0x1U, 0x401F84C8U, 0x0U, 0x401F83E0U
- #define **IOMUXC_GPIO_SD_B1_07_LPUART7_RTS_B** 0x401F81F0U, 0x2U, 0, 0, 0x401F83-E0U
- #define **IOMUXC_GPIO_SD_B1_07_SAI1_TX_DATA00** 0x401F81F0U, 0x3U, 0, 0, 0x401-F83E0U
- #define **IOMUXC_GPIO_SD_B1_07_LPSPI2_SCK** 0x401F81F0U, 0x4U, 0x401F8500U, 0x0U, 0x401F83E0U
- #define **IOMUXC_GPIO_SD_B1_07_GPIO3_IO07** 0x401F81F0U, 0x5U, 0, 0, 0x401F83E0U
- #define **IOMUXC_GPIO_SD_B1_08_USDHC2_DATA4** 0x401F81F4U, 0x0U, 0x401F85F8U, 0x0U, 0x401F83E4U
- #define **IOMUXC_GPIO_SD_B1_08_FLEXSPIA_DATA00** 0x401F81F4U, 0x1U, 0x401F84-A8U, 0x0U, 0x401F83E4U
- #define **IOMUXC_GPIO_SD_B1_08_LPUART7_TX** 0x401F81F4U, 0x2U, 0x401F855CU, 0x0U, 0x401F83E4U
- #define **IOMUXC_GPIO_SD_B1_08_SAI1_TX_BCLK** 0x401F81F4U, 0x3U, 0x401F85A8U, 0x0U, 0x401F83E4U
- #define **IOMUXC_GPIO_SD_B1_08_LPSPI2_SD0** 0x401F81F4U, 0x4U, 0x401F8508U, 0x0U, 0x401F83E4U
- #define **IOMUXC_GPIO_SD_B1_08_GPIO3_IO08** 0x401F81F4U, 0x5U, 0, 0, 0x401F83E4U
- #define **IOMUXC_GPIO_SD_B1_08_SEMC_CSX02** 0x401F81F4U, 0x6U, 0, 0, 0x401F83E4U
- #define **IOMUXC_GPIO_SD_B1_09_USDHC2_DATA5** 0x401F81F8U, 0x0U, 0x401F85FCU, 0x0U, 0x401F83E8U
- #define **IOMUXC_GPIO_SD_B1_09_FLEXSPIA_DATA01** 0x401F81F8U, 0x1U, 0x401F84A-CU, 0x0U, 0x401F83E8U
- #define **IOMUXC_GPIO_SD_B1_09_LPUART7_RX** 0x401F81F8U, 0x2U, 0x401F8558U, 0x0-U, 0x401F83E8U
- #define **IOMUXC_GPIO_SD_B1_09_SAI1_TX_SYNC** 0x401F81F8U, 0x3U, 0x401F85ACU, 0x0U, 0x401F83E8U
- #define **IOMUXC_GPIO_SD_B1_09_LPSPI2_SDI** 0x401F81F8U, 0x4U, 0x401F8504U, 0x0U, 0x401F83E8U
- #define **IOMUXC_GPIO_SD_B1_09_GPIO3_IO09** 0x401F81F8U, 0x5U, 0, 0, 0x401F83E8U
- #define **IOMUXC_GPIO_SD_B1_10_USDHC2_DATA6** 0x401F81FCU, 0x0U, 0x401F8600U, 0x0U, 0x401F83ECU
- #define **IOMUXC_GPIO_SD_B1_10_FLEXSPIA_DATA02** 0x401F81FCU, 0x1U, 0x401F84-B0U, 0x0U, 0x401F83ECU
- #define **IOMUXC_GPIO_SD_B1_10_LPUART2_RX** 0x401F81FCU, 0x2U, 0x401F852CU,

0x0U, 0x401F83ECU
- #define **IOMUXC_GPIO_SD_B1_10_LPI2C2_SDA** 0x401F81FCU, 0x3U, 0x401F84D8U, 0x0U, 0x401F83ECU
- #define **IOMUXC_GPIO_SD_B1_10_LPSPI2_PCS2** 0x401F81FCU, 0x4U, 0, 0, 0x401F83ECU
- #define **IOMUXC_GPIO_SD_B1_10_GPIO3_IO10** 0x401F81FCU, 0x5U, 0, 0, 0x401F83ECU
- #define **IOMUXC_GPIO_SD_B1_11_USDHC2_DATA7** 0x401F8200U, 0x0U, 0x401F8604U, 0x0U, 0x401F83F0U
- #define **IOMUXC_GPIO_SD_B1_11_FLEXSPIA_DATA03** 0x401F8200U, 0x1U, 0x401F84B4U, 0x0U, 0x401F83F0U
- #define **IOMUXC_GPIO_SD_B1_11_LPUART2_TX** 0x401F8200U, 0x2U, 0x401F8530U, 0x0U, 0x401F83F0U
- #define **IOMUXC_GPIO_SD_B1_11_LPI2C2_SCL** 0x401F8200U, 0x3U, 0x401F84D4U, 0x0U, 0x401F83F0U
- #define **IOMUXC_GPIO_SD_B1_11_LPSPI2_PCS3** 0x401F8200U, 0x4U, 0, 0, 0x401F83F0U
- #define **IOMUXC_GPIO_SD_B1_11_GPIO3_IO11** 0x401F8200U, 0x5U, 0, 0, 0x401F83F0U
- #define **IOMUXC_GPR_SAIMCLK_LOWBITMASK** (0x7U)
- #define **IOMUXC_GPR_SAIMCLK_HIGHBITMASK** (0x3U)

## Configuration

- static void IOMUXC_SetPinMux (uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t inputOnfield)
  *Sets the IOMUXC pin mux mode.*
- static void IOMUXC_SetPinConfig (uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t configValue)
  *Sets the IOMUXC pin configuration.*
- static void IOMUXC_EnableMode (IOMUXC_GPR_Type ∗base, uint32_t mode, bool enable)
  *Sets IOMUXC general configuration for some mode.*
- static void IOMUXC_SetSaiMClkClockSource (IOMUXC_GPR_Type ∗base, iomuxc_gpr_saimclk_t mclk, uint8_t clkSrc)
  *Sets IOMUXC general configuration for SAI MCLK selection.*
- static void IOMUXC_MQSEnterSoftwareReset (IOMUXC_GPR_Type ∗base, bool enable)
  *Enters or exit MQS software reset.*
- static void IOMUXC_MQSEnable (IOMUXC_GPR_Type ∗base, bool enable)
  *Enables or disables MQS.*
- static void IOMUXC_MQSConfig (IOMUXC_GPR_Type ∗base, iomuxc_mqs_pwm_oversample_rate_t rate, uint8_t divider)
  *Configure MQS PWM oversampling rate compared with mclk and divider ratio control for mclk from hmclk.*

## 50.2 Macro Definition Documentation

### 50.2.1 #define FSL_IOMUXC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

## 50.3 Function Documentation

### 50.3.1 static void IOMUXC_SetPinMux ( uint32_t *muxRegister*, uint32_t *muxMode*, uint32_t *inputRegister*, uint32_t *inputDaisy*, uint32_t *configRegister*, uint32_t *inputOnfield* ) [inline], [static]

Note

The first five parameters can be filled with the pin function ID macros.

This is an example to set the PTA6 as the lpuart0_tx:

```
* IOMUXC_SetPinMux(IOMUXC_PTA6_LPUART0_TX, 0);
*
```

This is an example to set the PTA0 as GPIOA0:

```
* IOMUXC_SetPinMux(IOMUXC_PTA0_GPIOA0, 0);
*
```

Parameters

| | |
|---|---|
| *muxRegister* | The pin mux register. |
| *muxMode* | The pin mux mode. |
| *inputRegister* | The select input register. |
| *inputDaisy* | The input daisy. |
| *configRegister* | The config register. |
| *inputOnfield* | Software input on field. |

### 50.3.2 static void IOMUXC_SetPinConfig ( uint32_t *muxRegister,* uint32_t *muxMode,* uint32_t *inputRegister,* uint32_t *inputDaisy,* uint32_t *configRegister,* uint32_t *configValue* ) [inline], [static]

Note

The previous five parameters can be filled with the pin function ID macros.

This is an example to set pin configuration for IOMUXC_PTA3_LPI2C0_SCLS:

```
* IOMUXC_SetPinConfig(IOMUXC_PTA3_LPI2C0_SCLS,IOMUXC_SW_PAD_CTL_PAD_PUS_MASK|
      IOMUXC_SW_PAD_CTL_PAD_PUS(2U))
*
```

Parameters

| | |
|---|---|
| *muxRegister* | The pin mux register. |
| *muxMode* | The pin mux mode. |
| *inputRegister* | The select input register. |
| *inputDaisy* | The input daisy. |
| *configRegister* | The config register. |
| *configValue* | The pin config value. |

### 50.3.3 static void IOMUXC_EnableMode ( IOMUXC_GPR_Type * *base,* uint32_t *mode,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The IOMUXC GPR base address. |
| *mode* | The mode for setting. the mode is the logical OR of "iomuxc_gpr_mode" |
| *enable* | True enable false disable. |

### 50.3.4 static void IOMUXC_SetSaiMClkClockSource ( IOMUXC_GPR_Type * *base,* iomuxc_gpr_saimclk_t *mclk,* uint8_t *clkSrc* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The IOMUXC GPR base address. |
| *mclk* | The SAI MCLK. |
| *clkSrc* | The clock source. Take refer to register setting details for the clock source in RM. |

### 50.3.5 static void IOMUXC_MQSEnterSoftwareReset ( IOMUXC_GPR_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The IOMUXC GPR base address. |
| *enable* | Enter or exit MQS software reset. |

### 50.3.6 static void IOMUXC_MQSEnable ( IOMUXC_GPR_Type * *base,* bool *enable* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The IOMUXC GPR base address. |
| *enable* | Enable or disable the MQS. |

### 50.3.7 static void IOMUXC_MQSConfig ( IOMUXC_GPR_Type * *base,* iomuxc_mqs_pwm_oversample_rate_t *rate,* uint8_t *divider* ) [inline], [static]

Parameters

| | |
|---|---|
| *base* | The IOMUXC GPR base address. |
| *rate* | The MQS PWM oversampling rate, refer to "iomuxc_mqs_pwm_oversample_rate_t". |
| *divider* | The divider ratio control for mclk from hmclk. mclk freq = 1 /(divider + 1) * hmclk freq. |

# Chapter 51
# DMA Manager

## 51.1 Overview

DMA Manager provides a series of functions to manage the DMAMUX instances and channels.

## 51.2 Function groups

### 51.2.1 DMAMGR Initialization and De-initialization

This function group initializes and deinitializes the DMA Manager.

### 51.2.2 DMAMGR Operation

This function group requests/releases the DMAMUX channel and configures the channel request source.

## 51.3 Typical use case

### 51.3.1 DMAMGR static channel allocattion

```
uint8_t channel;
dmamanager_handle_t dmamanager_handle;

/* Initialize DMAMGR */
DMAMGR_Init(&dmamanager_handle, EXAMPLE_DMA_BASEADDR, DMA_CHANNEL_NUMBER, startChannel);
/* Request a DMAMUX channel by static allocate mechanism */
channel = kDMAMGR_STATIC_ALLOCATE;
DMAMGR_RequestChannel(&dmamanager_handle, kDmaRequestMux0AlwaysOn63, channel, &handle)
      ;
```

### 51.3.2 DMAMGR dynamic channel allocation

```
uint8_t channel;
dmamanager_handle_t dmamanager_handle;

/* Initialize DMAMGR */
DMAMGR_Init(&dmamanager_handle, EXAMPLE_DMA_BASEADDR, DMA_CHANNEL_NUMBER, startChannel);
/* Request a DMAMUX channel by Dynamic allocate mechanism */
channel = DMAMGR_DYNAMIC_ALLOCATE;
DMAMGR_RequestChannel(&dmamanager_handle, kDmaRequestMux0AlwaysOn63, channel, &handle)
      ;
```

## Data Structures

- struct dmamanager_handle_t
  *dmamanager handle typedef. More...*

### Macros

- #define DMAMGR_DYNAMIC_ALLOCATE 0xFFU
  *Dynamic channel allocation mechanism.*

### Enumerations

- enum _dma_manager_status {
  kStatus_DMAMGR_ChannelOccupied = MAKE_STATUS(kStatusGroup_DMAMGR, 0),
  kStatus_DMAMGR_ChannelNotUsed = MAKE_STATUS(kStatusGroup_DMAMGR, 1),
  kStatus_DMAMGR_NoFreeChannel = MAKE_STATUS(kStatusGroup_DMAMGR, 2) }
  *DMA manager status.*

### DMAMGR Initialization and De-initialization

- void DMAMGR_Init (dmamanager_handle_t *dmamanager_handle, DMA_Type *dma_base, uint32_t channelNum, uint32_t startChannel)
  *Initializes the DMA manager.*
- void DMAMGR_Deinit (dmamanager_handle_t *dmamanager_handle)
  *Deinitializes the DMA manager.*

### DMAMGR Operation

- status_t DMAMGR_RequestChannel (dmamanager_handle_t *dmamanager_handle, uint32_t requestSource, uint32_t channel, void *handle)
  *Requests a DMA channel.*
- status_t DMAMGR_ReleaseChannel (dmamanager_handle_t *dmamanager_handle, void *handle)
  *Releases a DMA channel.*
- bool DMAMGR_IsChannelOccupied (dmamanager_handle_t *dmamanager_handle, uint32_t channel)
  *Get a DMA channel status.*

## 51.4 Data Structure Documentation

### 51.4.1 struct dmamanager_handle_t

Note

The contents of this structure are private and subject to change.

This dma manager handle structure is used to store the parameters transfered by users.And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.

### Data Fields

- void * dma_base
  *Peripheral DMA instance.*
- uint32_t channelNum

*Channel numbers for the DMA instance which need to be managed by dma manager.*

- uint32_t startChannel
  *The start channel that can be managed by dma manager,users need to transfer it with a certain number or NULL.*
- bool s_DMAMGR_Channels [64]
  *The s_DMAMGR_Channels is used to store dma manager state.*
- uint32_t DmamuxInstanceStart
  *The DmamuxInstance is used to calculate the DMAMUX Instance according to the DMA Instance.*
- uint32_t multiple
  *The multiple is used to calculate the multiple between DMAMUX count and DMA count.*

#### 51.4.1.0.0.72 Field Documentation

##### 51.4.1.0.0.72.1 void* dmamanager_handle_t::dma_base

##### 51.4.1.0.0.72.2 uint32_t dmamanager_handle_t::channelNum

##### 51.4.1.0.0.72.3 uint32_t dmamanager_handle_t::startChannel

##### 51.4.1.0.0.72.4 bool dmamanager_handle_t::s_DMAMGR_Channels[64]

##### 51.4.1.0.0.72.5 uint32_t dmamanager_handle_t::DmamuxInstanceStart

##### 51.4.1.0.0.72.6 uint32_t dmamanager_handle_t::multiple

## 51.5 Macro Definition Documentation

### 51.5.1 #define DMAMGR_DYNAMIC_ALLOCATE 0xFFU

## 51.6 Enumeration Type Documentation

### 51.6.1 enum _dma_manager_status

Enumerator

***kStatus_DMAMGR_ChannelOccupied*** Channel has been occupied.
***kStatus_DMAMGR_ChannelNotUsed*** Channel has not been used.
***kStatus_DMAMGR_NoFreeChannel*** All channels have been occupied.

## 51.7 Function Documentation

### 51.7.1 void DMAMGR_Init ( dmamanager_handle_t * *dmamanager_handle*, DMA_Type * *dma_base*, uint32_t *channelNum*, uint32_t *startChannel* )

This function initializes the DMA manager, ungates the DMAMUX clocks, and initializes the eDMA or DMA peripherals.

Parameters

| | |
|---|---|
| *dmamanager_handle* | DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory. |
| *dma_base* | Peripheral DMA instance base pointer. |
| *dmamux_base* | Peripheral DMAMUX instance base pointer. |
| *channelNum* | Channel numbers for the DMA instance which need to be managed by dma manager. |
| *startChannel* | The start channel that can be managed by dma manager. |

### 51.7.2 void DMAMGR_Deinit ( dmamanager_handle_t * *dmamanager_handle* )

This function deinitializes the DMA manager, disables the DMAMUX channels, gates the DMAMUX clocks, and deinitializes the eDMA or DMA peripherals.

Parameters

| | |
|---|---|
| *dmamanager_handle* | DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory. |

### 51.7.3 status_t DMAMGR_RequestChannel ( dmamanager_handle_t * *dmamanager_handle,* uint32_t *requestSource,* uint32_t *channel,* void * *handle* )

This function requests a DMA channel which is not occupied. The two channels to allocate the mechanism are dynamic and static channels. For the dynamic allocation mechanism (channe = DMAMGR_DYNAMIC_ALLOCATE), DMAMGR allocates a DMA channel according to the given request source and startChannel and then configures it. For static allocation mechanism, DMAMGR configures the given channel according to the given request source and channel number.

Parameters

| | |
|---|---|
| *dmamanager_handle* | DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory. |
| *requestSource* | DMA channel request source number. See the soc.h, see the enum dma_request_source_t |
| *channel* | The channel number users want to occupy. If using the dynamic channel allocate mechanism, set the channel equal to DMAMGR_DYNAMIC_ALLOCATE. |
| *handle* | DMA or eDMA handle pointer. |

Return values

| | |
|---|---|
| *kStatus_Success* | In a dynamic/static channel allocation mechanism, allocate the DMAMUX channel successfully. |
| *kStatus_DMAMGR_NoFreeChannel* | In a dynamic channel allocation mechanism, all DMAMUX channels are occupied. |
| *kStatus_DMAMGR_ChannelOccupied* | In a static channel allocation mechanism, the given channel is occupied. |

### 51.7.4 status_t DMAMGR_ReleaseChannel ( dmamanager_handle_t * *dmamanager_handle,* void * *handle* )

This function releases an occupied DMA channel.

Parameters

| | |
|---|---|
| *dmamanager_handle* | DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory. |
| *handle* | DMA or eDMA handle pointer. |

Return values

| | |
|---|---|
| *kStatus_Success* | Releases the given channel successfully. |
| *kStatus_DMAMGR_-ChannelNotUsed* | The given channel to be released had not been used before. |

### 51.7.5 bool DMAMGR_IsChannelOccupied ( dmamanager_handle_t * *dmamanager_handle,* uint32_t *channel* )

This function get a DMA channel status. Return 0 indicates the channel has not been used, return 1 indicates the channel has been occupied.

Parameters

| | |
|---|---|
| *dmamanager_-handle* | DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory. |
| *channel* | The channel number that users want get its status. |

# Chapter 52
# Debug Console

## 52.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. The below picture shows the laylout of debug console.

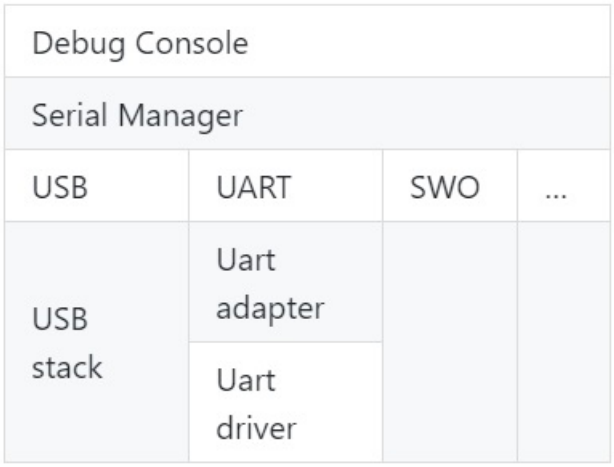


Figure 52.1.1: Debug console overview

## 52.2 Function groups

### 52.2.1 Initialization

To initialize the debug console, call the DbgConsole_Init() function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate,
      serial_port_type_t device, uint32_t clkSrcFreq);
```

Select the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_Uart = 1U,
```

```
    kSerialPort_UsbCdc,
    kSerialPort_Swo,
    kSerialPort_UsbCdcVirtual,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral.

This example shows how to call the DbgConsole_Init() given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,
                            BOARD_DEBUG_UART_CLK_FREQ);
```

### 52.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

| flags | Description |
|---|---|
| - | Left-justified within the given field width. Right-justified is the default. |
| + | Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign. |
| (space) | If no sign is written, a blank space is inserted before the value. |
| # | Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed. |
| 0 | Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier). |

| Width | Description |
|---|---|
| (number) | A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger. |
| * | The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted. |

| .precision | Description |
|---|---|
| .number | For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed. |
| .* | The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted. |

| length | Description |
|---|---|
| Do not support | |

| specifier | Description |
|---|---|
| d or i | Signed decimal integer |
| f | Decimal floating point |
| F | Decimal floating point capital letters |
| x | Unsigned hexadecimal integer |

| specifier | Description |
|---|---|
| X | Unsigned hexadecimal integer capital letters |
| o | Signed octal |
| b | Binary value |
| p | Pointer address |
| u | Unsigned decimal integer |
| c | Character |
| s | String of characters |
| n | Nothing printed |

- Support a format specifier for SCANF following this prototype " %[∗][width][length]specifier", which is explained below

| ∗ | Description |
|---|---|
| An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument. | |

| width | Description |
|---|---|
| This specifies the maximum number of characters to be read in the current reading operation. | |

| length | Description |
|---|---|
| hh | The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X). |
| h | The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X). |
| l | The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s. |
| ll | The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s. |
| L | The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G). |

| length | Description |
|---|---|
| j or z or t | Not supported |

| specifier | Qualifying Input | Type of argument |
|---|---|---|
| c | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char * |
| i | Integer: : Number optionally preceded with a + or - sign | int * |
| d | Decimal integer: Number optionally preceded with a + or - sign | int * |
| a, A, e, E, f, F, g, G | Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4 | float * |
| o | Octal Integer: | int * |
| s | String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab). | char * |
| u | Unsigned decimal integer. | unsigned int * |

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE    /* Select printf, scanf, putchar, getchar of SDK version. */
```

```
#define PRINTF                  DbgConsole_Printf
#define SCANF                   DbgConsole_Scanf
#define PUTCHAR                 DbgConsole_Putchar
#define GETCHAR                 DbgConsole_Getchar
#else                   /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF                  printf
#define SCANF                   scanf
#define PUTCHAR                 putchar
#define GETCHAR                 getchar
#endif /* SDK_DEBUGCONSOLE */
```

## 52.3 Typical use case

### Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

### Print out failure messages using MCUXpresso SDK __assert_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
       , line, func);
    for (;;)
    {}
}
```

### Note:

To use 'printf' and 'scanf' for GNUC Base, add file **'fsl_sbrk.c'** in path: **..\{package}\devices\{subset}\utilities\fsl_sbrk.c** to your project.

## Modules

- SWO
- Semihosting

## Macros

- #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U
  *Definition select redirect toolchain printf, scanf to uart or not.*
- #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U
  *Select SDK version printf, scanf.*
- #define DEBUGCONSOLE_DISABLE 2U
  *Disable debugconsole function.*
- #define SDK_DEBUGCONSOLE 1U
  *Definition to select sdk or toolchain printf, scanf.*
- #define SDK_DEBUGCONSOLE_UART
  *Definition to select redirect toolchain printf, scanf to uart or not.*
- #define PRINTF DbgConsole_Printf
  *Definition to select redirect toolchain printf, scanf to uart or not.*

## Typedefs

- typedef void(* printfCb )(char *buf, int32_t *indicator, char val, int len)
  *A function pointer which is used when format printf log.*

## Functions

- int StrFormatPrintf (const char *fmt, va_list ap, char *buf, printfCb cb)
  *This function outputs its parameters according to a formatted string.*
- int StrFormatScanf (const char *line_ptr, char *format, va_list args_ptr)
  *Converts an input line of ASCII characters based upon a provided string format.*

## Variables

- serial_handle_t g_serialHandle
  *serial manager handle*

## Initialization

- status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)
  *Initializes the peripheral used for debug messages.*
- status_t DbgConsole_Deinit (void)
  *De-initializes the peripheral used for debug messages.*

- int DbgConsole_Printf (const char *formatString,...)
  *Writes formatted output to the standard output stream.*
- int DbgConsole_Putchar (int ch)
  *Writes a character to stdout.*
- int DbgConsole_Scanf (char *formatString,...)
  *Reads formatted data from the standard input stream.*
- int DbgConsole_Getchar (void)
  *Reads a character from standard input.*
- status_t DbgConsole_Flush (void)
  *Debug console flush.*

## 52.4 Macro Definition Documentation

### 52.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

### 52.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

### 52.4.3 #define DEBUGCONSOLE_DISABLE 2U

### 52.4.4 #define SDK_DEBUGCONSOLE 1U

The macro only support to be redefined in project setting.

### 52.4.5 #define SDK_DEBUGCONSOLE_UART

### 52.4.6 #define PRINTF DbgConsole_Printf

if SDK_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

## 52.5 Function Documentation

### 52.5.1 status_t DbgConsole_Init ( uint8_t *instance*, uint32_t *baudRate*, serial_port_type_t *device*, uint32_t *clkSrcFreq* )

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

| | |
|---|---|
| *instance* | The instance of the module. |
| *baudRate* | The desired baud rate in bits per second. |
| *device* | Low level device type for the debug console, can be one of the following.<br>• kSerialPort_Uart,<br>• kSerialPort_UsbCdc<br>• kSerialPort_UsbCdcVirtual. |
| *clkSrcFreq* | Frequency of peripheral source clock. |

Returns

Indicates whether initialization was successful or not.

Return values

| | |
|---|---|
| *kStatus_Success* | Execution successfully |

### 52.5.2 status_t DbgConsole_Deinit ( void )

Call this function to disable debug log messages to be output via the specified peripheral initialized by the serial manager module.

Returns

Indicates whether de-initialization was successful or not.

### 52.5.3 int DbgConsole_Printf ( const char * *formatString,* ... )

Call this function to write a formatted output to the standard output stream.

Parameters

| | |
|---|---|
| *formatString* | Format control string. |

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 52.5.4 int DbgConsole_Putchar ( int *ch* )

Call this function to write a character to stdout.

Parameters

| | |
|---|---|
| *ch* | Character to be written. |

Returns

Returns the character written.

### 52.5.5 int DbgConsole_Scanf ( char * *formatString*, ... )

Call this function to read formatted data from the standard input stream.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Parameters

| | |
|---|---|
| *formatString* | Format control string. |

Returns

Returns the number of fields successfully converted and assigned.

### 52.5.6 int DbgConsole_Getchar ( void )

Call this function to read a character from standard input.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Returns

Returns the character read.

### 52.5.7 status_t DbgConsole_Flush ( void )

Call this function to wait the tx buffer empty. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

Returns

Indicates whether wait idle was successful or not.

### 52.5.8 int StrFormatPrintf ( const char * *fmt,* va_list *ap,* char * *buf,* printfCb *cb* )

Note

I/O is performed by calling given function pointer using following (∗func_ptr)(c);

Parameters

| | | |
|---|---|---|
| in | *fmt* | Format string for printf. |
| in | *ap* | Arguments to printf. |
| in | *buf* | pointer to the buffer |
| | *cb* | print callbck function pointer |

Returns

Number of characters to be print

### 52.5.9 int StrFormatScanf ( const char * *line_ptr,* char * *format,* va_list *args_ptr* )

Parameters

| | | |
|---|---|---|
| in | *line_ptr* | The input line of ASCII data. |
| in | *format* | Format first points to the format string. |
| in | *args_ptr* | The list of parameters. |

Returns

Number of input items converted and assigned.

Return values

| | |
|---|---|
| *IO_EOF* | When line_ptr is empty string "". |

## 52.6 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as printf() and scanf(), to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

### 52.6.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the SDK_DEBUGCONSOLE is disabled.

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

#### Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

#### Step 3: Starting semihosting

1. Choose "Semihosting_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting.

1. Make sure the SDK_DEBUGCONSOLE_UART is not defined, remove the default definition in fsl_debug_console.h.

1. Start the project by choosing Project>Download and Debug.
2. Choose View>Terminal I/O to display the output from the I/O operations.

### 52.6.2 Guide Semihosting for Keil µVision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

### 52.6.3 Guide Semihosting for MCUXpresso IDE

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK_DEBUGCONSOLE=0, if set SDK_DEBUGCONSOLE=1, the log will be redirect to the UART.

#### Step 2: Building the project

1. Compile and link the project.

#### Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

### 52.6.4 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
   - "Host Name (or IP address)" : localhost
   - "Port" :2333
   - "Connection type" : Telet.
   - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

**Add to "CMakeLists.txt"**

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__heap_size__=0x2000")

### Step 2: Building the project

1. Change "CMakeLists.txt":
   **Change** "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=nano.specs")"
   **to** "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=rdimon.specs")"
   **Replace paragraph**
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
   **To**
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
   **Remove**
   target_link_libraries(semihosting_ARMGCC.elf debug nosys)
2. Run "build_debug.bat" to build project

### Step 3: Starting semihosting

(a) Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

(b) After the setting, press "enter". The PuTTY window now shows the printf() output.

## 52.7 SWO

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDE also support SWO, such IAR and KEIL, both input and output are supported, reference below for detail.

### 52.7.1 Guide SWO for SDK

**NOTE:** After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

#### Step 1: Setting up the environment

1. Define SERIAL_PORT_TYPE_SWO in your project settings.
2. Prepare code, the port and baudrate can be decided by application, clkSrcFreq should be mcu core clock frequency:

```
DbgConsole_Init(instance, baudRate, SERIAL_PORT_TYPE_SWO, clkSrcFreq);
```

3. Use PRINTF or printf to print some thing in application.

#### Step 2: Building the project

#### Step 3: Download and run project

#### 52.7.1.1 Guide SWO for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK,and click the Activate button to enable trace data collection.
6. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log,The SDK_DEBUGCONSOLE_UART defined or not defined will

not effect debug function. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero,then debug function ok. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one,then debug function ok.

**NOTE:** Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h. For case a and c, Do and not do the above third step will be not affect function.

1. Start the project by choosing Project>Download and Debug.

#### Step 2: Building the project

#### Step 3: Starting swo

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.
3. Run application.

### 52.7.2 Guide SWO for Keil µVision

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log,the SDK_DEBUGCONSOLE_UART definition does not affect the functionality and skip the second step directly. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero,then start the second step. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one,then skip the second step directly.

**NOTE:** Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h.

1. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
2. Open Project>Options for target or using Alt+F7 or click.
3. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
4. Select “Debug” tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click OK, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

#### Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

#### Step 4: Run the project

1. Choose “Debug” on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

### 52.7.3 Guide SWO for MCUXpresso IDE

**NOTE:** MCUX support SWO for LPC-Link2 debug probe only.

### 52.7.4 Guide SWO for ARMGCC

**NOTE:** ARMGCC has no library support SWO.

# Chapter 53
# Notification Framework

## 53.1 Overview

This section describes the programming interface of the Notifier driver.

## 53.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.
   The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{

    status_t ret = kStatus_Success;

    ...
    ...
    ...

    return ret;
}
// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *userData)
{
```

```
    ...
    ...
    ...
}
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
                kNOTIFIER_CallbackBeforeAfter,
                (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
      APP_PowerModeSwitch, NULL);
    ...
    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
      kNOTIFIER_PolicyAgreement);
}
```

## Data Structures

- struct notifier_notification_block_t
  *notification block passed to the registered callback function. More...*
- struct notifier_callback_config_t
  *Callback configuration structure. More...*
- struct notifier_handle_t
  *Notifier handle structure. More...*

## Typedefs

- typedef void notifier_user_config_t
  *Notifier user configuration type.*
- typedef status_t(* notifier_user_function_t )(notifier_user_config_t *targetConfig, void *userData)
  *Notifier user function prototype Use this function to execute specific operations in configuration switch.*

- typedef status_t(∗ notifier_callback_t )(notifier_notification_block_t ∗notify, void ∗data)
  *Callback prototype.*

## Enumerations

- enum _notifier_status {
  kStatus_NOTIFIER_ErrorNotificationBefore,
  kStatus_NOTIFIER_ErrorNotificationAfter }
  *Notifier error codes.*
- enum notifier_policy_t {
  kNOTIFIER_PolicyAgreement,
  kNOTIFIER_PolicyForcible }
  *Notifier policies.*
- enum notifier_notification_type_t {
  kNOTIFIER_NotifyRecover = 0x00U,
  kNOTIFIER_NotifyBefore = 0x01U,
  kNOTIFIER_NotifyAfter = 0x02U }
  *Notification type.*
- enum notifier_callback_type_t {
  kNOTIFIER_CallbackBefore = 0x01U,
  kNOTIFIER_CallbackAfter = 0x02U,
  kNOTIFIER_CallbackBeforeAfter = 0x03U }
  *The callback type, which indicates kinds of notification the callback handles.*

## Functions

- status_t NOTIFIER_CreateHandle (notifier_handle_t ∗notifierHandle, notifier_user_config_t ∗∗configs, uint8_t configsNumber, notifier_callback_config_t ∗callbacks, uint8_t callbacksNumber, notifier_user_function_t userFunction, void ∗userData)
  *Creates a Notifier handle.*
- status_t NOTIFIER_SwitchConfig (notifier_handle_t ∗notifierHandle, uint8_t configIndex, notifier_policy_t policy)
  *Switches the configuration according to a pre-defined structure.*
- uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t ∗notifierHandle)
  *This function returns the last failed notification callback.*

# 53.3 Data Structure Documentation

## 53.3.1 struct notifier_notification_block_t

### Data Fields

- notifier_user_config_t ∗ targetConfig
  *Pointer to target configuration.*
- notifier_policy_t policy
  *Configure transition policy.*
- notifier_notification_type_t notifyType
  *Configure notification type.*

#### 53.3.1.0.0.73 Field Documentation

#### 53.3.1.0.0.73.1 notifier_user_config_t* notifier_notification_block_t::targetConfig

#### 53.3.1.0.0.73.2 notifier_policy_t notifier_notification_block_t::policy

#### 53.3.1.0.0.73.3 notifier_notification_type_t notifier_notification_block_t::notifyType

### 53.3.2 struct notifier_callback_config_t

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback.

## Data Fields

- notifier_callback_t callback
  *Pointer to the callback function.*
- notifier_callback_type_t callbackType
  *Callback type.*
- void * callbackData
  *Pointer to the data passed to the callback.*

#### 53.3.2.0.0.74 Field Documentation

#### 53.3.2.0.0.74.1 notifier_callback_t notifier_callback_config_t::callback

#### 53.3.2.0.0.74.2 notifier_callback_type_t notifier_callback_config_t::callbackType

#### 53.3.2.0.0.74.3 void* notifier_callback_config_t::callbackData

### 53.3.3 struct notifier_handle_t

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. NOTIFIER_CreateHandle() must be called to initialize this handle.

## Data Fields

- notifier_user_config_t ** configsTable
  *Pointer to configure table.*
- uint8_t configsNumber
  *Number of configurations.*
- notifier_callback_config_t * callbacksTable
  *Pointer to callback table.*

- uint8_t callbacksNumber
  *Maximum number of callback configurations.*
- uint8_t errorCallbackIndex
  *Index of callback returns error.*
- uint8_t currentConfigIndex
  *Index of current configuration.*
- notifier_user_function_t userFunction
  *User function.*
- void * userData
  *User data passed to user function.*

#### 53.3.3.0.0.75 Field Documentation

##### 53.3.3.0.0.75.1 notifier_user_config_t** notifier_handle_t::configsTable

##### 53.3.3.0.0.75.2 uint8_t notifier_handle_t::configsNumber

##### 53.3.3.0.0.75.3 notifier_callback_config_t* notifier_handle_t::callbacksTable

##### 53.3.3.0.0.75.4 uint8_t notifier_handle_t::callbacksNumber

##### 53.3.3.0.0.75.5 uint8_t notifier_handle_t::errorCallbackIndex

##### 53.3.3.0.0.75.6 uint8_t notifier_handle_t::currentConfigIndex

##### 53.3.3.0.0.75.7 notifier_user_function_t notifier_handle_t::userFunction

##### 53.3.3.0.0.75.8 void* notifier_handle_t::userData

## 53.4 Typedef Documentation

### 53.4.1 typedef void notifier_user_config_t

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

### 53.4.2 typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, NOTIFIER_SwitchConfig() exits.

Parameters

| | |
|---|---|
| *targetConfig* | target Configuration. |
| *userData* | Refers to other specific data passed to user function. |

Returns

An error code or kStatus_Success.

### 53.4.3 typedef status_t(∗ notifier_callback_t)(notifier_notification_block_t ∗notify, void ∗data)

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the notifier_callback_config_t callback configuration structure. Depending on callback type, function of this prototype is called (see NOTIFIER_SwitchConfig()) before configuration switch, after it or in both use cases to notify about the switch progress (see notifier_callback_type_t). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see notifier_notification_block_t) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see notifier_policy_t), the callback may deny the execution of the user function by returning an error code different than kStatus_Success (see NOTIFIER_SwitchConfig()).

Parameters

| | |
|---|---|
| *notify* | Notification block. |
| *data* | Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information. |

Returns

An error code or kStatus_Success.

## 53.5 Enumeration Type Documentation

### 53.5.1 enum _notifier_status

Used as return value of Notifier functions.

Enumerator

***kStatus_NOTIFIER_ErrorNotificationBefore*** An error occurs during send "BEFORE" notification.

***kStatus_NOTIFIER_ErrorNotificationAfter*** An error occurs during send "AFTER" notification.

### 53.5.2 enum notifier_policy_t

Defines whether the user function execution is forced or not. For kNOTIFIER_PolicyForcible, the user function is executed regardless of the callback results, while kNOTIFIER_PolicyAgreement policy is used to exit NOTIFIER_SwitchConfig() when any of the callbacks returns error code. See also NOTIFIER_SwitchConfig() description.

Enumerator

***kNOTIFIER_PolicyAgreement*** NOTIFIER_SwitchConfig() method is exited when any of the callbacks returns error code.
***kNOTIFIER_PolicyForcible*** The user function is executed regardless of the results.

### 53.5.3 enum notifier_notification_type_t

Used to notify registered callbacks

Enumerator

***kNOTIFIER_NotifyRecover*** Notify IP to recover to previous work state.
***kNOTIFIER_NotifyBefore*** Notify IP that configuration setting is going to change.
***kNOTIFIER_NotifyAfter*** Notify IP that configuration setting has been changed.

### 53.5.4 enum notifier_callback_type_t

Used in the callback configuration structure (notifier_callback_config_t) to specify when the registered callback is called during configuration switch initiated by the NOTIFIER_SwitchConfig(). Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect NOTIFIER_SwitchConfig() execution. See the NOTIFIER_SwitchConfig() and notifier_policy_t documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

***kNOTIFIER_CallbackBefore*** Callback handles BEFORE notification.
***kNOTIFIER_CallbackAfter*** Callback handles AFTER notification.
***kNOTIFIER_CallbackBeforeAfter*** Callback handles BEFORE and AFTER notification.

## 53.6 Function Documentation

### 53.6.1 status_t NOTIFIER_CreateHandle ( notifier_handle_t * *notifierHandle,* notifier_user_config_t ** *configs,* uint8_t *configsNumber,* notifier_callback-_config_t * *callbacks,* uint8_t *callbacksNumber,* notifier_user_function_t *userFunction,* void * *userData* )

Parameters

| | |
|---|---|
| *notifierHandle* | A pointer to the notifier handle. |
| *configs* | A pointer to an array with references to all configurations which is handled by the Notifier. |
| *configsNumber* | Number of configurations. Size of the configuration array. |
| *callbacks* | A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value. |
| *callbacks-Number* | Number of registered callbacks. Size of the callbacks array. |
| *userFunction* | User function. |
| *userData* | User data passed to user function. |

Returns

An error Code or kStatus_Success.

### 53.6.2 status_t NOTIFIER_SwitchConfig ( notifier_handle_t * *notifierHandle,* uint8_t *configIndex,* notifier_policy_t *policy* )

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER_SwitchConfig() exits.

Parameters

| | |
|---|---|
| *notifierHandle* | pointer to notifier handle |
| *configIndex* | Index of the target configuration. |
| *policy* | Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible. |

Returns

An error code or kStatus_Success.

### 53.6.3 uint8_t NOTIFIER_GetErrorCallbackIndex ( notifier_handle_t * *notifierHandle* )

This function returns an index of the last callback that failed during the configuration switch while the last NOTIFIER_SwitchConfig() was called. If the last NOTIFIER_SwitchConfig() call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

| | |
|---|---|
| *notifierHandle* | Pointer to the notifier handle |

Returns

Callback Index of the last failed callback or value equal to callbacks count.

# Chapter 54
# Shell

## 54.1 Overview

This part describes the programming interface of the Shell middleware. Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

## 54.2 Function groups

### 54.2.1 Initialization

To initialize the Shell middleware, call the SHELL_Init() function with these parameters. This function automatically enables the middleware.

```
void SHELL_Init(p_shell_context_t context, send_data_cb_t send_cb, recv_data_cb_t recv_cb, char *
      prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

This example shows how to call the SHELL_Init() given the user configuration structure.

```
SHELL_Init(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
```

### 54.2.2 Advanced Feature

- Support to get a character from standard input devices.

  ```
  static uint8_t GetChar(p_shell_context_t context);
  ```

| Commands | Description |
|---|---|
| Help | Lists all commands which are supported by Shell. |
| Exit | Exits the Shell program. |
| strCompare | Compares the two input strings. |

| Input character | Description |
|---|---|
| A | Gets the latest command in the history. |
| B | Gets the first command in the history. |
| C | Replaces one character at the right of the pointer. |

**Function groups**

| Input character | Description |
|---|---|
| D | Replaces one character at the left of the pointer. |
| | Run AutoComplete function |
| | Run cmdProcess function |
| | Clears a command. |

### 54.2.3 Shell Operation

```
SHELL_Init(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
SHELL_Main(&user_context);
```

## Data Structures

- struct shell_command_t
  *User command data configuration structure. More...*

## Macros

- #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE
  *Whether use non-blocking mode.*
- #define SHELL_AUTO_COMPLETE (1U)
  *Macro to set on/off auto-complete feature.*
- #define SHELL_BUFFER_SIZE (64U)
  *Macro to set console buffer size.*
- #define SHELL_MAX_ARGS (8U)
  *Macro to set maximum arguments in command.*
- #define SHELL_HISTORY_COUNT (3U)
  *Macro to set maximum count of history commands.*
- #define SHELL_IGNORE_PARAMETER_COUNT (0xFF)
  *Macro to bypass arguments check.*
- #define SHELL_HANDLE_SIZE (520U)
  *The handle size of the shell module.*
- #define SHELL_COMMAND_DEFINE(command, descriptor, callback, paramCount)
  *Defines the shell command structure.*
- #define SHELL_COMMAND(command) &g_shellCommand##command
  *Gets the shell command pointer.*

## Typedefs

- typedef void * shell_handle_t
  *The handle of the shell module.*
- typedef shell_status_t(* cmd_function_t )(shell_handle_t shellHandle, int32_t argc, char **argv)
  *User command function prototype.*

### Enumerations

- enum shell_status_t {
  kStatus_SHELL_Success = kStatus_Success,
  kStatus_SHELL_Error = MAKE_STATUS(kStatusGroup_SHELL, 1),
  kStatus_SHELL_OpenWriteHandleFailed = MAKE_STATUS(kStatusGroup_SHELL, 2),
  kStatus_SHELL_OpenReadHandleFailed = MAKE_STATUS(kStatusGroup_SHELL, 3) }

### Shell functional operation

- shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char *prompt)
  *Initializes the shell module.*
- shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t *shell-Command)
  *Registers the shell command.*
- shell_status_t SHELL_UnregisterCommand (shell_command_t *shellCommand)
  *Unregisters the shell command.*
- shell_status_t SHELL_Write (shell_handle_t shellHandle, char *buffer, uint32_t length)
  *Sends data to the shell output stream.*
- int SHELL_Printf (shell_handle_t shellHandle, const char *formatString,...)
  *Writes formatted output to the shell output stream.*
- void SHELL_Task (shell_handle_t shellHandle)
  *The task function for Shell.*

## 54.3 Data Structure Documentation

### 54.3.1 struct shell_command_t

### Data Fields

- const char * pcCommand
  *The command that is executed.*
- char * pcHelpString
  *String that describes how to use the command.*
- const cmd_function_t pFuncCallBack
  *A pointer to the callback function that returns the output generated by the command.*
- uint8_t cExpectedNumberOfParameters
  *Commands expect a fixed number of parameters, which may be zero.*
- list_element_t link
  *link of the element*

#### 54.3.1.0.0.76 Field Documentation

#### 54.3.1.0.0.76.1 const char* shell_command_t::pcCommand

For example "help". It must be all lower case.

#### 54.3.1.0.0.76.2 char* shell_command_t::pcHelpString

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

#### 54.3.1.0.0.76.3 const cmd_function_t shell_command_t::pFuncCallBack

#### 54.3.1.0.0.76.4 uint8_t shell_command_t::cExpectedNumberOfParameters

## 54.4 Macro Definition Documentation

### 54.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

### 54.4.2 #define SHELL_AUTO_COMPLETE (1U)

### 54.4.3 #define SHELL_BUFFER_SIZE (64U)

### 54.4.4 #define SHELL_MAX_ARGS (8U)

### 54.4.5 #define SHELL_HISTORY_COUNT (3U)

### 54.4.6 #define SHELL_HANDLE_SIZE (520U)

It is the sum of the SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE

### 54.4.7 #define SHELL_COMMAND_DEFINE( *command, descriptor, callback, paramCount* )

**Value:**

```
\
    shell_command_t g_shellCommand##command = {                         \
        (#command), (descriptor), (callback), (paramCount), {0},        \
    }
```

This macro is used to define the shell command structure shell_command_t. And then uses the macro SHELL_COMMAND to get the command structure pointer. The macro should not be used in any function.

This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

| | |
|---|---|
| *command* | The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read". |
| *descriptor* | The description of the command is used for showing the command usage when "help" is typing. |
| *callback* | The callback of the command is used to handle the command line when the input command is matched. |
| *paramCount* | The max parameter count of the current command. |

### 54.4.8 #define SHELL_COMMAND( *command* ) &g_shellCommand##command

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Parameters

| | |
|---|---|
| *command* | The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read". |

## 54.5 Typedef Documentation

### 54.5.1 typedef shell_status_t(∗ cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char ∗∗argv)

## 54.6 Enumeration Type Documentation

### 54.6.1 enum shell_status_t

Enumerator

***kStatus_SHELL_Success*** Success.
***kStatus_SHELL_Error*** Failed.
***kStatus_SHELL_OpenWriteHandleFailed*** Open write handle failed.
***kStatus_SHELL_OpenReadHandleFailed*** Open read handle failed.

## 54.7 Function Documentation

### 54.7.1 shell_status_t SHELL_Init ( shell_handle_t *shellHandle,* serial_handle_t *serialHandle,* char ∗ *prompt* )

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL-_Init function by passing in these parameters. This is an example.

```
*   static uint8_t s_shellHandleBuffer[SHELL_HANDLE_SIZE];
*   static shell_handle_t s_shellHandle = &s_shellHandleBuffer[0];
*   SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
*
```

Parameters

| | |
|---|---|
| *shellHandle* | Pointer to point to a memory space of size SHELL_HANDLE_SIZE allocated by the caller. |
| *serialHandle* | The serial manager module handle pointer. |
| *prompt* | The string prompt pointer of Shell. Only the global variable can be passed. |

Return values

| | |
|---|---|
| *kStatus_SHELL_Success* | The shell initialization succeed. |
| *kStatus_SHELL_Error* | An error occurred when the shell is initialized. |
| *kStatus_SHELL_OpenWriteHandleFailed* | Open the write handle failed. |
| *kStatus_SHELL_OpenReadHandleFailed* | Open the read handle failed. |

### 54.7.2 shell_status_t SHELL_RegisterCommand ( shell_handle_t *shellHandle*, shell_command_t * *shellCommand* )

This function is used to register the shell command by using the command configuration #shell_command_config_t. This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

| | |
|---|---|
| *shellHandle* | The shell module handle pointer. |
| *command* | The command element. |

Return values

| | |
|---|---|
| *kStatus_SHELL_Success* | Successfully register the command. |
| *kStatus_SHELL_Error* | An error occurred. |

### 54.7.3 shell_status_t SHELL_UnregisterCommand ( shell_command_t * *shellCommand* )

This function is used to unregister the shell command.

Parameters

| | |
|---|---|
| *command* | The command element. |

Return values

| | |
|---|---|
| *kStatus_SHELL_Success* | Successfully unregister the command. |

### 54.7.4 shell_status_t SHELL_Write ( shell_handle_t *shellHandle,* char * *buffer,* uint32_t *length* )

This function is used to send data to the shell output stream.

Parameters

| | |
|---|---|
| *shellHandle* | The shell module handle pointer. |
| *buffer* | Start address of the data to write. |
| *length* | Length of the data to write. |

Return values

| | |
|---|---|
| *kStatus_SHELL_Success* | Successfully send data. |
| *kStatus_SHELL_Error* | An error occurred. |

### 54.7.5 int SHELL_Printf ( shell_handle_t *shellHandle,* const char * *formatString,* ... )

Call this function to write a formatted output to the shell output stream.

Parameters

| | |
|---|---|
| *shellHandle* | The shell module handle pointer. |
| *formatString* | Format string. |

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 54.7.6 void SHELL_Task ( shell_handle_t *shellHandle* )

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

| | |
|---|---|
| *shellHandle* | The shell module handle pointer. |

# Chapter 55
# Serial Manager

## 55.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are uart, usbcdc and swo.

## Modules

- Serial Port SWO
- Serial Port USB
- Serial Port Uart
- Serial Port Virtual USB

## Data Structures

- struct serial_manager_config_t
  *serial manager config structure More...*
- struct serial_manager_callback_message_t
  *Callback message structure. More...*

## Macros

- #define SERIAL_PORT_TYPE_UART (1U)
  *Enable or disable uart port (1 - enable, 0 - disable)*
- #define SERIAL_PORT_TYPE_USBCDC (0U)
  *Enable or disable USB CDC port (1 - enable, 0 - disable)*
- #define SERIAL_PORT_TYPE_SWO (0U)
  *Enable or disable SWO port (1 - enable, 0 - disable)*
- #define SERIAL_PORT_TYPE_USBCDC_VIRTUAL (0U)
  *Enable or disable USB CDC virtual port (1 - enable, 0 - disable)*
- #define SERIAL_MANAGER_WRITE_HANDLE_SIZE (4U)
  *Set serial manager write handle size.*
- #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)
  *SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.*

## Typedefs

- typedef void(∗ serial_manager_callback_t )(void ∗callbackParam, serial_manager_callback_message_t ∗message, serial_manager_status_t status)
  *callback function*

## Enumerations

- enum serial_port_type_t {
  kSerialPort_Uart = 1U,
  kSerialPort_UsbCdc,
  kSerialPort_Swo,
  kSerialPort_UsbCdcVirtual }
  *serial port type*
- enum serial_manager_status_t {
  kStatus_SerialManager_Success = kStatus_Success,
  kStatus_SerialManager_Error = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1),
  kStatus_SerialManager_Busy = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2),
  kStatus_SerialManager_Notify = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3),
  kStatus_SerialManager_Canceled,
  kStatus_SerialManager_HandleConflict = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5),
  kStatus_SerialManager_RingBufferOverflow }
  *serial manager error code*

## Functions

- serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, serial_manager_config_t *config)
  *Initializes a serial manager module with the serial manager handle and the user configuration structure.*
- serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)
  *De-initializes the serial manager module instance.*
- serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)
  *Opens a writing handle for the serial manager module.*
- serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)
  *Closes a writing handle for the serial manager module.*
- serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)
  *Opens a reading handle for the serial manager module.*
- serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)
  *Closes a reading for the serial manager module.*
- serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)
  *Transmits data with the blocking mode.*
- serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)
  *Reads data with the blocking mode.*
- serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)
  *Prepares to enter low power consumption.*
- serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)
  *Restores from low power consumption.*

## 55.2 Data Structure Documentation

### 55.2.1 struct serial_manager_config_t

#### Data Fields

- uint8_t * ringBuffer
  *Ring buffer address, it is used to buffer data received by the hardware.*
- uint32_t ringBufferSize
  *The size of the ring buffer.*
- serial_port_type_t type
  *Serial port type.*
- void * portConfig
  *Serial port configuration.*

##### 55.2.1.0.0.77 Field Documentation

###### 55.2.1.0.0.77.1 uint8_t* serial_manager_config_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

### 55.2.2 struct serial_manager_callback_message_t

#### Data Fields

- uint8_t * buffer
  *Transferred buffer.*
- uint32_t length
  *Transferred data length.*

## 55.3 Enumeration Type Documentation

### 55.3.1 enum serial_port_type_t

Enumerator

***kSerialPort_Uart*** Serial port UART.
***kSerialPort_UsbCdc*** Serial port USB CDC.
***kSerialPort_Swo*** Serial port SWO.
***kSerialPort_UsbCdcVirtual*** Serial port USB CDC Virtual.

### 55.3.2 enum serial_manager_status_t

Enumerator

***kStatus_SerialManager_Success*** Success.

***kStatus_SerialManager_Error*** Failed.
***kStatus_SerialManager_Busy*** Busy.
***kStatus_SerialManager_Notify*** Ring buffer is not empty.
***kStatus_SerialManager_Canceled*** the non-blocking request is canceled
***kStatus_SerialManager_HandleConflict*** The handle is opened.
***kStatus_SerialManager_RingBufferOverflow*** The ring buffer is overflowed.

## 55.4 Function Documentation

### 55.4.1 serial_manager_status_t SerialManager_Init ( serial_handle_t *serialHandle,* serial_manager_config_t * *config* )

This function configures the serial manager module with user-defined settings. The user can configure the configuration structure. The parameter serialHandle is a pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The serial manager module supports two types of serial port, uart (includes UART, USART, LPSCI, LPUART, etc) and USB CDC. Please refer to serial_port_type_t for serial port setting. These two types can be set by using serial_manager_config_t.

Example below shows how to use this API to configure the serial manager. For UART,

```
*   #define SERIAL_MANAGER_RING_BUFFER_SIZE          (256U)
*   static uint8_t s_serialHandleBuffer[SERIAL_MANAGER_HANDLE_SIZE];
*   static serial_handle_t s_serialHandle = &s_serialHandleBuffer[0];
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_uart_config_t uartConfig;
*   config.type = kSerialPort_Uart;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   uartConfig.instance = 0;
*   uartConfig.clockRate = 24000000;
*   uartConfig.baudRate = 115200;
*   uartConfig.parityMode = kSerialManager_UartParityDisabled;
*   uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
*   uartConfig.enableRx = 1;
*   uartConfig.enableTx = 1;
*   config.portConfig = &uartConfig;
*   SerialManager_Init(s_serialHandle, &config);
*
```

For USB CDC,

```
*   #define SERIAL_MANAGER_RING_BUFFER_SIZE          (256U)
*   static uint8_t s_serialHandleBuffer[SERIAL_MANAGER_HANDLE_SIZE];
*   static serial_handle_t s_serialHandle = &s_serialHandleBuffer[0];
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_usb_cdc_config_t usbCdcConfig;
*   config.type = kSerialPort_UsbCdc;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   usbCdcConfig.controllerIndex =
      kSerialManager_UsbControllerKhci0;
*   config.portConfig = &usbCdcConfig;
*   SerialManager_Init(s_serialHandle, &config);
*
```

Parameters

| | |
|---|---|
| *serialHandle* | Pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. |
| *config* | Pointer to user-defined configuration structure. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_Error* | An error occurred. |
| *kStatus_SerialManager_Success* | The serial manager module initialization succeed. |

### 55.4.2 serial_manager_status_t SerialManager_Deinit ( serial_handle_t *serialHandle* )

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return kStatus_SerialManager_Busy.

Parameters

| | |
|---|---|
| *serialHandle* | The serial manager module handle pointer. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_Success* | The serial manager de-initialization succeed. |
| *kStatus_SerialManager_Busy* | Opened reading or writing handle is not closed. |

### 55.4.3 serial_manager_status_t SerialManager_OpenWriteHandle ( serial_handle_t *serialHandle*, serial_write_handle_t *writeHandle* )

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling SerialManager_OpenWriteHandle. Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

**Function Documentation**

Parameters

| | |
|---|---|
| *serialHandle* | The serial manager module handle pointer. |
| *writeHandle* | The serial manager module writing handle pointer. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_Error* | An error occurred. |
| *kStatus_SerialManager_HandleConflict* | The writing handle was opened. |
| *kStatus_SerialManager_Success* | The writing handle is opened. |

Example below shows how to use this API to write data. For task 1,

```
*   static uint8_t s_serialWriteHandleBuffer1[SERIAL_MANAGER_WRITE_HANDLE_SIZE
      ];
*   static serial_write_handle_t s_serialWriteHandle1 = &s_serialWriteHandleBuffer1[0];
*   static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
*   SerialManager_OpenWriteHandle(serialHandle, s_serialWriteHandle1);
*   SerialManager_InstallTxCallback(s_serialWriteHandle1, Task1_SerialManagerTxCallback,
      s_serialWriteHandle1);
*   SerialManager_WriteNonBlocking(s_serialWriteHandle1, s_nonBlockingWelcome1, sizeof(
      s_nonBlockingWelcome1) - 1);
*
```

For task 2,

```
*   static uint8_t s_serialWriteHandleBuffer2[SERIAL_MANAGER_WRITE_HANDLE_SIZE
      ];
*   static serial_write_handle_t s_serialWriteHandle2 = &s_serialWriteHandleBuffer2[0];
*   static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
*   SerialManager_OpenWriteHandle(serialHandle, s_serialWriteHandle2);
*   SerialManager_InstallTxCallback(s_serialWriteHandle2, Task2_SerialManagerTxCallback,
      s_serialWriteHandle2);
*   SerialManager_WriteNonBlocking(s_serialWriteHandle2, s_nonBlockingWelcome2, sizeof(
      s_nonBlockingWelcome2) - 1);
*
```

### 55.4.4 serial_manager_status_t SerialManager_CloseWriteHandle ( serial_write_handle_t *writeHandle* )

This function Closes a writing handle for the serial manager module.

Parameters

| | |
|---|---|
| *writeHandle* | The serial manager module writing handle pointer. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_Success* | The writing handle is closed. |

### 55.4.5 serial_manager_status_t SerialManager_OpenReadHandle ( serial_handle_t *serialHandle*, serial_read_handle_t *readHandle* )

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code kStatus_SerialManager_Busy would be returned when the previous reading handle is not closed. And There can only be one buffer for receiving for the reading handle at the same time.

Parameters

| | |
|---|---|
| *serialHandle* | The serial manager module handle pointer. |
| *readHandle* | The serial manager module reading handle pointer. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_Error* | An error occurred. |
| *kStatus_SerialManager_Success* | The reading handle is opened. |
| *kStatus_SerialManager_Busy* | Previous reading handle is not closed. |

Example below shows how to use this API to read data.

```
*   static uint8_t s_serialReadHandleBuffer[SERIAL_MANAGER_READ_HANDLE_SIZE];
*   static serial_read_handle_t s_serialReadHandle = &s_serialReadHandleBuffer[0];
*   SerialManager_OpenReadHandle(serialHandle, s_serialReadHandle);
*   static uint8_t s_nonBlockingBuffer[64];
*   SerialManager_InstallRxCallback(s_serialReadHandle, APP_SerialManagerRxCallback, s_serialReadHandle);
*   SerialManager_ReadNonBlocking(s_serialReadHandle, s_nonBlockingBuffer, sizeof(s_nonBlockingBuffer));
*
```

### 55.4.6 serial_manager_status_t SerialManager_CloseReadHandle ( serial_read_handle_t *readHandle* )

This function Closes a reading for the serial manager module.

Parameters

| | |
|---|---|
| *readHandle* | The serial manager module reading handle pointer. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_-Success* | The reading handle is closed. |

### 55.4.7 serial_manager_status_t SerialManager_WriteBlocking ( serial-_write_handle_t *writeHandle,* uint8_t * *buffer,* uint32_t *length* )

This is a blocking function, which polls the sending queue, waits for the sending queue to be empty. This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

Note

The function SerialManager_WriteBlocking and the function #SerialManager_WriteNonBlocking cannot be used at the same time. And, the function #SerialManager_CancelWriting cannot be used to abort the transmission of this function.

Parameters

| | |
|---|---|
| *writeHandle* | The serial manager module handle pointer. |
| *buffer* | Start address of the data to write. |
| *length* | Length of the data to write. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_-Success* | Successfully sent all data. |
| *kStatus_SerialManager_-Busy* | Previous transmission still not finished; data not all sent yet. |

| | |
|---|---|
| *kStatus_SerialManager_Error* | An error occurred. |

### 55.4.8 serial_manager_status_t SerialManager_ReadBlocking ( serial_read_handle_t *readHandle*, uint8_t * *buffer*, uint32_t *length* )

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

Note

The function SerialManager_ReadBlocking and the function #SerialManager_ReadNonBlocking cannot be used at the same time. And, the function #SerialManager_CancelReading cannot be used to abort the transmission of this function.

Parameters

| | |
|---|---|
| *readHandle* | The serial manager module handle pointer. |
| *buffer* | Start address of the data to store the received data. |
| *length* | The length of the data to be received. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_Success* | Successfully received all data. |
| *kStatus_SerialManager_Busy* | Previous transmission still not finished; data not all received yet. |
| *kStatus_SerialManager_Error* | An error occurred. |

### 55.4.9 serial_manager_status_t SerialManager_EnterLowpower ( serial_handle_t *serialHandle* )

This function is used to prepare to enter low power consumption.

Parameters

| | |
|---|---|
| *serialHandle* | The serial manager module handle pointer. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_-Success* | Successful operation. |

### 55.4.10 serial_manager_status_t SerialManager_ExitLowpower ( serial_handle_t *serialHandle* )

This function is used to restore from low power consumption.

Parameters

| | |
|---|---|
| *serialHandle* | The serial manager module handle pointer. |

Return values

| | |
|---|---|
| *kStatus_SerialManager_-Success* | Successful operation. |

## 55.5 Serial Port Uart

### 55.5.1 Overview

#### Data Structures

- struct serial_port_uart_config_t
  *serial port uart config struct More...*

#### Macros

- #define SERIAL_PORT_UART_HANDLE_SIZE (4U)
  *serial port uart handle size*

#### Enumerations

- enum serial_port_uart_parity_mode_t {
  kSerialManager_UartParityDisabled = 0x0U,
  kSerialManager_UartParityEven = 0x1U,
  kSerialManager_UartParityOdd = 0x2U }
  *serial port uart parity mode*
- enum serial_port_uart_stop_bit_count_t {
  kSerialManager_UartOneStopBit = 0U,
  kSerialManager_UartTwoStopBit = 1U }
  *serial port uart stop bit count*

### 55.5.2 Data Structure Documentation

#### 55.5.2.1 struct serial_port_uart_config_t

#### Data Fields

- uint32_t clockRate
  *clock rate*
- uint32_t baudRate
  *baud rate*
- serial_port_uart_parity_mode_t parityMode
  *Parity mode, disabled (default), even, odd.*
- serial_port_uart_stop_bit_count_t stopBitCount
  *Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- uint8_t instance
  *Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.*
- uint8_t enableRx
  *Enable RX.*
- uint8_t enableTx

*Enable TX.*

#### 55.5.2.1.0.78 Field Documentation

#### 55.5.2.1.0.78.1 uint8_t serial_port_uart_config_t::instance

### 55.5.3 Enumeration Type Documentation

#### 55.5.3.1 enum serial_port_uart_parity_mode_t

Enumerator

***kSerialManager_UartParityDisabled*** Parity disabled.
***kSerialManager_UartParityEven*** Parity even enabled.
***kSerialManager_UartParityOdd*** Parity odd enabled.

#### 55.5.3.2 enum serial_port_uart_stop_bit_count_t

Enumerator

***kSerialManager_UartOneStopBit*** One stop bit.
***kSerialManager_UartTwoStopBit*** Two stop bits.

## 55.6 Serial Port USB

### 55.6.1 Overview

#### Modules

- USB Device Configuration

#### Data Structures

- struct serial_port_usb_cdc_config_t
  *serial port usb config struct More...*

#### Macros

- #define SERIAL_PORT_USB_CDC_HANDLE_SIZE (72)
  *serial port usb handle size*
- #define USB_DEVICE_INTERRUPT_PRIORITY (3U)
  *USB interrupt priority.*

#### Enumerations

- enum serial_port_usb_cdc_controller_index_t {
  kSerialManager_UsbControllerKhci0 = 0U,
  kSerialManager_UsbControllerKhci1 = 1U,
  kSerialManager_UsbControllerEhci0 = 2U,
  kSerialManager_UsbControllerEhci1 = 3U,
  kSerialManager_UsbControllerLpcIp3511Fs0 = 4U,
  kSerialManager_UsbControllerLpcIp3511Fs1 = 5U,
  kSerialManager_UsbControllerLpcIp3511Hs0 = 6U,
  kSerialManager_UsbControllerLpcIp3511Hs1 = 7U,
  kSerialManager_UsbControllerOhci0 = 8U,
  kSerialManager_UsbControllerOhci1 = 9U,
  kSerialManager_UsbControllerIp3516Hs0 = 10U,
  kSerialManager_UsbControllerIp3516Hs1 = 11U }
  *USB controller ID.*

## 55.6.2 Data Structure Documentation

### 55.6.2.1 struct serial_port_usb_cdc_config_t

#### Data Fields

- serial_port_usb_cdc_controller_index_t controllerIndex
  *controller index*

## 55.6.3 Enumeration Type Documentation

### 55.6.3.1 enum serial_port_usb_cdc_controller_index_t

Enumerator

***kSerialManager_UsbControllerKhci0*** KHCI 0U.
***kSerialManager_UsbControllerKhci1*** KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.
***kSerialManager_UsbControllerEhci0*** EHCI 0U.
***kSerialManager_UsbControllerEhci1*** EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.
***kSerialManager_UsbControllerLpcIp3511Fs0*** LPC USB IP3511 FS controller 0.
***kSerialManager_UsbControllerLpcIp3511Fs1*** LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.
***kSerialManager_UsbControllerLpcIp3511Hs0*** LPC USB IP3511 HS controller 0.
***kSerialManager_UsbControllerLpcIp3511Hs1*** LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.
***kSerialManager_UsbControllerOhci0*** OHCI 0U.
***kSerialManager_UsbControllerOhci1*** OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.
***kSerialManager_UsbControllerIp3516Hs0*** IP3516HS 0U.
***kSerialManager_UsbControllerIp3516Hs1*** IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

### 55.6.4 USB Device Configuration

#### 55.6.4.1 Overview

**Macros**

- #define USB_DEVICE_CONFIG_SELF_POWER (1U)
  *Whether device is self power.*
- #define USB_DEVICE_CONFIG_ENDPOINTS (4U)
  *How many endpoints are supported in the stack.*
- #define USB_DEVICE_CONFIG_USE_TASK (0U)
  *Whether the device task is enabled.*
- #define USB_DEVICE_CONFIG_MAX_MESSAGES (8U)
  *How many the notification message are supported when the device task is enabled.*
- #define USB_DEVICE_CONFIG_USB20_TEST_MODE (0U)
  *Whether test mode enabled.*
- #define USB_DEVICE_CONFIG_CV_TEST (0U)
  *Whether device CV test is enabled.*
- #define USB_DEVICE_CONFIG_COMPLIANCE_TEST (0U)
  *Whether device compliance test is enabled.*
- #define USB_DEVICE_CONFIG_KEEP_ALIVE_MODE (0U)
  *Whether the keep alive feature enabled.*
- #define USB_DEVICE_CONFIG_BUFFER_PROPERTY_CACHEABLE (0U)
  *Whether the transfer buffer is cache-enabled or not.*
- #define USB_DEVICE_CONFIG_LOW_POWER_MODE (0U)
  *Whether the low power mode is enabled or not.*
- #define USB_DEVICE_CONFIG_REMOTE_WAKEUP (0U)
  *The device remote wakeup is unsupported.*
- #define USB_DEVICE_CONFIG_DETACH_ENABLE (0U)
  *Whether the device detached feature is enabled or not.*
- #define USB_DEVICE_CONFIG_ERROR_HANDLING (0U)
  *Whether handle the USB bus error.*
- #define USB_DEVICE_CHARGER_DETECT_ENABLE (0U)
  *Whether the device charger detect feature is enabled or not.*

**class instance define**

- #define USB_DEVICE_CONFIG_HID (0U)
  *HID instance count.*
- #define USB_DEVICE_CONFIG_CDC_ACM (1U)
  *CDC ACM instance count.*
- #define USB_DEVICE_CONFIG_MSC (0U)
  *MSC instance count.*
- #define USB_DEVICE_CONFIG_AUDIO (0U)
  *Audio instance count.*
- #define USB_DEVICE_CONFIG_PHDC (0U)
  *PHDC instance count.*
- #define USB_DEVICE_CONFIG_VIDEO (0U)
  *Video instance count.*
- #define USB_DEVICE_CONFIG_CCID (0U)

*CCID instance count.*

- #define USB_DEVICE_CONFIG_PRINTER (0U)
  *Printer instance count.*
- #define USB_DEVICE_CONFIG_DFU (0U)
  *DFU instance count.*

#### 55.6.4.2 Macro Definition Documentation

##### 55.6.4.2.1 #define USB_DEVICE_CONFIG_SELF_POWER (1U)

1U supported, 0U not supported

##### 55.6.4.2.2 #define USB_DEVICE_CONFIG_ENDPOINTS (4U)

##### 55.6.4.2.3 #define USB_DEVICE_CONFIG_USE_TASK (0U)

##### 55.6.4.2.4 #define USB_DEVICE_CONFIG_MAX_MESSAGES (8U)

##### 55.6.4.2.5 #define USB_DEVICE_CONFIG_USB20_TEST_MODE (0U)

##### 55.6.4.2.6 #define USB_DEVICE_CONFIG_CV_TEST (0U)

##### 55.6.4.2.7 #define USB_DEVICE_CONFIG_COMPLIANCE_TEST (0U)

If the macro is enabled, the test mode and CV test macroes will be set.

##### 55.6.4.2.8 #define USB_DEVICE_CONFIG_KEEP_ALIVE_MODE (0U)

##### 55.6.4.2.9 #define USB_DEVICE_CONFIG_BUFFER_PROPERTY_CACHEABLE (0U)

##### 55.6.4.2.10 #define USB_DEVICE_CONFIG_LOW_POWER_MODE (0U)

##### 55.6.4.2.11 #define USB_DEVICE_CONFIG_REMOTE_WAKEUP (0U)

##### 55.6.4.2.12 #define USB_DEVICE_CONFIG_DETACH_ENABLE (0U)

##### 55.6.4.2.13 #define USB_DEVICE_CONFIG_ERROR_HANDLING (0U)

##### 55.6.4.2.14 #define USB_DEVICE_CHARGER_DETECT_ENABLE (0U)

## 55.7 Serial Port SWO

### 55.7.1 Overview

#### Data Structures

- struct serial_port_swo_config_t
  *serial port swo config struct More...*

#### Macros

- #define SERIAL_PORT_SWO_HANDLE_SIZE (12U)
  *serial port swo handle size*

#### Enumerations

- enum serial_port_swo_protocol_t {
  kSerialManager_SwoProtocolManchester = 1U,
  kSerialManager_SwoProtocolNrz = 2U }
  *serial port swo protocol*

### 55.7.2 Data Structure Documentation

#### 55.7.2.1 struct serial_port_swo_config_t

#### Data Fields

- uint32_t clockRate
  *clock rate*
- uint32_t baudRate
  *baud rate*
- uint32_t port
  *Port used to transfer data.*
- serial_port_swo_protocol_t protocol
  *SWO protocol.*

### 55.7.3 Enumeration Type Documentation

#### 55.7.3.1 enum serial_port_swo_protocol_t

Enumerator

***kSerialManager_SwoProtocolManchester*** SWO Manchester protocol.
***kSerialManager_SwoProtocolNrz*** SWO UART/NRZ protocol.

## 55.8 Serial Port Virtual USB

### 55.8.1 Overview

This chapter describes how to redirect the serial manager stream to application CDC. The weak functions can be implemented by application to redirect the serial manager stream. The weak functions are following,

USB_DeviceVcomInit - Initialize the cdc vcom.

USB_DeviceVcomDeinit - De-initialize the cdc vcom.

USB_DeviceVcomWrite - Write data with non-blocking mode. After data is sent, the installed TX callback should be called with the result.

USB_DeviceVcomRead - Read data with non-blocking mode. After data is received, the installed RX callback should be called with the result.

USB_DeviceVcomCancelWrite - Cancel write request.

USB_DeviceVcomInstallTxCallback - Install TX callback.

USB_DeviceVcomInstallRxCallback - Install RX callback.

USB_DeviceVcomIsrFunction - The hardware ISR function.

### Data Structures

- struct serial_port_usb_cdc_virtual_config_t
  *serial port usb config struct More...*

### Macros

- #define SERIAL_PORT_USB_VIRTUAL_HANDLE_SIZE (40U)
  *serial port uart handle size*

### Enumerations

- enum serial_port_usb_cdc_virtual_controller_index_t {
  kSerialManager_UsbVirtualControllerKhci0 = 0U,
  kSerialManager_UsbVirtualControllerKhci1 = 1U,
  kSerialManager_UsbVirtualControllerEhci0 = 2U,
  kSerialManager_UsbVirtualControllerEhci1 = 3U,
  kSerialManager_UsbVirtualControllerLpcIp3511Fs0 = 4U,
  kSerialManager_UsbVirtualControllerLpcIp3511Fs1,
  kSerialManager_UsbVirtualControllerLpcIp3511Hs0 = 6U,
  kSerialManager_UsbVirtualControllerLpcIp3511Hs1,
  kSerialManager_UsbVirtualControllerOhci0 = 8U,
  kSerialManager_UsbVirtualControllerOhci1 = 9U,
  kSerialManager_UsbVirtualControllerIp3516Hs0 = 10U,
  kSerialManager_UsbVirtualControllerIp3516Hs1 = 11U }
  *USB controller ID.*

### Variables

- serial_port_usb_cdc_virtual_controller_index_t serial_port_usb_cdc_virtual_config_t::controller-Index
  *controller index*

## 55.8.2 Data Structure Documentation

### 55.8.2.1 struct serial_port_usb_cdc_virtual_config_t

### Data Fields

- serial_port_usb_cdc_virtual_controller_index_t controllerIndex
  *controller index*

## 55.8.3 Enumeration Type Documentation

### 55.8.3.1 enum serial_port_usb_cdc_virtual_controller_index_t

Enumerator

***kSerialManager_UsbVirtualControllerKhci0*** KHCI 0U.

***kSerialManager_UsbVirtualControllerKhci1*** KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

***kSerialManager_UsbVirtualControllerEhci0*** EHCI 0U.

***kSerialManager_UsbVirtualControllerEhci1*** EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

***kSerialManager_UsbVirtualControllerLpcIp3511Fs0*** LPC USB IP3511 FS controller 0.

***kSerialManager_UsbVirtualControllerLpcIp3511Fs1*** LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

***kSerialManager_UsbVirtualControllerLpcIp3511Hs0*** LPC USB IP3511 HS controller 0.

***kSerialManager_UsbVirtualControllerLpcIp3511Hs1*** LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

***kSerialManager_UsbVirtualControllerOhci0*** OHCI 0U.

***kSerialManager_UsbVirtualControllerOhci1*** OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

***kSerialManager_UsbVirtualControllerIp3516Hs0*** IP3516HS 0U.

***kSerialManager_UsbVirtualControllerIp3516Hs1*** IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

# Chapter 56
# GenericList

## 56.1 Overview

### Data Structures

- struct list_handle_t
  *The list structure. More...*
- struct list_element_handle_t
  *The list element. More...*

### Enumerations

- enum list_status_t {
  kLIST_Ok = kStatus_Success,
  kLIST_DuplicateError = MAKE_STATUS(kStatusGroup_LIST, 1),
  kLIST_Full = MAKE_STATUS(kStatusGroup_LIST, 2),
  kLIST_Empty = MAKE_STATUS(kStatusGroup_LIST, 3),
  kLIST_OrphanElement = MAKE_STATUS(kStatusGroup_LIST, 4) }

### Functions

- void LIST_Init (list_handle_t list, uint32_t max)

- list_handle_t LIST_GetList (list_element_handle_t element)
  *Gets the list that contains the given element.*
- list_status_t LIST_AddHead (list_handle_t list, list_element_handle_t element)
  *Links element to the head of the list.*
- list_status_t LIST_AddTail (list_handle_t list, list_element_handle_t element)
  *Links element to the tail of the list.*
- list_element_handle_t LIST_RemoveHead (list_handle_t list)
  *Unlinks element from the head of the list.*
- list_element_handle_t LIST_GetHead (list_handle_t list)
  *Gets head element handle.*
- list_element_handle_t LIST_GetNext (list_element_handle_t element)
  *Gets next element handle for given element handle.*
- list_element_handle_t LIST_GetPrev (list_element_handle_t element)
  *Gets previous element handle for given element handle.*
- list_status_t LIST_RemoveElement (list_element_handle_t element)
  *Unlinks an element from its list.*
- list_status_t LIST_AddPrevElement (list_element_handle_t element, list_element_handle_t newElement)
  *Links an element in the previous position relative to a given member of a list.*
- uint32_t LIST_GetSize (list_handle_t list)
  *Gets the current size of a list.*

- uint32_t LIST_GetAvailableSize (list_handle_t list)
  *Gets the number of free places in the list.*

## 56.2 Data Structure Documentation

### 56.2.1 struct list_t

#### Data Fields

- struct list_element_tag * head
  *list head*
- struct list_element_tag * tail
  *list tail*
- uint16_t size
  *list size*
- uint16_t max
  *list max number of elements*

### 56.2.2 struct list_element_t

#### Data Fields

- struct list_element_tag * next
  *next list element*
- struct list_element_tag * prev
  *previous list element*
- struct list_tag * list
  *pointer to the list*

## 56.3 Enumeration Type Documentation

### 56.3.1 enum list_status_t

Include

Public macro definitions

Public type definitions

The list status

Enumerator

***kLIST_Ok*** Success.
***kLIST_DuplicateError*** Duplicate Error.
***kLIST_Full*** FULL.
***kLIST_Empty*** Empty.
***kLIST_OrphanElement*** Orphan Element.

## 56.4 Function Documentation

### 56.4.1 void LIST_Init ( list_handle_t *list,* uint32_t *max* )

Public prototypes

Initialize the list.

This function initialize the list.

Parameters

| | |
|---|---|
| *list* | - List handle to initialize. |
| *max* | - Maximum number of elements in list. 0 for unlimited. |

### 56.4.2 list_handle_t LIST_GetList ( list_element_handle_t *element* )

Parameters

| | |
|---|---|
| *element* | - Handle of the element. |

Return values

| | |
|---|---|
| *NULL* | if element is orphan, Handle of the list the element is inserted into. |

### 56.4.3 list_status_t LIST_AddHead ( list_handle_t *list,* list_element_handle_t *element* )

Parameters

| | |
|---|---|
| *list* | - Handle of the list. |
| *element* | - Handle of the element. |

Return values

| | |
|---|---|
| *kLIST_Full* | if list is full, kLIST_Ok if insertion was successful. |

### 56.4.4 list_status_t LIST_AddTail ( list_handle_t *list,* list_element_handle_t *element* )

Parameters

| | |
|---|---|
| *list* | - Handle of the list. |
| *element* | - Handle of the element. |

Return values

| | |
|---|---|
| *kLIST_Full* | if list is full, kLIST_Ok if insertion was successful. |

### 56.4.5 list_element_handle_t LIST_RemoveHead ( list_handle_t *list* )

Parameters

| | |
|---|---|
| *list* | - Handle of the list. |

Return values

| | |
|---|---|
| *NULL* | if list is empty, handle of removed element(pointer) if removal was successful. |

### 56.4.6 list_element_handle_t LIST_GetHead ( list_handle_t *list* )

Parameters

| | |
|---|---|
| *list* | - Handle of the list. |

Return values

| | |
|---|---|
| *NULL* | if list is empty, handle of removed element(pointer) if removal was successful. |

### 56.4.7 list_element_handle_t LIST_GetNext ( list_element_handle_t *element* )

Parameters

| | |
|---|---|
| *element* | - Handle of the element. |

Return values

| | |
|---|---|
| *NULL* | if list is empty, handle of removed element(pointer) if removal was successful. |

### 56.4.8 list_element_handle_t LIST_GetPrev ( list_element_handle_t *element* )

Parameters

| | |
|---|---|
| *element* | - Handle of the element. |

Return values

| | |
|---|---|
| *NULL* | if list is empty, handle of removed element(pointer) if removal was successful. |

### 56.4.9 list_status_t LIST_RemoveElement ( list_element_handle_t *element* )

Parameters

| | |
|---|---|
| *element* | - Handle of the element. |

Return values

| | |
|---|---|
| *kLIST_OrphanElement* | if element is not part of any list. |
| *kLIST_Ok* | if removal was successful. |

### 56.4.10 list_status_t LIST_AddPrevElement ( list_element_handle_t *element,* list_element_handle_t *newElement* )

Parameters

| | |
|---|---|
| *element* | - Handle of the element. |
| *newElement* | - New element to insert before the given member. |

Return values

| | |
|---|---|
| *kLIST_OrphanElement* | if element is not part of any list. |
| *kLIST_Ok* | if removal was successful. |

### 56.4.11 uint32_t LIST_GetSize ( list_handle_t *list* )

Parameters

| | |
|---|---|
| *list* | - Handle of the list. |

Return values

| | |
|---|---|
| *Current* | size of the list. |

### 56.4.12 uint32_t LIST_GetAvailableSize ( list_handle_t *list* )

Parameters

| | |
|---|---|
| *list* | - Handle of the list. |

Return values

| | |
|---|---|
| *Available* | size of the list. |

# Chapter 57
# UART_Adapter

## 57.1 Overview

### Data Structures

- struct hal_uart_config_t
  *uart configuration structure. More...*
- struct hal_uart_transfer_t
  *uart transfer structure. More...*

### Macros

- #define UART_ADAPTER_NON_BLOCKING_MODE (0U)
  *Enable or disable Uart adapter non-blocking mode (1 - enable, 0 - disable)*
- #define HAL_UART_TRANSFER_MODE (0U)
  *Whether enable transactional function of the uart.*

### Typedefs

- typedef void(∗ hal_uart_transfer_callback_t )(hal_uart_handle_t handle, hal_uart_status_t status, void ∗callbackParam)
  *uart transfer callback function.*

### Enumerations

- enum hal_uart_status_t {
  kStatus_HAL_UartSuccess = kStatus_Success,
  kStatus_HAL_UartTxBusy = MAKE_STATUS(kStatusGroup_HAL_UART, 1),
  kStatus_HAL_UartRxBusy = MAKE_STATUS(kStatusGroup_HAL_UART, 2),
  kStatus_HAL_UartTxIdle = MAKE_STATUS(kStatusGroup_HAL_UART, 3),
  kStatus_HAL_UartRxIdle = MAKE_STATUS(kStatusGroup_HAL_UART, 4),
  kStatus_HAL_UartBaudrateNotSupport,
  kStatus_HAL_UartProtocolError,
  kStatus_HAL_UartError = MAKE_STATUS(kStatusGroup_HAL_UART, 7) }
  *uart status*
- enum hal_uart_parity_mode_t {
  kHAL_UartParityDisabled = 0x0U,
  kHAL_UartParityEven = 0x1U,
  kHAL_UartParityOdd = 0x2U }
  *uart parity mode.*
- enum hal_uart_stop_bit_count_t {
  kHAL_UartOneStopBit = 0U,
  kHAL_UartTwoStopBit = 1U }
  *uart stop bit count.*

### Initialization and deinitialization

- hal_uart_status_t HAL_UartInit (hal_uart_handle_t handle, hal_uart_config_t *config)
  *Initializes a uart instance with the uart handle and the user configuration structure.*
- hal_uart_status_t HAL_UartDeinit (hal_uart_handle_t handle)
  *Deinitializes a uart instance.*

### Blocking bus Operations

- hal_uart_status_t HAL_UartReceiveBlocking (hal_uart_handle_t handle, uint8_t *data, size_t length)
  *Reads RX data register using a blocking method.*
- hal_uart_status_t HAL_UartSendBlocking (hal_uart_handle_t handle, const uint8_t *data, size_t length)
  *Writes to the TX register using a blocking method.*

## 57.2 Data Structure Documentation

### 57.2.1 struct hal_uart_config_t

### Data Fields

- uint32_t srcClock_Hz
  *Source clock.*
- uint32_t baudRate_Bps
  *Baud rate.*
- hal_uart_parity_mode_t parityMode
  *Parity mode, disabled (default), even, odd.*
- hal_uart_stop_bit_count_t stopBitCount
  *Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- uint8_t enableRx
  *Enable RX.*
- uint8_t enableTx
  *Enable TX.*
- uint8_t instance
  *Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.*

#### 57.2.1.0.0.1 Field Documentation

##### 57.2.1.0.0.1.1 uint8_t hal_uart_config_t::instance

Invalid instance value will cause initialization failure.

### 57.2.2 struct hal_uart_transfer_t

### Data Fields

- uint8_t * data

*The buffer of data to be transfer.*

- size_t dataSize
  *The byte count to be transfer.*

#### 57.2.2.0.0.2 Field Documentation

#### 57.2.2.0.0.2.1 uint8_t* hal_uart_transfer_t::data

#### 57.2.2.0.0.2.2 size_t hal_uart_transfer_t::dataSize

## 57.3 Macro Definition Documentation

### 57.3.1 #define HAL_UART_TRANSFER_MODE (0U)

(0 - disable, 1 - enable)

## 57.4 Typedef Documentation

### 57.4.1 typedef void(* hal_uart_transfer_callback_t)(hal_uart_handle_t handle, hal_uart_status_t status, void *callbackParam)

## 57.5 Enumeration Type Documentation

### 57.5.1 enum hal_uart_status_t

Enumerator

***kStatus_HAL_UartSuccess*** Successfully.
***kStatus_HAL_UartTxBusy*** TX busy.
***kStatus_HAL_UartRxBusy*** RX busy.
***kStatus_HAL_UartTxIdle*** HAL uart transmitter is idle.
***kStatus_HAL_UartRxIdle*** HAL uart receiver is idle.
***kStatus_HAL_UartBaudrateNotSupport*** Baudrate is not support in current clock source.
***kStatus_HAL_UartProtocolError*** Error occurs for Noise, Framing, Parity, etc. For transactional transfer, The up layer needs to abort the transfer and then starts again
***kStatus_HAL_UartError*** Error occurs on HAL uart.

### 57.5.2 enum hal_uart_parity_mode_t

Enumerator

***kHAL_UartParityDisabled*** Parity disabled.
***kHAL_UartParityEven*** Parity even enabled.
***kHAL_UartParityOdd*** Parity odd enabled.

### 57.5.3 enum hal_uart_stop_bit_count_t

Enumerator

***kHAL_UartOneStopBit*** One stop bit.
***kHAL_UartTwoStopBit*** Two stop bits.

## 57.6 Function Documentation

### 57.6.1 hal_uart_status_t HAL_UartInit ( hal_uart_handle_t *handle*, hal_uart_config_t * *config* )

This function configures the uart module with user-defined settings. The user can configure the configuration structure. The parameter handle is a pointer to point to a memory space of size #HAL_UART_HANDLE_SIZE allocated by the caller. Example below shows how to use this API to configure the uart.

```
*   uint8_t g_UartHandleBuffer[HAL_UART_HANDLE_SIZE];
*   hal_uart_handle_t g_UartHandle = &g_UartHandleBuffer[0];
*   hal_uart_config_t config;
*   config.srcClock_Hz = 48000000;
*   config.baudRate_Bps = 115200U;
*   config.parityMode = kHAL_UartParityDisabled;
*   config.stopBitCount = kHAL_UartOneStopBit;
*   config.enableRx = 1;
*   config.enableTx = 1;
*   config.instance = 0;
*   HAL_UartInit(g_UartHandle, &config);
*
```

Parameters

| | |
|---|---|
| *handle* | Pointer to point to a memory space of size #HAL_UART_HANDLE_SIZE allocated by the caller. |
| *config* | Pointer to user-defined configuration structure. |

Return values

| | |
|---|---|
| *kStatus_HAL_Uart-BaudrateNotSupport* | Baudrate is not support in current clock source. |
| *kStatus_HAL_Uart-Success* | uart initialization succeed |

### 57.6.2 hal_uart_status_t HAL_UartDeinit ( hal_uart_handle_t *handle* )

This function waits for TX complete, disables TX and RX, and disables the uart clock.

Parameters

| | |
|---|---|
| *handle* | uart handle pointer. |

Return values

| | |
|---|---|
| *kStatus_HAL_Uart-Success* | uart de-initialization succeed |

### 57.6.3 hal_uart_status_t HAL_UartReceiveBlocking ( hal_uart_handle_t *handle*, uint8_t * *data*, size_t *length* )

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the RX register.

Note

The function HAL_UartReceiveBlocking and the function #HAL_UartTransferReceiveNon-Blocking cannot be used at the same time. And, the function #HAL_UartTransferAbortReceive cannot be used to abort the transmission of this function.

Parameters

| | |
|---|---|
| *handle* | uart handle pointer. |
| *data* | Start address of the buffer to store the received data. |
| *length* | Size of the buffer. |

Return values

| | |
|---|---|
| *kStatus_HAL_UartError* | An error occurred while receiving data. |
| *kStatus_HAL_UartParity-Error* | A parity error occurred while receiving data. |
| *kStatus_HAL_Uart-Success* | Successfully received all data. |

### 57.6.4 hal_uart_status_t HAL_UartSendBlocking ( hal_uart_handle_t *handle*, const uint8_t * *data*, size_t *length* )

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Note

The function HAL_UartSendBlocking and the function #HAL_UartTransferSendNonBlocking cannot be used at the same time. And, the function #HAL_UartTransferAbortSend cannot be used to abort the transmission of this function.

Parameters

| | |
|---|---|
| *handle* | uart handle pointer. |
| *data* | Start address of the data to write. |
| *length* | Size of the data to write. |

Return values

| | |
|---|---|
| *kStatus_HAL_Uart-Success* | Successfully sent all data. |

***How to Reach Us:***

**Home Page:**
nxp.com

**Web Support:**
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.