MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKAPIRM
Rev. 0
Dec 2019

Contents

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

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^{TM} . 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_DMA_Busy = 5000
- kStatus_DMIC_Busy = 5800
- kStatus_DMIC_Idle = 5801
- kStatus_DMIC_OverRunError = 5802
- kStatus_DMIC_UnderRunError = 5803
- kStatus_I2C_Busy = 2600
- kStatus_I2C_Idle = 2601
- kStatus_I2C_Nak = 2602
- kStatus_I2C_InvalidParameter = 2603
- kStatus_I2C_BitError = 2604
- kStatus_I2C_ArbitrationLost = 2605
- kStatus_I2C_NoTransferInProgress = 2606
- kStatus_I2C_DmaRequestFail = 2607
- #kStatus_I2C_StartStopError = 2608
- #kStatus_I2C_UnexpectedState = 2609
- kStatus_I2C_Timeout = 2610
- kStatus_SPI_Busy = 1400
- kStatus_SPI_Idle = 1401
- kStatus_SPI_Error = 1402
- kStatus_USART_TxBusy = 5700
- kStatus_USART_RxBusy = 5701
- kStatus_USART_TxIdle = 5702
- kStatus_USART_RxIdle = 5703
- kStatus_USART_TxError = 5707
- kStatus_USART_RxError = 5709
- kStatus_USART_RxRingBufferOverrun = 5708
- kStatus_USART_NoiseError = 5710
- kStatus_USART_FramingError = 5711
- kStatus_USART_ParityError = 5712
- kStatus_USART_BaudrateNotSupport = 5713
- kStatus_SPIFI_Busy = 5900
- kStatus_SPIFI_Idle = 5901
- kStatus_SPIFI_Error = 5902
- kStatus_FLASHIAP_Success = 0
- kStatus_FLASHIAP_InvalidCommand = 2501
- kStatus_FLASHIAP_SrcAddrError = 2502
- kStatus_FLASHIAP_DstAddrError = 2503
- kStatus_FLASHIAP_SrcAddrNotMapped = 2504

- kStatus_FLASHIAP_DstAddrNotMapped = 2505
- kStatus_FLASHIAP_CountError = 2506
- kStatus_FLASHIAP_InvalidSector = 2507
- kStatus_FLASHIAP_SectorNotblank = 2508
- kStatus_FLASHIAP_NotPrepared = 2509
- kStatus_FLASHIAP_CompareError = 2510
- kStatus_FLASHIAP_Busy = 2511
- kStatus_FLASHIAP_ParamError = 2512
- kStatus_FLASHIAP_AddrError = 2513
- kStatus_FLASHIAP_AddrNotMapped = 2514
- kStatus_FLASHIAP_NoPower = 2514
- kStatus_FLASHIAP_NoClock = 2527

Chapter 3
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 4
Architectural Overview

This chapter provides the architectural overview for the MCUXpresso Software Development Kit (MCU-Xpresso 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

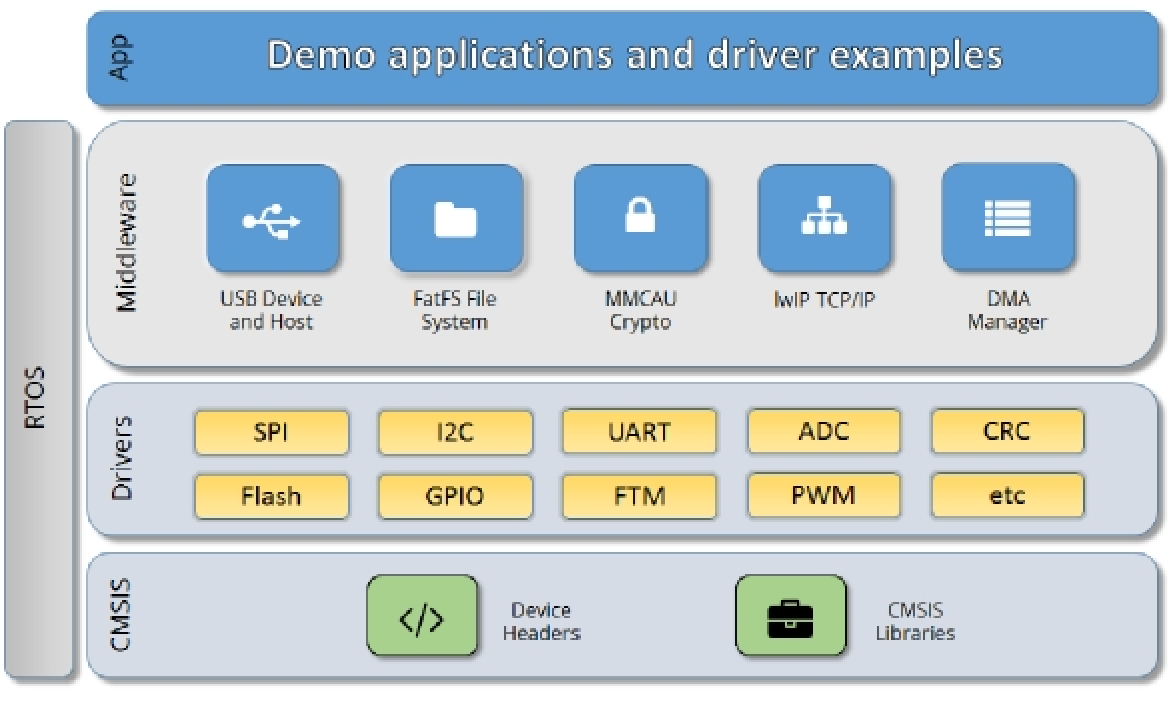

Figure 1: MCUXpresso SDK Block Diagram

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/DM-A 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 5
ADC: 12-bit SAR Analog-to-Digital Converter Driver

5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit Successive Approximation (SAR) 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 Interrupt Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/fsl_-adc

Files

- file fsl_adc.h

Data Structures

- struct adc_config_t
  Define structure for configuring the block. More...
- struct adc_conv_seq_config_t
  Define structure for configuring conversion sequence. More...
- struct adc_result_info_t
  Define structure of keeping conversion result information. More...

Enumerations

- enum _adc_status_flags {
  kADC_ThresholdCompareFlagOnChn0 = 1U << 0U,
  kADC_ThresholdCompareFlagOnChn1 = 1U << 1U,
  kADC_ThresholdCompareFlagOnChn2 = 1U << 2U,
  kADC_ThresholdCompareFlagOnChn3 = 1U << 3U,
  kADC_ThresholdCompareFlagOnChn4 = 1U << 4U,
  kADC_ThresholdCompareFlagOnChn5 = 1U << 5U,
  kADC_ThresholdCompareFlagOnChn6 = 1U << 6U,
  kADC_ThresholdCompareFlagOnChn7 = 1U << 7U,
  kADC_ThresholdCompareFlagOnChn8 = 1U << 8U,
  kADC_ThresholdCompareFlagOnChn9 = 1U << 9U,
  kADC_ThresholdCompareFlagOnChn10 = 1U << 10U,
  kADC_ThresholdCompareFlagOnChn11 = 1U << 11U,
  kADC_OverrunFlagForChn0,
  kADC_OverrunFlagForChn1,
  kADC_OverrunFlagForChn2,
  kADC_OverrunFlagForChn3,
  kADC_OverrunFlagForChn4,
  kADC_OverrunFlagForChn5,
  kADC_OverrunFlagForChn6,
  kADC_OverrunFlagForChn7,
  kADC_OverrunFlagForChn8,
  kADC_OverrunFlagForChn9,
  kADC_OverrunFlagForChn10,
  kADC_OverrunFlagForChn11,
  kADC_GlobalOverrunFlagForSeqA = 1U << 24U,
  kADC_GlobalOverrunFlagForSeqB = 1U << 25U,
  kADC_ConvSeqAInterruptFlag = 1U << 28U,
  kADC_ConvSeqBInterruptFlag = 1U << 29U,
  kADC_ThresholdCompareInterruptFlag = 1U << 30U,
  kADC_OverrunInterruptFlag = (int)(1U << 31U) }
  Flags.
- enum _adc_interrupt_enable {
  kADC_ConvSeqAInterruptEnable = ADC_INTEN_SEQA_INTEN_MASK,
  kADC_OverrunInterruptEnable = ADC_INTEN_OVR_INTEN_MASK }
  Interrupts.
- enum adc_clock_mode_t {
  kADC_ClockSynchronousMode,
  kADC_ClockAsynchronousMode = 1U }
  Define selection of clock mode.
- enum adc_resolution_t {
  kADC_Resolution6bit = 3U,
  kADC_Resolution8bit = 2U,
  kADC_Resolution10bit = 1U,

kADC_Resolution12bit = 0U }
  Define selection of resolution.
- enum adc_trigger_polarity_t {
  kADC_TriggerPolarityNegativeEdge = 0U,
  kADC_TriggerPolarityPositiveEdge = 1U }
  Define selection of polarity of selected input trigger for conversion sequence.
- enum adc_priority_t {
  kADC_PriorityLow = 0U,
  kADC_PriorityHigh = 1U }
  Define selection of conversion sequence's priority.
- enum adc_seq_interrupt_mode_t {
  kADC_InterruptForEachConversion = 0U,
  kADC_InterruptForEachSequence = 1U }
  Define selection of conversion sequence's interrupt.
- enum adc_threshold_compare_status_t {
  kADC_ThresholdCompareInRange = 0U,
  kADC_ThresholdCompareBelowRange = 1U,
  kADC_ThresholdCompareAboveRange = 2U }
  Define status of threshold compare result.
- enum adc_threshold_crossing_status_t {
  kADC_ThresholdCrossingNoDetected = 0U,
  kADC_ThresholdCrossingDownward = 2U,
  kADC_ThresholdCrossingUpward = 3U }
  Define status of threshold crossing detection result.
- enum adc_threshold_interrupt_mode_t {
  kADC_ThresholdInterruptDisabled = 0U,
  kADC_ThresholdInterruptOnOutside = 1U,
  kADC_ThresholdInterruptOnCrossing = 2U }
  Define interrupt mode for threshold compare event.
- enum adc_inforesult_t {
  kADC_Resolution12bitInfoResultShift = 0U,
  kADC_Resolution10bitInfoResultShift = 2U,
  kADC_Resolution8bitInfoResultShift = 4U,
  kADC_Resolution6bitInfoResultShift = 6U }
  Define the info result mode of different resolution.
- enum adc_tempsensor_common_mode_t {
  kADC_HighNegativeOffsetAdded = 0x0U,
  kADC_IntermediateNegativeOffsetAdded,
  kADC_NoOffsetAdded = 0x8U,
  kADC_LowPositiveOffsetAdded = 0xcU }
  Define common modes for Temerature sensor.
- enum adc_second_control_t {
  kADC_Impedance621Ohm = 0x1U << 9U,
  kADC_Impedance55kOhm,
  kADC_Impedance87kOhm = 0x1fU << 9U,
  kADC_NormalFunctionalMode = 0x0U << 14U,
  kADC_MultiplexeTestMode = 0x1U << 14U,

kADC_ADCInUnityGainMode = 0x2U << 14U }
Define source impedance modes for GPADC control.

Driver version

- #define FSL_ADC_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))
  ADC driver version 2.3.1.

Initialization and Deinitialization

- void ADC_Init (ADC_Type *base, const adc_config_t *config)
  Initialize the ADC module.
- void ADC_Deinit (ADC_Type *base)
  Deinitialize the ADC module.
- void ADC_GetDefaultConfig (adc_config_t *config)
  Gets an available pre-defined settings for initial configuration.
- void ADC_EnableTemperatureSensor (ADC_Type *base, bool enable)
  Enable the internal temperature sensor measurement.

Control conversion sequence A.

- static void ADC_EnableConvSeqA (ADC_Type *base, bool enable)
  Enable the conversion sequence A.
- void ADC_SetConvSeqAConfig (ADC_Type *base, const adc_conv_seq_config_t *config)
  Configure the conversion sequence A.
- static void ADC_DoSoftwareTriggerConvSeqA (ADC_Type *base)
  Do trigger the sequence's conversion by software.
- static void ADC_EnableConvSeqABurstMode (ADC_Type *base, bool enable)
  Enable the burst conversion of sequence A.

Data result.

- bool ADC_GetConvSeqAGlobalConversionResult (ADC_Type *base, adc_result_info_t *info)
  Get the global ADC conversion infomation of sequence A.
- bool ADC_GetChannelConversionResult (ADC_Type *base, uint32_t channel, adc_result_info_t *info)
  Get the channel's ADC conversion completed under each conversion sequence.

Threshold function.

- static void ADC_SetThresholdPair0 (ADC_Type *base, uint32_t lowValue, uint32_t highValue)
  Set the threshhold pair 0 with low and high value.
- static void ADC_SetThresholdPair1 (ADC_Type *base, uint32_t lowValue, uint32_t highValue)
  Set the threshhold pair 1 with low and high value.
- static void ADC_SetChannelWithThresholdPair0 (ADC_Type *base, uint32_t channelMask)
  Set given channels to apply the threshold pare 0.
- static void ADC_SetChannelWithThresholdPair1 (ADC_Type *base, uint32_t channelMask)
  Set given channels to apply the threshold pare 1.

Interrupts.

- static void ADC_EnableInterrupts (ADC_Type *base, uint32_t mask)
  Enable interrupts for conversion sequences.
- static void ADC_DisableInterrupts (ADC_Type *base, uint32_t mask)
  Disable interrupts for conversion sequence.
- static void ADC_EnableShresholdCompareInterrupt (ADC_Type *base, uint32_t channel, adc_threshold_interrupt_mode_t mode)
  Enable the interrupt of threshold compare event for each channel.
- static void ADC_EnableThresholdCompareInterrupt (ADC_Type *base, uint32_t channel, adc_threshold_interrupt_mode_t mode)
  Enable the interrupt of threshold compare event for each channel.

Status.

- static uint32_t ADC_GetStatusFlags (ADC_Type *base)
  Get status flags of ADC module.
- static void ADC_ClearStatusFlags (ADC_Type *base, uint32_t mask)
  Clear status flags of ADC module.

5.3 Data Structure Documentation

5.3.1 struct adc_config_t

Data Fields

- adc_clock_mode_t clockMode
  Select the clock mode for ADC converter.
- uint32_t clockDividerNumber
  This field is only available when using kADC_ClockSynchronousMode for "clockMode" field.
- adc_resolution_t resolution
  Select the conversion bits.
- uint32_t sampleTimeNumber
  By default, with value as "0U", the sample period would be 2.5 ADC clocks.

5.3.1.0.0.1 Field Documentation

5.3.1.0.0.1.1 adc_clock_mode_t adc_config_t::clockMode

5.3.1.0.0.1.2 uint32_t adc_config_t::clockDividerNumber

The divider would be plused by 1 based on the value in this field. The available range is in 8 bits.

5.3.1.0.0.1.3 adc_resolution_t adc_config_t::resolution

5.3.1.0.0.1.4 uint32_t adc_config_t::sampleTimeNumber

Then, to plus the "sampleTimeNumber" value here. The available value range is in 3 bits.

5.3.2 struct adc_conv_seq_config_t

Data Fields

- uint32_t channelMask
  Selects which one or more of the ADC channels will be sampled and conver
  sequence is launched.
- uint32_t triggerMask
  Selects which one or more of the available hardware trigger sources will
  conversion sequence to be initiated.
- adc_trigger_polarity_t triggerPolarity
  Select the trigger to lauch conversion sequence.
- bool enableSyncBypass
  To enable this feature allows the hardware trigger input to bypass synchr
  flip-flop stages and therefore shorten the time between the trigger input signal and the start of a conversion.
- bool enableSingleStep
  When enabling this feature, a trigger will launch a single conversion on
  channel in the sequence instead of the default response of launching an entire sequence of conversions.
- adc_seq_interrupt_mode_t interruptMode
  Select the interrpt/DMA trigger mode.

5.3.2.0.0.2 Field Documentation

5.3.2.0.0.2.1 uint32_t adc_conv_seq_config_t::channelMask

The masked channels would be involved in current conversion sequence, beginning with the lowest-order. The available range is in 12-bit.

5.3.2.0.0.2.2 uint32_t adc_conv_seq_config_t::triggerMask

The available range is 6-bit.

5.3.2.0.0.2.3 adc_trigger_polarity_t adc_conv_seq_config_t::triggerPolarity

5.3.2.0.0.2.4 bool adc_conv_seq_config_t::enableSyncBypass

5.3.2.0.0.2.5 bool adc_conv_seq_config_t::enableSingleStep

5.3.2.0.0.2.6 adc_seq_interrupt_mode_t adc_conv_seq_config_t::interruptMode

5.3.3 struct adc_result_info_t

Data Fields

- uint32_t result
  Keep the conversion data value.
- adc_threshold_compare_status_t thresholdCompareStatus
  Keep the threshold compare status.
- adc_threshold_crossing_status_t thresholdCorssingStatus

Keep the threshold crossing status.

- uint32_t channelNumber

  Keep the channel number for this conversion.
- bool overrunFlag

  Keep the status whether the conversion is overrun or not.

5.3.3.0.0.3 Field Documentation

5.3.3.0.0.3.1 uint32_t adc_result_info_t::result

5.3.3.0.0.3.2 adc_threshold_compare_status_t adc_result_info_t::thresholdCompareStatus

5.3.3.0.0.3.3 adc_threshold_crossing_status_t adc_result_info_t::thresholdCorssingStatus

5.3.3.0.0.3.4 uint32_t adc_result_info_t::channelNumber

5.3.3.0.0.3.5 bool adc_result_info_t::overrunFlag

5.4 Macro Definition Documentation

5.4.1 #define FSL_ADC_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))

5.5 Enumeration Type Documentation

5.5.1 enum _adc_status_flags

Enumerator

kADC_ThresholdCompareFlagOnChn0 Threshold comparison event on Channel 0.
kADC_ThresholdCompareFlagOnChn1 Threshold comparison event on Channel 1.
kADC_ThresholdCompareFlagOnChn2 Threshold comparison event on Channel 2.
kADC_ThresholdCompareFlagOnChn3 Threshold comparison event on Channel 3.
kADC_ThresholdCompareFlagOnChn4 Threshold comparison event on Channel 4.
kADC_ThresholdCompareFlagOnChn5 Threshold comparison event on Channel 5.
kADC_ThresholdCompareFlagOnChn6 Threshold comparison event on Channel 6.
kADC_ThresholdCompareFlagOnChn7 Threshold comparison event on Channel 7.
kADC_ThresholdCompareFlagOnChn8 Threshold comparison event on Channel 8.
kADC_ThresholdCompareFlagOnChn9 Threshold comparison event on Channel 9.
kADC_ThresholdCompareFlagOnChn10 Threshold comparison event on Channel 10.
kADC_ThresholdCompareFlagOnChn11 Threshold comparison event on Channel 11.
kADC_OverrunFlagForChn0 Mirror the OVERRUN status flag from the result register for ADC channel 0.
kADC_OverrunFlagForChn1 Mirror the OVERRUN status flag from the result register for ADC channel 1.
kADC_OverrunFlagForChn2 Mirror the OVERRUN status flag from the result register for ADC channel 2.
kADC_OverrunFlagForChn3 Mirror the OVERRUN status flag from the result register for ADC channel 3.

kADC_OverrunFlagForChn4 Mirror the OVERRUN status flag from the result register for ADC channel 4.
kADC_OverrunFlagForChn5 Mirror the OVERRUN status flag from the result register for ADC channel 5.
kADC_OverrunFlagForChn6 Mirror the OVERRUN status flag from the result register for ADC channel 6.
kADC_OverrunFlagForChn7 Mirror the OVERRUN status flag from the result register for ADC channel 7.
kADC_OverrunFlagForChn8 Mirror the OVERRUN status flag from the result register for ADC channel 8.
kADC_OverrunFlagForChn9 Mirror the OVERRUN status flag from the result register for ADC channel 9.
kADC_OverrunFlagForChn10 Mirror the OVERRUN status flag from the result register for ADC channel 10.
kADC_OverrunFlagForChn11 Mirror the OVERRUN status flag from the result register for ADC channel 11.
kADC_GlobalOverrunFlagForSeqA Mirror the glabal OVERRUN status flag for conversion sequence A.
kADC_GlobalOverrunFlagForSeqB Mirror the global OVERRUN status flag for conversion sequence B.
kADC_ConvSeqAInterruptFlag Sequence A interrupt/DMA trigger.
kADC_ConvSeqBInterruptFlag Sequence B interrupt/DMA trigger.
kADC_ThresholdCompareInterruptFlag Threshold comparision interrupt flag.
kADC_OverrunInterruptFlag Overrun interrupt flag.

5.5.2 enum _adc_interrupt_enable

Note

Not all the interrupt options are listed here

Enumerator

kADC_ConvSeqAInterruptEnable Enable interrupt upon completion of each individual conversion in sequence A, or entire sequence.
kADC_OverrunInterruptEnable Enable the detection of an overrun condition on any of the channel data registers will cause an overrun interrupt/DMA trigger.

5.5.3 enum adc_clock_mode_t

Enumerator

kADC_ClockSynchronousMode The ADC clock would be derived from the system clock based on "clockDividerNumber".

kADC_ClockAsynchronousMode The ADC clock would be based on the SYSCON block's divider.

5.5.4 enum adc_resolution_t

Enumerator

kADC_Resolution6bit 6-bit resolution.
kADC_Resolution8bit 8-bit resolution.
kADC_Resolution10bit 10-bit resolution.
kADC_Resolution12bit 12-bit resolution.

5.5.5 enum adc_trigger_polarity_t

Enumerator

kADC_TriggerPolarityNegativeEdge A negative edge launches the conversion sequence on the trigger(s).
kADC_TriggerPolarityPositiveEdge A positive edge launches the conversion sequence on the trigger(s).

5.5.6 enum adc_priority_t

Enumerator

kADC_PriorityLow This sequence would be preempted when another sequence is started.
kADC_PriorityHigh This sequence would preempt other sequence even when it is started.

5.5.7 enum adc_seq_interrupt_mode_t

Enumerator

kADC_InterruptForEachConversion The sequence interrupt/DMA trigger will be set at the end of each individual ADC conversion inside this conversion sequence.
kADC_InterruptForEachSequence The sequence interrupt/DMA trigger will be set when the entire set of this sequence conversions completes.

5.5.8 enum adc_threshold_compare_status_t

Enumerator

kADC_ThresholdCompareInRange LOW threshold <= conversion value <= HIGH threshold.
kADC_ThresholdCompareBelowRange conversion value < LOW threshold.
kADC_ThresholdCompareAboveRange conversion value > HIGH threshold.

5.5.9 enum adc_threshold_crossing_status_t

Enumerator

kADC_ThresholdCrossingNoDetected No threshold Crossing detected.
kADC_ThresholdCrossingDownward Downward Threshold Crossing detected.
kADC_ThresholdCrossingUpward Upward Threshold Crossing Detected.

5.5.10 enum adc_threshold_interrupt_mode_t

Enumerator

kADC_ThresholdInterruptDisabled Threshold comparison interrupt is disabled.
kADC_ThresholdInterruptOnOutside Threshold comparison interrupt is enabled on outside threshold.
kADC_ThresholdInterruptOnCrossing Threshold comparison interrupt is enabled on crossing threshold.

5.5.11 enum adc_inforesult_t

Enumerator

kADC_Resolution12bitInfoResultShift Info result shift of Resolution12bit.
kADC_Resolution10bitInfoResultShift Info result shift of Resolution10bit.
kADC_Resolution8bitInfoResultShift Info result shift of Resolution8bit.
kADC_Resolution6bitInfoResultShift Info result shift of Resolution6bit.

5.5.12 enum adc_tempsensor_common_mode_t

Enumerator

kADC_HighNegativeOffsetAdded Temerature sensor common mode: high negative offset added.

kADC_IntermediateNegativeOffsetAdded Temerature sensor common mode: intermediate negative offset added.
kADC_NoOffsetAdded Temerature sensor common mode: no offset added.
kADC_LowPositiveOffsetAdded Temerature sensor common mode: low positive offset added.

5.5.13 enum adc_second_control_t

Enumerator

kADC_Impedance621Ohm Extand ADC sampling time according to source impedance 1: 0.621 kOhm.
kADC_Impedance55kOhm Extand ADC sampling time according to source impedance 20 (default): 55 kOhm.
kADC_Impedance87kOhm Extand ADC sampling time according to source impedance 31: 87 kOhm.
kADC_NormalFunctionalMode TEST mode: Normal functional mode.
kADC_MultiplexeTestMode TEST mode: Multiplexer test mode.
kADC_ADCInUnityGainMode TEST mode: ADC in unity gain mode.

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 configuration structure, see to adc_config_t.

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 initial configuration structure with an available settings. The default values are:

```
*   config->clockMode = kADC_ClockSynchronousMode;
*   config->clockDividerNumber = 0U;
*   config->resolution = kADC_Resolution12bit;
*   config->enableBypassCalibration = false;
*   config->sampleTimeNumber = 0U;
*
```

Parameters

config	Pointer to configuration structure.

5.6.4 void ADC_EnableTemperatureSensor (ADC_Type ∗ *base,* bool *enable*)

When enabling the internal temperature sensor measurement, the channel 0 would be connected to internal sensor instead of external pin.

Parameters

base	ADC peripheral base address.
enable	Switcher to enable the feature or not.

5.6.5 static void ADC_EnableConvSeqA (ADC_Type ∗ *base,* bool *enable*) [inline], [static]

In order to avoid spuriously triggering the sequence, the trigger to conversion sequence should be ready before the sequence is ready. when the sequence is disabled, the trigger would be ignored. Also, it is suggested to disable the sequence during changing the sequence's setting.

Parameters

base	ADC peripheral base address.
enable	Switcher to enable the feature or not.

5.6.6 void ADC_SetConvSeqAConfig (ADC_Type ∗ *base,* const adc_conv_seq_config_t ∗ *config*)

Parameters

base	ADC peripheral base address.
config	Pointer to configuration structure, see to adc_conv_seq_config_t.

5.6.7 static void ADC_DoSoftwareTriggerConvSeqA (ADC_Type * *base*) [inline], [static]

Parameters

base	ADC peripheral base address.

5.6.8 static void ADC_EnableConvSeqABurstMode (ADC_Type * *base*, bool *enable*) [inline], [static]

Enable the burst mode would cause the conversion sequence to be cntinuously cycled through. Other triggers would be ignored while this mode is enabled. Repeated conversions could be halted by disabling this mode. And the sequence currently in process will be completed before cnversions are terminated. Note that a new sequence could begin just before the burst mode is disabled.

Parameters

base	ADC peripheral base address.
enable	Switcher to enable this feature.

5.6.9 bool ADC_GetConvSeqAGlobalConversionResult (ADC_Type * *base*, adc_result_info_t * *info*)

Parameters

base	ADC peripheral base address.
info	Pointer to information structure, see to adc_result_info_t;

Return values

true	The conversion result is ready.
false	The conversion result is not ready yet.

5.6.10 bool ADC_GetChannelConversionResult (ADC_Type * *base*, uint32_t *channel*, adc_result_info_t * *info*)

Parameters

base	ADC peripheral base address.
channel	The indicated channel number.
info	Pointer to information structure, see to adc_result_info_t;

Return values

true	The conversion result is ready.
false	The conversion result is not ready yet.

5.6.11 static void ADC_SetThresholdPair0 (ADC_Type * *base*, uint32_t *lowValue*, uint32_t *highValue*) [inline], [static]

Parameters

base	ADC peripheral base address.
lowValue	LOW threshold value.
highValue	HIGH threshold value.

5.6.12 static void ADC_SetThresholdPair1 (ADC_Type * *base*, uint32_t *lowValue*, uint32_t *highValue*) [inline], [static]

Parameters

base	ADC peripheral base address.
lowValue	LOW threshold value. The available value is with 12-bit.
highValue	HIGH threshold value. The available value is with 12-bit.

5.6.13 static void ADC_SetChannelWithThresholdPair0 (ADC_Type * *base*, uint32_t *channelMask*) [inline], [static]

Parameters

base	ADC peripheral base address.
channelMask	Indicated channels' mask.

5.6.14 static void ADC_SetChannelWithThresholdPair1 (ADC_Type * *base*, uint32_t *channelMask*) [inline], [static]

Parameters

base	ADC peripheral base address.
channelMask	Indicated channels' mask.

5.6.15 static void ADC_EnableInterrupts (ADC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ADC peripheral base address.
mask	Mask of interrupt mask value for global block except each channal, see to _adc_-interrupt_enable.

5.6.16 static void ADC_DisableInterrupts (ADC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ADC peripheral base address.
mask	Mask of interrupt mask value for global block except each channel, see to _adc_-interrupt_enable.

5.6.17 static void ADC_EnableShresholdCompareInterrupt (ADC_Type * *base*, uint32_t *channel*, adc_threshold_interrupt_mode_t *mode*) [inline], [static]

5.6.18 static void ADC_EnableThresholdCompareInterrupt (ADC_Type * *base*, uint32_t *channel*, adc_threshold_interrupt_mode_t *mode*) [inline], [static]

Parameters

base	ADC peripheral base address.
channel	Channel number.
mode	Interrupt mode for threshold compare event, see to adc_threshold_interrupt_mode_t.

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

Parameters

base	ADC peripheral base address.

Returns

Mask of status flags of module, see to _adc_status_flags.

5.6.20 static void ADC_ClearStatusFlags (ADC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ADC peripheral base address.
mask	Mask of status flags of module, see to _adc_status_flags.

Chapter 6
AES: AES encryption decryption driver

6.1 Overview

The MCUXpresso SDK provides a peripheral driver for the AES module in MCUXpresso SDK devices.

The driver provides blocking synchronous APIs. The AES operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions do not return until an A-ES operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status, as well as plaintext or ciphertext data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

6.2 AES Driver Initialization and Configuration

Clock to the AES module has to be enabled before using the driver API. The function AES_SetKey() has to be used to store encryption key into device registers prior to using other API.

6.3 Comments about API usage in RTOS

AES operations provided by this driver are not re-entrant. Because of this, the application software should ensure the AES module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

6.4 AES Driver Examples

Encrypt plaintext and decrypt it back by AES engine Refer to the driver examples codes located at <SD-K_ROOT>/boards/<BOARD>/driver_examples/aes Encrypts AES using CTR block mode. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/aes Generation of GCM tag only Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver-_examples/aes

Files

- file fsl_aes.h

Driver version

- #define FSL_AES_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  Defines LPC AES driver version 2.0.1.

AES Functional Operation

- status_t AES_SetKey (AES_Type *base, const uint8_t *key, size_t keySize)
  Sets AES key.

- status_t AES_EncryptEcb (AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)
  Encrypts AES using the ECB block mode.
- status_t AES_DecryptEcb (AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)
  Decrypts AES using the ECB block mode.
- status_t AES_EncryptCbc (AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[AES_IV_SIZE])
  Encrypts AES using CBC block mode.
- status_t AES_DecryptCbc (AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[AES_IV_SIZE])
  Decrypts AES using CBC block mode.
- status_t AES_EncryptCfb (AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[AES_IV_SIZE])
  Encrypts AES using CFB block mode.
- status_t AES_DecryptCfb (AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[AES_IV_SIZE])
  Decrypts AES using CFB block mode.
- status_t AES_EncryptOfb (AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[AES_IV_SIZE])
  Encrypts AES using OFB block mode.
- status_t AES_DecryptOfb (AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[AES_IV_SIZE])
  Decrypts AES using OFB block mode.
- status_t AES_CryptCtr (AES_Type *base, const uint8_t *input, uint8_t *output, size_t size, uint8_t counter[AES_BLOCK_SIZE], uint8_t counterlast[AES_BLOCK_SIZE], size_t *szLeft)
  Encrypts or decrypts AES using CTR block mode.
- status_t AES_EncryptTagGcm (AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, uint8_t *tag, size_t tagSize)
  Encrypts AES and tags using GCM block mode.
- status_t AES_DecryptTagGcm (AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *tag, size_t tagSize)
  Decrypts AES and authenticates using GCM block mode.
- void **AES_Init** (AES_Type *base)
- void **AES_Deinit** (AES_Type *base)
- #define AES_BLOCK_SIZE 16
  AES block size in bytes.
- #define AES_IV_SIZE 16
  AES Input Vector size in bytes.

6.5 Macro Definition Documentation

6.5.1 #define FSL_AES_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Change log:

- Version 2.0.0

  - initial version
- Version 2.0.1
  - GCM constant time tag comparison

6.6 Function Documentation

6.6.1 status_t AES_SetKey (AES_Type * *base*, const uint8_t * *key*, size_t *keySize*)

Sets AES key.

Parameters

base	AES peripheral base address
key	Input key to use for encryption or decryption
keySize	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from Set Key operation

6.6.2 status_t AES_EncryptEcb (AES_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size*)

Encrypts AES using the ECB block mode.

Parameters

	base	AES peripheral base address
	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

6.6.3 status_t AES_DecryptEcb (AES_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*)

Decrypts AES using the ECB block mode.

Parameters

	base	AES peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plain text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from decrypt operation

6.6.4 status_t AES_EncryptCbc (AES_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size*, const uint8_t *iv[AES_IV_SIZE]*)

Parameters

	base	AES peripheral base address
	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

6.6.5 status_t AES_DecryptCbc (AES_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*, const uint8_t *iv[AES_IV_SIZE]*)

Parameters

	base	AES peripheral base address

	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

6.6.6 status_t AES_EncryptCfb (AES_Type ∗ *base*, const uint8_t ∗ *plaintext*, uint8_t ∗ *ciphertext*, size_t *size*, const uint8_t *iv[AES_IV_SIZE]*)

Parameters

	base	AES peripheral base address
	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 be used as the first input block.

Returns

Status from encrypt operation

6.6.7 status_t AES_DecryptCfb (AES_Type ∗ *base*, const uint8_t ∗ *ciphertext*, uint8_t ∗ *plaintext*, size_t *size*, const uint8_t *iv[AES_IV_SIZE]*)

Parameters

	base	AES peripheral base address
	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 be used as the first input block.

Returns

Status from decrypt operation

6.6.8 status_t AES_EncryptOfb (AES_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size*, const uint8_t *iv[AES_IV_SIZE]*)

Parameters

	base	AES peripheral base address
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes.
	iv	Input Initial vector to be used as the first input block.

Returns

Status from encrypt operation

6.6.9 status_t AES_DecryptOfb (AES_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*, const uint8_t *iv[AES_IV_SIZE]*)

Parameters

	base	AES peripheral base address
	ciphertext	Input cipher text to decrypt
out	*plaintext*	Output plain text

	size	Size of input and output data in bytes.
	iv	Input Initial vector to be used as the first input block.

Returns

Status from decrypt operation

6.6.10 status_t AES_CryptCtr (AES_Type ∗ *base,* const uint8_t ∗ *input,* uint8_t ∗ *output,* size_t *size,* uint8_t *counter[AES_BLOCK_SIZE],* uint8_t *counterlast[AES_BLOCK_SIZE],* size_t ∗ *szLeft*)

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

	base	AES peripheral base address
	input	Input data for CTR block mode
`out`	*output*	Output data for CTR block mode
	size	Size of input and output data in bytes
`in,out`	*counter*	Input counter (updates on return)
`out`	*counterlast*	Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
`out`	*szLeft*	Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from crypt operation

6.6.11 status_t AES_EncryptTagGcm (AES_Type ∗ *base,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* size_t *size,* const uint8_t ∗ *iv,* size_t *ivSize,* const uint8_t ∗ *aad,* size_t *aadSize,* uint8_t ∗ *tag,* size_t *tagSize*)

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the ’tag’ field.

Parameters

	base	AES peripheral base address
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text.
	size	Size of input and output data in bytes
	iv	Input initial vector
	ivSize	Size of the IV
	aad	Input additional authentication data
	aadSize	Input size in bytes of AAD
out	*tag*	Output hash tag. Set to NULL to skip tag processing.
	tagSize	Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from encrypt operation

6.6.12 status_t AES_DecryptTagGcm (AES_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*, const uint8_t * *iv*, size_t *ivSize*, const uint8_t * *aad*, size_t *aadSize*, const uint8_t * *tag*, size_t *tagSize*)

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

Parameters

	base	AES peripheral base address
	ciphertext	Input cipher text to decrypt
out	*plaintext*	Output plain text.
	size	Size of input and output data in bytes
	iv	Input initial vector

	ivSize	Size of the IV
	aad	Input additional authentication data
	aadSize	Input size in bytes of AAD
	tag	Input hash tag to compare. Set to NULL to skip tag processing.
	tagSize	Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from decrypt operation

Chapter 7
Clock: Clock driver

7.1 Overview

The MCUXpresso SDK provides a clock driver for MCUXpresso SDK devices.

7.2 Function groups

Clock driver provides these functions:

- Functions to obtain frequency of specified clock
- Functions to configure the clock selection muxes.
- Functions to setup peripheral clock dividers
- Functions to enable specific AHB clock channel

7.2.1 SYSCON Clock frequency functions

SYSCON clock module provides clocks, such as ADCCLK, DMICCLK, FXCOMCLK,WDTOSC, R-TCOSC and SYSPLL. The functions CLOCK_EnableClock() and CLOCK_DisableClock() enables and disables the various clocks. The SYSCON clock driver provides functions to get the frequency of clocks, such as CLOCK_GetFreq(),

7.2.2 SYSCON clock Selection Muxes

The SYSCON clock driver provides the function to configure the clock selected. The function CLOCK_-AttachClk() is implemented for this. The function selects the clock source for a particular peripheral like MAINCLK, DMIC, FLEXCOMM, USB, ADC and PLL.

7.2.3 SYSCON clock dividers

The SYSCON clock module provides the function to setup the peripheral clock dividers. The function CLOCK_SetClkDiv() configures the CLKDIV registers for various periperals like USB, DMIC, I2S, SY-STICK, AHB, ADC and also for CLKOUT and TRACE functions.

Files

- file fsl_clock.h

Data Structures

- struct ClockCapacitanceCompensation_t
  Board specific constant capacitance characteristics Should be supplied by board manufacturer for best performance. More...

Macros

- #define FLEXCOMM_CLOCKS
  Clock ip name array for FLEXCOMM.
- #define CTIMER_CLOCKS
  Clock ip name array for CTIMER.
- #define GINT_CLOCKS
  Clock ip name array for GINT.
- #define WWDT_CLOCKS
  Clock ip name array for WWDT.
- #define DMIC_CLOCKS
  Clock ip name array for DMIC.
- #define ADC_CLOCKS
  Clock ip name array for ADC.
- #define SPIFI_CLOCKS
  Clock ip name array for SPIFI.
- #define GPIO_CLOCKS
  Clock ip name array for GPIO.
- #define DMA_CLOCKS
  Clock ip name array for DMA.

Enumerations

- enum CHIP_SYSCON_MAINCLKSRC_T {
  SYSCON_MAINCLKSRC_FRO12M,
  SYSCON_MAINCLKSRC_OSC32K,
  SYSCON_MAINCLKSRC_XTAL32M,
  SYSCON_MAINCLKSRC_FRO32M,
  SYSCON_MAINCLKSRC_FRO48M,
  SYSCON_MAINCLKSRC_EXT,
  SYSCON_MAINCLKSRC_FRO1M }
  Clock sources for main system clock.
- enum CHIP_SYSCON_FRGCLKSRC_T {
  SYSCON_FRGCLKSRC_MAINCLK,
  SYSCON_FRGCLKSRC_OSC32M,
  SYSCON_FRGCLKSRC_FRO48MHZ,
  SYSCON_FRGCLKSRC_NONE }
  Fractional Divider clock sources.
- enum clock_name_t {
  kCLOCK_Rom = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_ROM_SHIFT),
  kCLOCK_Sram0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_S-

RAM_CTRL0_SHIFT),
kCLOCK_Sram1 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_S-RAM_CTRL1_SHIFT),
kCLOCK_Flash = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_FL-ASH_SHIFT),
kCLOCK_Spifi = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_SPI-FI_SHIFT),
kCLOCK_InputMux = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0-_MUX_SHIFT),
kCLOCK_Iocon = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_IO-CON_SHIFT),
kCLOCK_Gpio0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_G-PIO_SHIFT),
kCLOCK_Pint = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_PIN-T_SHIFT) ,
kCLOCK_Dma = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_DM-A_SHIFT),
kCLOCK_Iso7816 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_I-SO7816_SHIFT),
kCLOCK_WdtOsc = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_-WWDT_SHIFT),
kCLOCK_Rtc = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_RTC-_SHIFT),
kCLOCK_AnaInt,
kCLOCK_WakeTmr,
kCLOCK_Adc0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_AD-C_SHIFT),
kCLOCK_Efuse = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_EF-USE_SHIFT),
kCLOCK_FlexComm0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-L1_USART0_SHIFT),
kCLOCK_FlexComm1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-L1_USART1_SHIFT),
kCLOCK_FlexComm2 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-L1_I2C0_SHIFT),
kCLOCK_FlexComm3 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-L1_I2C1_SHIFT),
kCLOCK_FlexComm4 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-L1_SPI0_SHIFT),
kCLOCK_FlexComm5 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-L1_SPI1_SHIFT),
kCLOCK_Ir = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_IR_SH-IFT),
kCLOCK_Pwm = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_PW-

M_SHIFT),
kCLOCK_Rng = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_RN-G_SHIFT),
kCLOCK_FlexComm6 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-L1_I2C2_SHIFT),
kCLOCK_Usart0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_U-SART0_SHIFT),
kCLOCK_Usart1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_U-SART1_SHIFT),
kCLOCK_I2c0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_I2-C0_SHIFT),
kCLOCK_I2c1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_I2-C1_SHIFT),
kCLOCK_Spi0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_SP-I0_SHIFT),
kCLOCK_Spi1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_SP-I1_SHIFT),
kCLOCK_I2c2 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_I2-C2_SHIFT),
kCLOCK_Modem = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_-MODEM_MASTER_SHIFT),
kCLOCK_Aes = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_AES-_SHIFT),
kCLOCK_Rfp = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_RFP-_SHIFT),
kCLOCK_DMic = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_D-MIC_SHIFT),
kCLOCK_Sha0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_HA-

SH_SHIFT),
kCLOCK_Timer0 = CLK_GATE_DEFINE(ASYNC_CLK_CTRL0, 1),
kCLOCK_Timer1 = CLK_GATE_DEFINE(ASYNC_CLK_CTRL0, 2),
kCLOCK_MainClk = (1 << 16),
kCLOCK_CoreSysClk,
kCLOCK_BusClk,
kCLOCK_Xtal32k,
kCLOCK_Xtal32M,
kCLOCK_Fro32k,
kCLOCK_Fro1M,
kCLOCK_Fro12M,
kCLOCK_Fro32M,
kCLOCK_Fro48M,
kCLOCK_Fro64M,
kCLOCK_ExtClk,
kCLOCK_WdtClk,
kCLOCK_Frg,
kCLOCK_ClkOut,
kCLOCK_Fmeas,
kCLOCK_Sha = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_HAS-H_SHIFT) }
Clock name definition.
- enum clock_sel_ofst_t {
CM_MAINCLKSEL = REG_OFST(SYSCON, MAINCLKSEL),
CM_OSC32CLKSEL = REG_OFST(SYSCON, OSC32CLKSEL),
CM_CLKOUTCLKSEL = REG_OFST(SYSCON, CLKOUTSEL),
CM_SPIFICLKSEL = REG_OFST(SYSCON, SPIFICLKSEL),
CM_ADCCLKSEL = REG_OFST(SYSCON, ADCCLKSEL),
CM_USARTCLKSEL = REG_OFST(SYSCON, USARTCLKSEL),
CM_I2CCLKSEL = REG_OFST(SYSCON, I2CCLKSEL),
CM_SPICLKSEL = REG_OFST(SYSCON, SPICLKSEL),
CM_IRCLKSEL = REG_OFST(SYSCON, IRCLKSEL),
CM_PWMCLKSEL = REG_OFST(SYSCON, PWMCLKSEL),
CM_WDTCLKSEL = REG_OFST(SYSCON, WDTCLKSEL),
CM_MODEMCLKSEL = REG_OFST(SYSCON, MODEMCLKSEL),
CM_FRGCLKSEL = REG_OFST(SYSCON, FRGCLKSEL),
CM_DMICLKSEL = REG_OFST(SYSCON, DMICCLKSEL),
CM_WKTCLKSEL = REG_OFST(SYSCON, WKTCLKSEL) }
Clock source selector definition.
- enum clock_attach_id_t {

```
kFRO12M_to_MAIN_CLK = MUX_A(CM_MAINCLKSEL, 0),
kOSC32K_to_MAIN_CLK = MUX_A(CM_MAINCLKSEL, 1),
kXTAL32M_to_MAIN_CLK = MUX_A(CM_MAINCLKSEL, 2),
kFRO32M_to_MAIN_CLK = MUX_A(CM_MAINCLKSEL, 3),
kFRO48M_to_MAIN_CLK = MUX_A(CM_MAINCLKSEL, 4),
kEXT_CLK_to_MAIN_CLK = MUX_A(CM_MAINCLKSEL, 5),
kFROM1M_to_MAIN_CLK = MUX_A(CM_MAINCLKSEL, 6),
kFRO32M_to_OSC32M_CLK = MUX_A(CM_OSC32CLKSEL, 0),
kXTAL32M_to_OSC32M_CLK = MUX_A(CM_OSC32CLKSEL, 1),
kFRO32K_to_OSC32K_CLK = MUX_A(CM_OSC32CLKSEL, 2),
kXTAL32K_to_OSC32K_CLK = MUX_A(CM_OSC32CLKSEL, 3),
kMAIN_CLK_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 0),
kXTAL32K_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 1),
kFRO32K_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 2),
kXTAL32M_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 3),
kDCDC_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 4),
kFRO48M_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 5),
kFRO1M_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 6),
kNONE_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 7),
kMAIN_CLK_to_SPIFI = MUX_A(CM_SPIFICLKSEL, 0),
kXTAL32M_to_SPIFI = MUX_A(CM_SPIFICLKSEL, 1),
kFRO64M_to_SPIFI = MUX_A(CM_SPIFICLKSEL, 2),
kFRO48M_to_SPIFI = MUX_A(CM_SPIFICLKSEL, 3),
kXTAL32M_to_ADC_CLK = MUX_A(CM_ADCCLKSEL, 0),
kFRO12M_to_ADC_CLK = MUX_A(CM_ADCCLKSEL, 1),
kNONE_to_ADC_CLK = MUX_A(CM_ADCCLKSEL, 2),
kOSC32M_to_USART_CLK = MUX_A(CM_USARTCLKSEL, 0),
kFRO48M_to_USART_CLK = MUX_A(CM_USARTCLKSEL, 1),
kFRG_CLK_to_USART_CLK = MUX_A(CM_USARTCLKSEL, 2),
kNONE_to_USART_CLK = MUX_A(CM_USARTCLKSEL, 3),
kOSC32M_to_I2C_CLK = MUX_A(CM_I2CCLKSEL, 0),
kFRO48M_to_I2C_CLK = MUX_A(CM_I2CCLKSEL, 1),
kNONE_to_I2C_CLK = MUX_A(CM_I2CCLKSEL, 2),
kOSC32M_to_SPI_CLK = MUX_A(CM_SPICLKSEL, 0),
kFRO48M_to_SPI_CLK = MUX_A(CM_SPICLKSEL, 1),
kNONE_to_SPI_CLK = MUX_A(CM_SPICLKSEL, 2),
kOSC32M_to_IR_CLK = MUX_A(CM_IRCLKSEL, 0),
kFRO48M_to_IR_CLK = MUX_A(CM_IRCLKSEL, 1),
kNONE_to_IR_CLK = MUX_A(CM_IRCLKSEL, 2),
kOSC32M_to_PWM_CLK = MUX_A(CM_PWMCLKSEL, 0),
kFRO48M_to_PWM_CLK = MUX_A(CM_PWMCLKSEL, 1),
kNONE_to_PWM_CLK = MUX_A(CM_PWMCLKSEL, 2),
kOSC32M_to_WDT_CLK = MUX_A(CM_WDTCLKSEL, 0),
kOSC32K_to_WDT_CLK = MUX_A(CM_WDTCLKSEL, 1),
kFRO1M_to_WDT_CLK = MUX_A(CM_WDTCLKSEL, 2),
kMAIN_CLK_to_FRG_CLK = MUX_A(CM_FRGCLKSEL, 0),
kOSC32M_to_FRG_CLK = MUX_A(CM_FRGCLKSEL, 1),
kFRO48M_to_FRG_CLK = MUX_A(CM_FRGCLKSEL, 2),
kNONE_to_FRG_CLK = MUX_A(CM_FRGCLKSEL, 3),
```

kFRO48M_to_ASYNC_APB = MUX_A(CM_ASYNCAPB, 3) }
Clock attach definition.

- enum clock_div_name_t
Clock divider definition.
- enum main_clock_src_t {
kCLOCK_MainFro12M = 0,
kCLOCK_MainOsc32k = 1,
kCLOCK_MainXtal32M = 2,
kCLOCK_MainFro32M = 3,
kCLOCK_MainFro48M = 4,
kCLOCK_MainExtClk = 5,
kCLOCK_MainFro1M = 6 }
Clock source selections for the Main Clock.
- enum clkout_clock_src_t {
kCLOCK_ClkoutMainClk = 0,
kCLOCK_ClkoutXtal32k = 1,
kCLOCK_ClkoutFro32k = 2,
kCLOCK_ClkoutXtal32M = 3,
kCLOCK_ClkoutDcDcTest = 4,
kCLOCK_ClkoutFro48M = 5,
kCLOCK_ClkoutFro1M = 6,
kCLOCK_ClkoutNoClock = 7 }
Clock source selections for CLKOUT.
- enum wdt_clock_src_t {
kCLOCK_WdtOsc32MClk = 0,
kCLOCK_WdtOsc32kClk = 1,
kCLOCK_WdtFro1M = 2,
kCLOCK_WdtNoClock = 3 }
Clock source definition for Watchdog timer.
- enum frg_clock_src_t {
kCLOCK_FrgMainClk = 0,
kCLOCK_FrgOsc32MClk = 1,
kCLOCK_FrgFro48M = 2,
kCLOCK_FrgNoClock = 3 }
Clock source definition for fractional divider.
- enum apb_clock_src_t {
kCLOCK_ApbMainClk = 0,
kCLOCK_ApbXtal32M = 1,
kCLOCK_ApbFro32M = 2,
kCLOCK_ApbFro48M = 3 }
Clock source definition for the APB.
- enum fmeas_clock_src_t {

kCLOCK_fmeasClkIn = 0,
kCLOCK_fmeasXtal32Mhz = 1,
kCLOCK_fmeasFRO1Mhz = 2,
kCLOCK_fmeasXtal32kHz = 3,
kCLOCK_fmeasMainClock = 4,
kCLOCK_fmeasGPIO_0_4 = 5,
kCLOCK_fmeasGPIO_0_20 = 6,
kCLOCK_fmeasGPIO_0_16 = 7,
kCLOCK_fmeasGPIO_0_15 = 8 }
Clock source definition for frequency measure.

- enum spifi_clock_src_t {
kCLOCK_SpifiMainClk = 0,
kCLOCK_SpifiXtal32M = 1,
kCLOCK_SpifiFro64M = 2,
kCLOCK_SpifiFro48M = 3,
kCLOCK_SpifiNoClock = 4 }
Clock source selection for SPIFI.
- enum adc_clock_src_t {
kCLOCK_AdcXtal32M = 0,
kCLOCK_AdcFro12M = 1,
kCLOCK_AdcNoClock = 2 }
Clock definition for ADC.
- enum pwm_clock_source_t {
kCLOCK_PWMOsc32Mclk = 0x0,
kCLOCK_PWMFro48Mclk = 0x1,
kCLOCK_PWMNoClkSel = 0x2,
kCLOCK_PWMTestClk = 0x3 }
PWM Clock source selection values.
- enum Fro_ClkSel_t {
FRO12M_ENA = (1 << 0),
FRO32M_ENA = (1 << 1),
FRO48M_ENA = (1 << 2),
FRO64M_ENA = (1 << 3),
FRO96M_ENA = (1 << 4) }
FRO clock selection values.

Functions

- uint32_t CLOCK_GetFreq (clock_name_t clock)
Obtains frequency of specified clock.
- void CLOCK_AttachClk (clock_attach_id_t connection)
Selects clock source using <name>SEL register in syscon.
- void CLOCK_SetClkDiv (clock_div_name_t div_name, uint32_t divided_by_value, bool reset)
Selects clock divider using <name>DIV register in syscon.
- void CLOCK_EnableClock (clock_ip_name_t clk)
Enables specific AHB clock channel.
- void CLOCK_DisableClock (clock_ip_name_t clk)

Disables specific AHB clock channel.
- bool CLOCK_IsClockEnable (clock_ip_name_t clk)
  Check if clock is enabled.
- uint32_t CLOCK_GetApbCLkFreq (void)
  Obtains frequency of APB Bus clock.
- uint32_t CLOCK_GetSpifiClkFreq (void)
  Return Frequency of Spifi Clock.
- void CLOCK_uDelay (uint32_t delayUs)
  Delay execution by busy waiting.
- void CLOCK_XtalBasicTrim (void)
  Sets default trim values for 32MHz XTAL.
- void CLOCK_Xtal32M_Trim (int32_t XO_32M_OSC_CAP_Delta_x1000, const ClockCapacitanceCompensation_t ∗capa_charac)
  Sets board-specific trim values for 32MHz XTAL.
- void CLOCK_Xtal32k_Trim (int32_t XO_32k_OSC_CAP_Delta_x1000, const ClockCapacitanceCompensation_t ∗capa_charac)
  Sets board-specific trim values for 32kHz XTAL.
- void CLOCK_SetXtal32M_LDO (void)
  Enables and sets LDO for 32MHz XTAL.
- void CLOCK_Xtal32M_WaitUntilStable (uint32_t u32AdditionalWait_us)
  Waits for 32MHz XTAL to stabilise.

7.3 Data Structure Documentation

7.3.1 struct ClockCapacitanceCompensation_t

Capacitances are expressed in hundreds of pF

7.4 Macro Definition Documentation

7.4.1 #define FLEXCOMM_CLOCKS

Value:

```
{
        \
          kCLOCK_Usart0, kCLOCK_Usart1,
        kCLOCK_I2c0, kCLOCK_I2c1, kCLOCK_Spi0,
        kCLOCK_Spi1, kCLOCK_I2c2 \
    }
```

7.4.2 #define CTIMER_CLOCKS

Value:

```
{                                   \
          kCLOCK_Timer0, kCLOCK_Timer1 \
    }
```

7.4.3 #define GINT_CLOCKS

Value:

```
{               \
        kCLOCK_Gint \
    }
```

7.4.4 #define WWDT_CLOCKS

Value:

```
{                 \
        kCLOCK_WdtOsc \
    }
```

7.4.5 #define DMIC_CLOCKS

Value:

```
{               \
        kCLOCK_DMic \
    }
```

7.4.6 #define ADC_CLOCKS

Value:

```
{               \
        kCLOCK_Adc0 \
    }
```

7.4.7 #define SPIFI_CLOCKS

Value:

```
{                \
        kCLOCK_Spifi \
    }
```

7.4.8 #define GPIO_CLOCKS

Value:

```
{              \
        kCLOCK_Gpio0 \
    }
```

7.4.9 #define DMA_CLOCKS

Value:

```
{            \
        kCLOCK_Dma \
    }
```

7.5 Enumeration Type Documentation

7.5.1 enum CHIP_SYSCON_MAINCLKSRC_T

Enumerator

SYSCON_MAINCLKSRC_FRO12M FRO 12MHz.
SYSCON_MAINCLKSRC_OSC32K OSC 32kHz.
SYSCON_MAINCLKSRC_XTAL32M XTAL 32MHz.
SYSCON_MAINCLKSRC_FRO32M FRO 32MHz.
SYSCON_MAINCLKSRC_FRO48M FRO 48MHz.
SYSCON_MAINCLKSRC_EXT External clock.
SYSCON_MAINCLKSRC_FRO1M FRO 1MHz.

7.5.2 enum CHIP_SYSCON_FRGCLKSRC_T

Enumerator

SYSCON_FRGCLKSRC_MAINCLK Main Clock.
SYSCON_FRGCLKSRC_OSC32M 32MHz Clock (XTAL or FRO)
SYSCON_FRGCLKSRC_FRO48MHZ FRO 48-MHz.
SYSCON_FRGCLKSRC_NONE FRO 48-MHz.

7.5.3 enum clock_name_t

Enumerator

kCLOCK_Rom ROM clock.
kCLOCK_Sram0 SRAM0 clock.
kCLOCK_Sram1 SRAM1 clock.
kCLOCK_Flash Flash clock.
kCLOCK_Spifi SPIFI clock.
kCLOCK_InputMux InputMux clock.
kCLOCK_Iocon IOCON clock.
kCLOCK_Gpio0 GPIO0 clock.
kCLOCK_Pint PINT clock.
kCLOCK_Dma DMA clock.
kCLOCK_Iso7816 ISO7816 clock.
kCLOCK_WdtOsc WDTOSC clock.
kCLOCK_Rtc RTC clock.
kCLOCK_AnaInt Analog Interrupt Control module clock.
kCLOCK_WakeTmr Wake up Timers clock.
kCLOCK_Adc0 ADC0 clock.
kCLOCK_Efuse EFuse clock.
kCLOCK_FlexComm0 FlexComm0 clock.
kCLOCK_FlexComm1 FlexComm1 clock.
kCLOCK_FlexComm2 FlexComm2 clock.
kCLOCK_FlexComm3 FlexComm3 clock.
kCLOCK_FlexComm4 FlexComm4 clock.
kCLOCK_FlexComm5 FlexComm5 clock.
kCLOCK_Ir Infra Red clock.
kCLOCK_Pwm PWM clock.
kCLOCK_Rng RNG clock.
kCLOCK_FlexComm6 FlexComm6 clock.
kCLOCK_Usart0 USART0 clock.
kCLOCK_Usart1 USART1 clock.
kCLOCK_I2c0 I2C0 clock.
kCLOCK_I2c1 I2C1 clock.
kCLOCK_Spi0 SPI0 clock.
kCLOCK_Spi1 SPI1 clock.
kCLOCK_I2c2 I2C2 clock.
kCLOCK_Modem MODEM clock.
kCLOCK_Aes AES clock.
kCLOCK_Rfp RFP clock.
kCLOCK_DMic DMIC clock.
kCLOCK_Sha0 SHA0 clock.
kCLOCK_Timer0 Timer0 clock.
kCLOCK_Timer1 Timer1 clock.

kCLOCK_MainClk MAIN_CLK.
kCLOCK_CoreSysClk Core/system clock.
kCLOCK_BusClk AHB bus clock.
kCLOCK_Xtal32k 32kHz crystal oscillator
kCLOCK_Xtal32M 32MHz crystal oscillator
kCLOCK_Fro32k 32kHz free running oscillator
kCLOCK_Fro1M 1MHz Free Running Oscillator
kCLOCK_Fro12M 12MHz Free Running Oscillator
kCLOCK_Fro32M 32MHz Free Running Oscillator
kCLOCK_Fro48M 48MHz Free Running Oscillator
kCLOCK_Fro64M 64Mhz Free Running Oscillator
kCLOCK_ExtClk External clock.
kCLOCK_WdtClk Watchdog clock.
kCLOCK_Frg Fractional divider.
kCLOCK_ClkOut Clock out.
kCLOCK_Fmeas FMEAS clock.
kCLOCK_Sha Hash clock.

7.5.4 enum clock_sel_ofst_t

Enumerator

CM_MAINCLKSEL Clock source selector of Main clock source.
CM_OSC32CLKSEL Clock source selector of OSC32KCLK and OSC32MCLK.
CM_CLKOUTCLKSEL Clock source selector of CLKOUT.
CM_SPIFICLKSEL Clock source selector of SPIFI.
CM_ADCCLKSEL Clock source selector of ADC.
CM_USARTCLKSEL Clock source selector of USART0 & 1.
CM_I2CCLKSEL Clock source selector of I2C0, 1 and 2.
CM_SPICLKSEL Clock source selector of SPI0 & 1.
CM_IRCLKSEL Clock source selector of Infra Red.
CM_PWMCLKSEL Clock source selector of PWM.
CM_WDTCLKSEL Clock source selector of Watchdog Timer.
CM_MODEMCLKSEL Clock source selector of Modem.
CM_FRGCLKSEL Clock source selector of Fractional Rate Generator (FRG)
CM_DMICLKSEL Clock source selector of Digital microphone (DMIC)
CM_WKTCLKSEL Clock source selector of Wake-up Timer.

7.5.5 enum clock_attach_id_t

Enumerator

kFRO12M_to_MAIN_CLK Select FRO 12M for main clock.

kOSC32K_to_MAIN_CLK Select OSC 32K for main clock.
kXTAL32M_to_MAIN_CLK Select XTAL 32M for main clock.
kFRO32M_to_MAIN_CLK Select FRO 32M for main clock.
kFRO48M_to_MAIN_CLK Select FRO 48M for main clock.
kEXT_CLK_to_MAIN_CLK Select external clock for main clock.
kFROM1M_to_MAIN_CLK Select FRO 1M for main clock.
kFRO32M_to_OSC32M_CLK Select FRO 32M for OSC32KCLK and OSC32MCLK.
kXTAL32M_to_OSC32M_CLK Select XTAL 32M for OSC32KCLK and OSC32MCLK.
kFRO32K_to_OSC32K_CLK Select FRO 32K for OSC32KCLK and OSC32MCLK.
kXTAL32K_to_OSC32K_CLK Select XTAL 32K for OSC32KCLK and OSC32MCLK.
kMAIN_CLK_to_CLKOUT Select main clock for CLKOUT.
kXTAL32K_to_CLKOUT Select XTAL 32K for CLKOUT.
kFRO32K_to_CLKOUT Select FRO 32K for CLKOUT.
kXTAL32M_to_CLKOUT Select XTAL 32M for CLKOUT.
kDCDC_to_CLKOUT Select DCDC for CLKOUT.
kFRO48M_to_CLKOUT Select FRO 48M for CLKOUT.
kFRO1M_to_CLKOUT Select FRO 1M for CLKOUT.
kNONE_to_CLKOUT No clock for CLKOUT.
kMAIN_CLK_to_SPIFI Select main clock for SPIFI.
kXTAL32M_to_SPIFI Select XTAL 32M for SPIFI.
kFRO64M_to_SPIFI Select FRO 64M for SPIFI.
kFRO48M_to_SPIFI Select FRO 48M for SPIFI.
kXTAL32M_to_ADC_CLK Select XTAL 32M for ADC.
kFRO12M_to_ADC_CLK Select FRO 12M for ADC.
kNONE_to_ADC_CLK No clock for ADC.
kOSC32M_to_USART_CLK Select OSC 32M for USART0 & 1.
kFRO48M_to_USART_CLK Select FRO 48M for USART0 & 1.
kFRG_CLK_to_USART_CLK Select FRG clock for USART0 & 1.
kNONE_to_USART_CLK No clock for USART0 & 1.
kOSC32M_to_I2C_CLK Select OSC 32M for I2C0, 1 and 2.
kFRO48M_to_I2C_CLK Select FRO 48M for I2C0, 1 and 2.
kNONE_to_I2C_CLK No clock for I2C0, 1 and 2.
kOSC32M_to_SPI_CLK Select OSC 32M for SPI0 & 1.
kFRO48M_to_SPI_CLK Select FRO 48M for SPI0 & 1.
kNONE_to_SPI_CLK No clock for SPI0 & 1.
kOSC32M_to_IR_CLK Select OSC 32M for Infra Red.
kFRO48M_to_IR_CLK Select FRO 48M for Infra Red.
kNONE_to_IR_CLK No clock for Infra Red.
kOSC32M_to_PWM_CLK Select OSC 32M for PWM.
kFRO48M_to_PWM_CLK Select FRO 48M for PWM.
kNONE_to_PWM_CLK No clock for PWM.
kOSC32M_to_WDT_CLK Select OSC 32M for Watchdog Timer.
kOSC32K_to_WDT_CLK Select FRO 32K for Watchdog Timer.
kFRO1M_to_WDT_CLK Select FRO 1M for Watchdog Timer.
kMAIN_CLK_to_FRG_CLK Select main clock for FRG.

kOSC32M_to_FRG_CLK Select OSC 32M for FRG.
kFRO48M_to_FRG_CLK Select FRO 48M for FRG.
kNONE_to_FRG_CLK No clock for FRG.
kMAIN_CLK_to_DMI_CLK Select main clock for DMIC.
kOSC32K_to_DMI_CLK Select OSC 32K for DMIC.
kFRO48M_to_DMI_CLK Select FRO 48M for DMIC.
kMCLK_to_DMI_CLK Select external clock for DMIC.
kFRO1M_to_DMI_CLK Select FRO 1M for DMIC.
kFRO12M_to_DMI_CLK Select FRO 12M for DMIC.
kNONE_to_DMI_CLK No clock for DMIC.
kOSC32K_to_WKT_CLK Select OSC 32K for WKT.
kNONE_to_WKT_CLK No clock for WKT.
kXTAL32M_DIV2_to_ZIGBEE_CLK Select XTAL 32M for ZIGBEE.
kNONE_to_ZIGBEE_CLK No clock for ZIGBEE.
kMAIN_CLK_to_ASYNC_APB Select main clock for Asynchronous APB.
kXTAL32M_to_ASYNC_APB Select XTAL 32M for Asynchronous APB.
kFRO32M_to_ASYNC_APB Select FRO 32M for Asynchronous APB.
kFRO48M_to_ASYNC_APB Select FRO 48M for Asynchronous APB.

7.5.6 enum main_clock_src_t

Enumerator

kCLOCK_MainFro12M FRO 12M for main clock.
kCLOCK_MainOsc32k OSC 32K for main clock.
kCLOCK_MainXtal32M XTAL 32M for main clock.
kCLOCK_MainFro32M FRO 32M for main clock.
kCLOCK_MainFro48M FRO 48M for main clock.
kCLOCK_MainExtClk External clock for main clock.
kCLOCK_MainFro1M FRO 1M for main clock.

7.5.7 enum clkout_clock_src_t

Enumerator

kCLOCK_ClkoutMainClk CPU & System Bus clock for CLKOUT.
kCLOCK_ClkoutXtal32k XTAL 32K for CLKOUT.
kCLOCK_ClkoutFro32k FRO 32K for CLKOUT.
kCLOCK_ClkoutXtal32M XTAL 32M for CLKOUT.
kCLOCK_ClkoutDcDcTest DCDC Test for CLKOUT.
kCLOCK_ClkoutFro48M FRO 48M for CLKOUT.
kCLOCK_ClkoutFro1M FRO 1M for CLKOUT.
kCLOCK_ClkoutNoClock No clock for CLKOUT.

7.5.8 enum wdt_clock_src_t

Enumerator

kCLOCK_WdtOsc32MClk OSC 32M for WDT.
kCLOCK_WdtOsc32kClk OSC 32K for WDT.
kCLOCK_WdtFro1M FRO 1M for WDT.
kCLOCK_WdtNoClock No clock for WDT.

7.5.9 enum frg_clock_src_t

Enumerator

kCLOCK_FrgMainClk CPU & System Bus clock for FRG.
kCLOCK_FrgOsc32MClk OSC 32M clock for FRG.
kCLOCK_FrgFro48M FRO 48M for FRG.
kCLOCK_FrgNoClock No clock for FRG.

7.5.10 enum apb_clock_src_t

Enumerator

kCLOCK_ApbMainClk CPU & System Bus clock for APB bridge.
kCLOCK_ApbXtal32M XTAL 32M for APB bridge.
kCLOCK_ApbFro32M FRO 32M for APB bridge.
kCLOCK_ApbFro48M FRO 48M for APB bridge.

7.5.11 enum fmeas_clock_src_t

Enumerator

kCLOCK_fmeasClkIn Clock in for FMEAS.
kCLOCK_fmeasXtal32Mhz XTAL 32M for FMEAS.
kCLOCK_fmeasFRO1Mhz FRO 1M for FMEAS.
kCLOCK_fmeasXtal32kHz XTAL 32K for FMEAS.
kCLOCK_fmeasMainClock CPU & System Bus clock for FMEAS.
kCLOCK_fmeasGPIO_0_4 GPIO0_4 input for FMEAS.
kCLOCK_fmeasGPIO_0_20 GPIO0_20 input for FMEAS.
kCLOCK_fmeasGPIO_0_16 GPIO0_16 input for FMEAS.
kCLOCK_fmeasGPIO_0_15 GPIO0_15 input for FMEAS.

7.5.12 enum spifi_clock_src_t

Enumerator

kCLOCK_SpifiMainClk CPU & System Bus clock for SPIFI.
kCLOCK_SpifiXtal32M XTAL 32M for SPIFI.
kCLOCK_SpifiFro64M FRO 64M for SPIFI.
kCLOCK_SpifiFro48M FRO 48M for SPIFI.
kCLOCK_SpifiNoClock No clock for SPIFI.

7.5.13 enum adc_clock_src_t

Enumerator

kCLOCK_AdcXtal32M XTAL 32MHz for ADC.
kCLOCK_AdcFro12M FRO 12MHz for ADC.
kCLOCK_AdcNoClock No clock for ADC.

7.5.14 enum pwm_clock_source_t

Enumerator

kCLOCK_PWMOsc32Mclk 32MHz FRO or XTAL clock
kCLOCK_PWMFro48Mclk FRO 48MHz clock.
kCLOCK_PWMNoClkSel No clock selected - Shutdown functional PWM clock for power saving.
kCLOCK_PWMTestClk Test clock input - Shutdown functional PWM clock for power saving.

7.5.15 enum Fro_ClkSel_t

Enumerator

FRO12M_ENA FRO12M.
FRO32M_ENA FRO32M.
FRO48M_ENA FRO48M.
FRO64M_ENA FRO64M.
FRO96M_ENA FRO96M.

7.6 Function Documentation

7.6.1 uint32_t CLOCK_GetFreq (clock_name_t *clock*)

Parameters

clock_name_t	specify clock to be read

Returns

uint32_t frequency

Note

7.6.2 void CLOCK_AttachClk (clock_attach_id_t *connection*)

Parameters

clock_attach_id_t	specify clock mapping

Returns

none

Note

7.6.3 void CLOCK_SetClkDiv (clock_div_name_t *div_name,* uint32_t *divided_by_value,* bool *reset*)

Parameters

clock_div_name_t	specifies which DIV register we are accessing

uint32_t	specifies divisor
bool	true if a syscon clock reset should also be carried out

Returns

none

Note

7.6.4 void CLOCK_EnableClock (clock_ip_name_t *clk*)

Parameters

clock_ip_name_t	specifies which peripheral clock we are controlling

Returns

none

Note

clock_ip_name_t is a typedef clone of clock_name_t

7.6.5 void CLOCK_DisableClock (clock_ip_name_t *clk*)

Parameters

clock_ip_name_t	specifies which peripheral clock we are controlling

Returns

none

Note

clock_ip_name_t is a typedef clone of clock_name_t

7.6.6 bool CLOCK_IsClockEnable (clock_ip_name_t *clk*)

Parameters

clock_ip_-name_t	specifies which peripheral clock we are controlling

Returns

bool

Note

clock_ip_name_t is a typedef clone of clock_name_t

7.6.7 uint32_t CLOCK_GetApbCLkFreq (void)

Parameters

none	

Returns

uint32_t frequency

Note

7.6.8 uint32_t CLOCK_GetSpifiClkFreq (void)

Returns

Frequency of Spifi.

7.6.9 void CLOCK_uDelay (uint32_t *delayUs*)

Parameters

delayUs	delay duration in micro seconds

Returns

none

7.6.10 void CLOCK_XtalBasicTrim (void)

Parameters

none	

Returns

none

Note

Has no effect if CLOCK_Xtal32M_Trim has been called

7.6.11 void CLOCK_Xtal32M_Trim (int32_t *XO_32M_OSC_CAP_Delta_x1000*, const ClockCapacitanceCompensation_t * *capa_charac*)

Parameters

XO_32M_OSC_CAP_Delta_x1000	capacitance correction in fF (femtoFarad)
capa_charac	board 32M capacitance characteristics pointer

Returns

none

Note

capa_charac must point to a struct set in board.c using CLOCK_32MfXtalIecLoadpF Load capacitance, pF CLOCK_32MfXtalPPcbParCappF PCB +ve parasitic capacitance, pF CLOCK_32MfXtalNPcbParCappF PCB -ve parasitic capacitance, pF

7.6.12 void CLOCK_Xtal32k_Trim (int32_t *XO_32k_OSC_CAP_Delta_x1000,* const ClockCapacitanceCompensation_t * *capa_charac*)

Parameters

XO_32k_OSC-_CAP_Delta_-x1000	capacitance correction in fF
capa_charac	board 32k capacitance characteristics pointer

Returns

none

Note

capa_charac must point to a struct set in board.c using CLOCK_32kfXtalIecLoadpF Load capacitance, pF CLOCK_32kfXtalPPcbParCappF PCB +ve parasitic capacitance, pF CLOCK_32kfXtalNPcbParCappF PCB -ve parasitic capacitance, pF

7.6.13 void CLOCK_SetXtal32M_LDO (void)

Parameters

none	

Returns

none

7.6.14 void CLOCK_Xtal32M_WaitUntilStable (uint32_t *u32AdditionalWait_us*)

Parameters

u32Additional-Wait_us	Additional wait after hardware indicates that stability has been reached

Returns

none

Note

Operates as a tight loop. Worst case would be ∼600ms

Chapter 8
Common Driver

8.1 Overview

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

Macros

- #define ADC_RSTS
- #define MAKE_STATUS(group, code) ((((group)∗100) + (code)))
  Construct a status code value from a group and code number.
- #define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
  Construct the version number for drivers.
- #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U
  No debug console.
- #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U
  Debug console based on UART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U
  Debug console based on LPUART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U
  Debug console based on LPSCI.
- #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U
  Debug console based on USBCDC.
- #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U
  Debug console based on FLEXCOMM.
- #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U
  Debug console based on i.MX UART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U
  Debug console based on LPC_VUSART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U
  Debug console based on LPC_USART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U
  Debug console based on SWO.
- #define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
  Computes the number of elements in an array.

Typedefs

- typedef int32_t status_t
  Type used for all status and error return values.

Enumerations

- enum SYSCON_RSTn_t {
kFLASH_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_FLASH_RST_SHIFT),
kSPIFI_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_SPIFI_RST_SHIFT),
kMUX_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_MUX_RST_SHIFT),
kIOCON_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_IOCON_RST_SHIFT),
kGPIO0_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_GPIO_RST_SHIFT),
kPINT_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_PINT_RST_SHIFT),
kGINT_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_GINT_RST_SHIFT),
kDMA_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_DMA_RST_SHIFT),
kWWDT_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_WWDT_RST_SHIFT),
kRTC_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_RTC_RST_SHIFT),
kANA_INT_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_ANA_INT_CTRL_RST_SHI-FT),
kWKT_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_WAKE_UP_TIMERS_RST_SHIF-T),
kADC0_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_ADC_RST_SHIFT),
kEFUSE_RST_SHIFT_RSTn = (0 | SYSCON_PRESETCTRL0_EFUSE_RST_SHIFT),
kFC0_RST_SHIFT_RSTn,
kFC1_RST_SHIFT_RSTn,
kFC2_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_I2C0_RST_SHIFT),
kFC3_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_I2C1_RST_SHIFT),
kFC4_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_SPI0_RST_SHIFT),
kFC5_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_SPI1_RST_SHIFT),
kIRB_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_IR_RST_SHIFT),
kPWM_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_PWM_RST_SHIFT),
kRNG_RST_SHIFT_RSTn,
kFC6_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_I2C2_RST_SHIFT),
kUSART0_RST_SHIFT_RSTn = kFC0_RST_SHIFT_RSTn,
kUSART1_RST_SHIFT_RSTn = kFC1_RST_SHIFT_RSTn,
kI2C0_RST_SHIFT_RSTn = kFC2_RST_SHIFT_RSTn,
kI2C1_RST_SHIFT_RSTn = kFC3_RST_SHIFT_RSTn,
kSPI0_RST_SHIFT_RSTn = kFC4_RST_SHIFT_RSTn,
kSPI1_RST_SHIFT_RSTn = kFC5_RST_SHIFT_RSTn,
kI2C2_RST_SHIFT_RSTn = kFC6_RST_SHIFT_RSTn,
kMODEM_MASTER_SHIFT_RSTn,
kAES_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_AES_RST_SHIFT),
kRFP_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_RFP_RST_SHIFT),
kDMIC_RST_SHIFT_RSTn,
kHASH_RST_SHIFT_RSTn = ((1UL << 16) | SYSCON_PRESETCTRL1_HASH_RST_SHIFT),
kCTIMER0_RST_SHIFT_RSTn = ((2UL << 16) | ASYNC_SYSCON_ASYNCPRESETCTRL_-CT32B0_SHIFT),
kCTIMER1_RST_SHIFT_RSTn = ((2UL << 16) | ASYNC_SYSCON_ASYNCPRESETCTRL_-CT32B1_SHIFT) }

Enumeration for peripheral reset control bits.

- enum _status_groups {

kStatusGroup_Generic = 0,
kStatusGroup_FLASH = 1,
kStatusGroup_LPSPI = 4,
kStatusGroup_FLEXIO_SPI = 5,
kStatusGroup_DSPI = 6,
kStatusGroup_FLEXIO_UART = 7,
kStatusGroup_FLEXIO_I2C = 8,
kStatusGroup_LPI2C = 9,
kStatusGroup_UART = 10,
kStatusGroup_I2C = 11,
kStatusGroup_LPSCI = 12,
kStatusGroup_LPUART = 13,
kStatusGroup_SPI = 14,
kStatusGroup_XRDC = 15,
kStatusGroup_SEMA42 = 16,
kStatusGroup_SDHC = 17,
kStatusGroup_SDMMC = 18,
kStatusGroup_SAI = 19,
kStatusGroup_MCG = 20,
kStatusGroup_SCG = 21,
kStatusGroup_SDSPI = 22,
kStatusGroup_FLEXIO_I2S = 23,
kStatusGroup_FLEXIO_MCULCD = 24,
kStatusGroup_FLASHIAP = 25,
kStatusGroup_FLEXCOMM_I2C = 26,
kStatusGroup_I2S = 27,
kStatusGroup_IUART = 28,
kStatusGroup_CSI = 29,
kStatusGroup_MIPI_DSI = 30,
kStatusGroup_SDRAMC = 35,
kStatusGroup_POWER = 39,
kStatusGroup_ENET = 40,
kStatusGroup_PHY = 41,
kStatusGroup_TRGMUX = 42,
kStatusGroup_SMARTCARD = 43,
kStatusGroup_LMEM = 44,
kStatusGroup_QSPI = 45,
kStatusGroup_DMA = 50,
kStatusGroup_EDMA = 51,
kStatusGroup_DMAMGR = 52,
kStatusGroup_FLEXCAN = 53,
kStatusGroup_LTC = 54,
kStatusGroup_FLEXIO_CAMERA = 55,
kStatusGroup_LPC_SPI = 56,
kStatusGroup_LPC_USART = 57,
kStatusGroup_DMIC = 58,
kStatusGroup_SDIF = 59,
kStatusGroup_SPIFI = 60,
kStatusGroup_OTP = 61,

kStatusGroup_CODEC = 148 }
Status group numbers.
- enum
Generic status return codes.

Functions

- void RESET_SetPeripheralReset (reset_ip_name_t peripheral)
Assert reset to peripheral.
- void RESET_ClearPeripheralReset (reset_ip_name_t peripheral)
Clear reset to peripheral.
- void RESET_PeripheralReset (reset_ip_name_t peripheral)
Reset peripheral module.
- void RESET_SystemReset (void)
Reset the chip.
- static status_t EnableIRQ (IRQn_Type interrupt)
Enable specific interrupt.
- static status_t DisableIRQ (IRQn_Type interrupt)
Disable specific interrupt.
- static uint32_t DisableGlobalIRQ (void)
Disable the global IRQ.
- static void EnableGlobalIRQ (uint32_t primask)
Enable the global IRQ.
- void EnableDeepSleepIRQ (IRQn_Type interrupt)
Enable specific interrupt for wake-up from deep-sleep mode.
- void DisableDeepSleepIRQ (IRQn_Type interrupt)
Disable specific interrupt for wake-up from deep-sleep mode.
- void ∗ SDK_Malloc (size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.
- void SDK_Free (void ∗ptr)
Free memory.
- void SDK_DelayAtLeastUs (uint32_t delay_us, uint32_t coreClock_Hz)
Delay at least for some time.

Driver version

- #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
common driver version 2.2.0.

Min/max macros

- #define **MIN**(a, b) (((a) < (b)) ? (a) : (b))
- #define **MAX**(a, b) (((a) > (b)) ? (a) : (b))

UINT16_MAX/UINT32_MAX value

- #define **UINT16_MAX** ((uint16_t)-1)
- #define **UINT32_MAX** ((uint32_t)-1)

Timer utilities

- #define USEC_TO_COUNT(us, clockFreqInHz) (uint64_t)(((uint64_t)(us) ∗ (clockFreqInHz)) / 1000000U)
  Macro to convert a microsecond period to raw count value.
- #define COUNT_TO_USEC(count, clockFreqInHz) (uint64_t)((uint64_t)count ∗ 1000000U / clockFreqInHz)
  Macro to convert a raw count value to microsecond.
- #define MSEC_TO_COUNT(ms, clockFreqInHz) (uint64_t)((uint64_t)ms ∗ clockFreqInHz / 1000U)
  Macro to convert a millisecond period to raw count value.
- #define COUNT_TO_MSEC(count, clockFreqInHz) (uint64_t)((uint64_t)count ∗ 1000U / clockFreqInHz)
  Macro to convert a raw count value to millisecond.

Alignment variable definition macros

- #define **SDK_ALIGN**(var, alignbytes) var
- #define SDK_SIZEALIGN(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1)) & (unsigned int)(∼(unsigned int)((alignbytes)-1)))
  Macro to change a value to a given size aligned value.

Non-cacheable region definition macros

- #define **AT_NONCACHEABLE_SECTION**(var) var
- #define **AT_NONCACHEABLE_SECTION_ALIGN**(var, alignbytes) var
- #define **AT_NONCACHEABLE_SECTION_INIT**(var) var
- #define **AT_NONCACHEABLE_SECTION_ALIGN_INIT**(var, alignbytes) var

Suppress fallthrough warning macro

- #define **SUPPRESS_FALL_THROUGH_WARNING**()

8.2 Macro Definition Documentation

8.2.1 #define ADC_RSTS

Value:

```
{                         \
        kADC0_RST_SHIFT_RSTn \
    } /* Reset bits for ADC peripheral */
```

Array initializers with peripheral reset bits

8.2.2 #define MAKE_STATUS(*group*, *code*) ((((group)*100) + (code)))

8.2.3 #define MAKE_VERSION(*major*, *minor*, *bugfix*) (((major) << 16) | ((minor) << 8) | (bugfix))

8.2.4 #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

8.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

8.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

8.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

8.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

8.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

8.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

8.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

8.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

8.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

8.2.14 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

8.2.15 #define ARRAY_SIZE(*x*) (sizeof(x) / sizeof((x)[0]))

8.3 Typedef Documentation

8.3.1 typedef int32_t status_t

8.4 Enumeration Type Documentation

8.4.1 enum SYSCON_RSTn_t

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

Enumerator

kFLASH_RST_SHIFT_RSTn Flash controller reset control
kSPIFI_RST_SHIFT_RSTn SpiFi reset control
kMUX_RST_SHIFT_RSTn Input mux reset control
kIOCON_RST_SHIFT_RSTn IOCON reset control
kGPIO0_RST_SHIFT_RSTn GPIO0 reset control
kPINT_RST_SHIFT_RSTn Pin interrupt (PINT) reset control
kGINT_RST_SHIFT_RSTn Grouped interrupt (PINT) reset control.
kDMA_RST_SHIFT_RSTn DMA reset control
kWWDT_RST_SHIFT_RSTn Watchdog timer reset control
kRTC_RST_SHIFT_RSTn RTC reset control
kANA_INT_RST_SHIFT_RSTn Analog interrupt controller reset
kWKT_RST_SHIFT_RSTn Wakeup timer reset
kADC0_RST_SHIFT_RSTn ADC0 reset control
kEFUSE_RST_SHIFT_RSTn EFUSE reset control
kFC0_RST_SHIFT_RSTn Flexcomm Interface 0 reset control
kFC1_RST_SHIFT_RSTn Flexcomm Interface 1 reset control
kFC2_RST_SHIFT_RSTn Flexcomm Interface 2 reset control
kFC3_RST_SHIFT_RSTn Flexcomm Interface 3 reset control
kFC4_RST_SHIFT_RSTn Flexcomm Interface 4 reset control
kFC5_RST_SHIFT_RSTn Flexcomm Interface 5 reset control
kIRB_RST_SHIFT_RSTn IR Blaster reset control
kPWM_RST_SHIFT_RSTn PWM reset control
kRNG_RST_SHIFT_RSTn Random number generator reset control
kFC6_RST_SHIFT_RSTn Flexcomm Interface 6 reset control
kUSART0_RST_SHIFT_RSTn USART0 reset control == Flexcomm0
kUSART1_RST_SHIFT_RSTn USART0 reset control == Flexcomm1
kI2C0_RST_SHIFT_RSTn I2C0 reset control == Flexcomm 2
kI2C1_RST_SHIFT_RSTn I2C1 reset control == Flexcomm 3
kSPI0_RST_SHIFT_RSTn SPI0 reset control == Flexcomm 4
kSPI1_RST_SHIFT_RSTn SPI1 reset control == Flexcomm 5
kI2C2_RST_SHIFT_RSTn I2C2 reset control == Flexcomm 6
kMODEM_MASTER_SHIFT_RSTn AHB Modem master interface reset
kAES_RST_SHIFT_RSTn Encryption module reset control
kRFP_RST_SHIFT_RSTn Radio front end controller reset
kDMIC_RST_SHIFT_RSTn Digital microphone interface reset control
kHASH_RST_SHIFT_RSTn Hash SHA reset
kCTIMER0_RST_SHIFT_RSTn CT32B0 reset control
kCTIMER1_RST_SHIFT_RSTn CT32B1 reset control

8.4.2 enum _status_groups

Enumerator

kStatusGroup_Generic Group number for generic status codes.
kStatusGroup_FLASH Group number for FLASH status codes.
kStatusGroup_LPSPI Group number for LPSPI status codes.
kStatusGroup_FLEXIO_SPI Group number for FLEXIO SPI status codes.
kStatusGroup_DSPI Group number for DSPI status codes.
kStatusGroup_FLEXIO_UART Group number for FLEXIO UART status codes.
kStatusGroup_FLEXIO_I2C Group number for FLEXIO I2C status codes.
kStatusGroup_LPI2C Group number for LPI2C status codes.
kStatusGroup_UART Group number for UART status codes.
kStatusGroup_I2C Group number for UART status codes.
kStatusGroup_LPSCI Group number for LPSCI status codes.
kStatusGroup_LPUART Group number for LPUART status codes.
kStatusGroup_SPI Group number for SPI status code.
kStatusGroup_XRDC Group number for XRDC status code.
kStatusGroup_SEMA42 Group number for SEMA42 status code.
kStatusGroup_SDHC Group number for SDHC status code.
kStatusGroup_SDMMC Group number for SDMMC status code.
kStatusGroup_SAI Group number for SAI status code.
kStatusGroup_MCG Group number for MCG status codes.
kStatusGroup_SCG Group number for SCG status codes.
kStatusGroup_SDSPI Group number for SDSPI status codes.
kStatusGroup_FLEXIO_I2S Group number for FLEXIO I2S status codes.
kStatusGroup_FLEXIO_MCULCD Group number for FLEXIO LCD status codes.
kStatusGroup_FLASHIAP Group number for FLASHIAP status codes.
kStatusGroup_FLEXCOMM_I2C Group number for FLEXCOMM I2C status codes.
kStatusGroup_I2S Group number for I2S status codes.
kStatusGroup_IUART Group number for IUART status codes.
kStatusGroup_CSI Group number for CSI status codes.
kStatusGroup_MIPI_DSI Group number for MIPI DSI status codes.
kStatusGroup_SDRAMC Group number for SDRAMC status codes.
kStatusGroup_POWER Group number for POWER status codes.
kStatusGroup_ENET Group number for ENET status codes.
kStatusGroup_PHY Group number for PHY status codes.
kStatusGroup_TRGMUX Group number for TRGMUX status codes.
kStatusGroup_SMARTCARD Group number for SMARTCARD status codes.
kStatusGroup_LMEM Group number for LMEM status codes.
kStatusGroup_QSPI Group number for QSPI status codes.
kStatusGroup_DMA Group number for DMA status codes.
kStatusGroup_EDMA Group number for EDMA status codes.
kStatusGroup_DMAMGR Group number for DMAMGR status codes.
kStatusGroup_FLEXCAN Group number for FlexCAN status codes.

kStatusGroup_LTC Group number for LTC status codes.
kStatusGroup_FLEXIO_CAMERA Group number for FLEXIO CAMERA status codes.
kStatusGroup_LPC_SPI Group number for LPC_SPI status codes.
kStatusGroup_LPC_USART Group number for LPC_USART status codes.
kStatusGroup_DMIC Group number for DMIC status codes.
kStatusGroup_SDIF Group number for SDIF status codes.
kStatusGroup_SPIFI Group number for SPIFI status codes.
kStatusGroup_OTP Group number for OTP status codes.
kStatusGroup_MCAN Group number for MCAN status codes.
kStatusGroup_CAAM Group number for CAAM status codes.
kStatusGroup_ECSPI Group number for ECSPI status codes.
kStatusGroup_USDHC Group number for USDHC status codes.
kStatusGroup_LPC_I2C Group number for LPC_I2C status codes.
kStatusGroup_DCP Group number for DCP status codes.
kStatusGroup_MSCAN Group number for MSCAN status codes.
kStatusGroup_ESAI Group number for ESAI status codes.
kStatusGroup_FLEXSPI Group number for FLEXSPI status codes.
kStatusGroup_MMDC Group number for MMDC status codes.
kStatusGroup_PDM Group number for MIC status codes.
kStatusGroup_SDMA Group number for SDMA status codes.
kStatusGroup_ICS Group number for ICS status codes.
kStatusGroup_SPDIF Group number for SPDIF status codes.
kStatusGroup_LPC_MINISPI Group number for LPC_MINISPI status codes.
kStatusGroup_HASHCRYPT Group number for Hashcrypt status codes.
kStatusGroup_LPC_SPI_SSP Group number for LPC_SPI_SSP status codes.
kStatusGroup_I3C Group number for I3C status codes.
kStatusGroup_LPC_I2C_1 Group number for LPC_I2C_1 status codes.
kStatusGroup_NOTIFIER Group number for NOTIFIER status codes.
kStatusGroup_DebugConsole Group number for debug console status codes.
kStatusGroup_SEMC Group number for SEMC status codes.
kStatusGroup_ApplicationRangeStart Starting number for application groups.
kStatusGroup_IAP Group number for IAP status codes.
kStatusGroup_HAL_GPIO Group number for HAL GPIO status codes.
kStatusGroup_HAL_UART Group number for HAL UART status codes.
kStatusGroup_HAL_TIMER Group number for HAL TIMER status codes.
kStatusGroup_HAL_SPI Group number for HAL SPI status codes.
kStatusGroup_HAL_I2C Group number for HAL I2C status codes.
kStatusGroup_HAL_FLASH Group number for HAL FLASH status codes.
kStatusGroup_HAL_PWM Group number for HAL PWM status codes.
kStatusGroup_HAL_RNG Group number for HAL RNG status codes.
kStatusGroup_TIMERMANAGER Group number for TiMER MANAGER status codes.
kStatusGroup_SERIALMANAGER Group number for SERIAL MANAGER status codes.
kStatusGroup_LED Group number for LED status codes.
kStatusGroup_BUTTON Group number for BUTTON status codes.
kStatusGroup_EXTERN_EEPROM Group number for EXTERN EEPROM status codes.

kStatusGroup_SHELL Group number for SHELL status codes.
kStatusGroup_MEM_MANAGER Group number for MEM MANAGER status codes.
kStatusGroup_LIST Group number for List status codes.
kStatusGroup_OSA Group number for OSA status codes.
kStatusGroup_COMMON_TASK Group number for Common task status codes.
kStatusGroup_MSG Group number for messaging status codes.
kStatusGroup_SDK_OCOTP Group number for OCOTP status codes.
kStatusGroup_SDK_FLEXSPINOR Group number for FLEXSPINOR status codes.
kStatusGroup_CODEC Group number for codec status codes.

8.4.3 anonymous enum

8.5 Function Documentation

8.5.1 void RESET_SetPeripheralReset (reset_ip_name_t *peripheral*)

Asserts reset signal to specified peripheral module.

Parameters

peripheral	Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

8.5.2 void RESET_ClearPeripheralReset (reset_ip_name_t *peripheral*)

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

peripheral	Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

8.5.3 void RESET_PeripheralReset (reset_ip_name_t *peripheral*)

Reset peripheral module.

Parameters

peripheral	Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

8.5.4 void RESET_SystemReset (void)

Full software reset of the chip. On reboot, function POWER_GetResetCause() from fsl_power.h will return RESET_SYS_REQ

8.5.5 static status_t EnableIRQ (IRQn_Type *interrupt*) [inline], [static]

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

interrupt	The IRQ number.

Return values

kStatus_Success	Interrupt enabled successfully
kStatus_Fail	Failed to enable the interrupt

8.5.6 static status_t DisableIRQ (IRQn_Type *interrupt*) [inline], [static]

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

interrupt	The IRQ number.

Return values

kStatus_Success	Interrupt disabled successfully
kStatus_Fail	Failed to disable the interrupt

8.5.7 static uint32_t DisableGlobalIRQ (void) [inline], [static]

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

Returns

Current primask value.

8.5.8 static void EnableGlobalIRQ (uint32_t *primask*) [inline], [static]

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

primask	value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

8.5.9 void EnableDeepSleepIRQ (IRQn_Type *interrupt*)

Enable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note

This function also enables the interrupt in the NVIC (EnableIRQ() is called internaly).

Parameters

interrupt	The IRQ number.

8.5.10 void DisableDeepSleepIRQ (IRQn_Type *interrupt*)

Disable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note

This function also disables the interrupt in the NVIC (DisableIRQ() is called internaly).

Parameters

interrupt	The IRQ number.

8.5.11 void∗ SDK_Malloc (size_t *size,* size_t *alignbytes*)

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

size	The length required to malloc.
alignbytes	The alignment size.

Return values

The	allocated memory.

8.5.12 void SDK_Free (void ∗ *ptr*)

Parameters

ptr	The memory to be release.

8.5.13 void SDK_DelayAtLeastUs (uint32_t *delay_us,* uint32_t *coreClock_Hz*)

Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

delay_us	Delay time in unit of microsecond.
coreClock_Hz	Core clock frequency with Hz.

Chapter 9
CTIMER: Standard counter/timers

9.1 Overview

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

9.2 Function groups

The cTimer driver supports the generation of PWM signals, input capture, and setting up the timer match conditions.

9.2.1 Initialization and deinitialization

The function CTIMER_Init() initializes the cTimer with specified configurations. The function CTIMER_GetDefaultConfig() gets the default configurations. The initialization function configures the counter/timer mode and input selection when running in counter mode.

The function CTIMER_Deinit() stops the timer and turns off the module clock.

9.2.2 PWM Operations

The function CTIMER_SetupPwm() sets up channels for PWM output. Each channel has its own duty cycle, however the same PWM period is applied to all channels requesting the 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 CTIMER_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular channel.

9.2.3 Match Operation

The function CTIMER_SetupMatch() sets up channels for match operation. Each channel is configured with a match value: if the counter should stop on match, if counter should reset on match, and output pin action. The output signal can be cleared, set, or toggled on match.

9.2.4 Input capture operations

The function CTIMER_SetupCapture() sets up an channel for input capture. The user can specify the capture edge and if a interrupt should be generated when processing the input signal.

9.3 Typical use case

9.3.1 Match example

Set up a match channel to toggle output when a match occurs. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer

9.3.2 PWM output example

Set up a channel for PWM output. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer

Files

- file fsl_ctimer.h

Data Structures

- struct ctimer_match_config_t
  Match configuration. More...
- struct ctimer_config_t
  Timer configuration structure. More...

Enumerations

- enum ctimer_capture_channel_t {
  kCTIMER_Capture_0 = 0U,
  kCTIMER_Capture_1,
  kCTIMER_Capture_2,
  kCTIMER_Capture_3 }
  List of Timer capture channels.
- enum ctimer_capture_edge_t {
  kCTIMER_Capture_RiseEdge = 1U,
  kCTIMER_Capture_FallEdge = 2U,
  kCTIMER_Capture_BothEdge = 3U }
  List of capture edge options.
- enum ctimer_match_t {
  kCTIMER_Match_0 = 0U,
  kCTIMER_Match_1,
  kCTIMER_Match_2,
  kCTIMER_Match_3 }
  List of Timer match registers.
- enum ctimer_match_output_control_t {
  kCTIMER_Output_NoAction = 0U,
  kCTIMER_Output_Clear,
  kCTIMER_Output_Set,
  kCTIMER_Output_Toggle }

List of output control options.
- enum ctimer_timer_mode_t
  List of Timer modes.
- enum ctimer_interrupt_enable_t {
  kCTIMER_Match0InterruptEnable = CTIMER_MCR_MR0I_MASK,
  kCTIMER_Match1InterruptEnable = CTIMER_MCR_MR1I_MASK,
  kCTIMER_Match2InterruptEnable = CTIMER_MCR_MR2I_MASK,
  kCTIMER_Match3InterruptEnable = CTIMER_MCR_MR3I_MASK }
  List of Timer interrupts.
- enum ctimer_status_flags_t {
  kCTIMER_Match0Flag = CTIMER_IR_MR0INT_MASK,
  kCTIMER_Match1Flag = CTIMER_IR_MR1INT_MASK,
  kCTIMER_Match2Flag = CTIMER_IR_MR2INT_MASK,
  kCTIMER_Match3Flag = CTIMER_IR_MR3INT_MASK }
  List of Timer flags.
- enum ctimer_callback_type_t {
  kCTIMER_SingleCallback,
  kCTIMER_MultipleCallback }
  Callback type when registering for a callback.

Functions

- void CTIMER_SetupMatch (CTIMER_Type ∗base, ctimer_match_t matchChannel, const ctimer_match_config_t ∗config)
  Setup the match register.
- void CTIMER_SetupCapture (CTIMER_Type ∗base, ctimer_capture_channel_t capture, ctimer_capture_edge_t edge, bool enableInt)
  Setup the capture.
- static uint32_t CTIMER_GetTimerCountValue (CTIMER_Type ∗base)
  Get the timer count value from TC register.
- void CTIMER_RegisterCallBack (CTIMER_Type ∗base, ctimer_callback_t ∗cb_func, ctimer_callback_type_t cb_type)
  Register callback.
- static void CTIMER_Reset (CTIMER_Type ∗base)
  Reset the counter.

Driver version

- #define FSL_CTIMER_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  Version 2.0.2.

Initialization and deinitialization

- void CTIMER_Init (CTIMER_Type ∗base, const ctimer_config_t ∗config)
  Ungates the clock and configures the peripheral for basic operation.
- void CTIMER_Deinit (CTIMER_Type ∗base)
  Gates the timer clock.
- void CTIMER_GetDefaultConfig (ctimer_config_t ∗config)
  Fills in the timers configuration structure with the default settings.

PWM setup operations

- status_t CTIMER_SetupPwmPeriod (CTIMER_Type *base, ctimer_match_t matchChannel, uint32_t pwmPeriod, uint32_t pulsePeriod, bool enableInt)
  Configures the PWM signal parameters.
- status_t CTIMER_SetupPwm (CTIMER_Type *base, ctimer_match_t matchChannel, uint8_t dutyCyclePercent, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, bool enableInt)
  Configures the PWM signal parameters.
- static void CTIMER_UpdatePwmPulsePeriod (CTIMER_Type *base, ctimer_match_t matchChannel, uint32_t pulsePeriod)
  Updates the pulse period of an active PWM signal.
- void CTIMER_UpdatePwmDutycycle (CTIMER_Type *base, ctimer_match_t matchChannel, uint8_t dutyCyclePercent)
  Updates the duty cycle of an active PWM signal.

Interrupt Interface

- static void CTIMER_EnableInterrupts (CTIMER_Type *base, uint32_t mask)
  Enables the selected Timer interrupts.
- static void CTIMER_DisableInterrupts (CTIMER_Type *base, uint32_t mask)
  Disables the selected Timer interrupts.
- static uint32_t CTIMER_GetEnabledInterrupts (CTIMER_Type *base)
  Gets the enabled Timer interrupts.

Status Interface

- static uint32_t CTIMER_GetStatusFlags (CTIMER_Type *base)
  Gets the Timer status flags.
- static void CTIMER_ClearStatusFlags (CTIMER_Type *base, uint32_t mask)
  Clears the Timer status flags.

Counter Start and Stop

- static void CTIMER_StartTimer (CTIMER_Type *base)
  Starts the Timer counter.
- static void CTIMER_StopTimer (CTIMER_Type *base)
  Stops the Timer counter.

9.4 Data Structure Documentation

9.4.1 struct ctimer_match_config_t

This structure holds the configuration settings for each match register.

Data Fields

- uint32_t matchValue
  This is stored in the match register.

- bool enableCounterReset
  true: Match will reset the counter false: Match will not reser the counter
- bool enableCounterStop
  true: Match will stop the counter false: Match will not stop the counter
- ctimer_match_output_control_t outControl
  Action to be taken on a match on the EM bit/output.
- bool outPinInitState
  Initial value of the EM bit/output.
- bool enableInterrupt
  true: Generate interrupt upon match false: Do not generate interrupt on match

9.4.2 struct ctimer_config_t

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the CTIMER_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Data Fields

- ctimer_timer_mode_t mode
  Timer mode.
- ctimer_capture_channel_t input
  Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC.
- uint32_t prescale
  Prescale value.

9.5 Enumeration Type Documentation

9.5.1 enum ctimer_capture_channel_t

Enumerator

kCTIMER_Capture_0 Timer capture channel 0.
kCTIMER_Capture_1 Timer capture channel 1.
kCTIMER_Capture_2 Timer capture channel 2.
kCTIMER_Capture_3 Timer capture channel 3.

9.5.2 enum ctimer_capture_edge_t

Enumerator

kCTIMER_Capture_RiseEdge Capture on rising edge.

kCTIMER_Capture_FallEdge Capture on falling edge.
kCTIMER_Capture_BothEdge Capture on rising and falling edge.

9.5.3 enum ctimer_match_t

Enumerator

kCTIMER_Match_0 Timer match register 0.
kCTIMER_Match_1 Timer match register 1.
kCTIMER_Match_2 Timer match register 2.
kCTIMER_Match_3 Timer match register 3.

9.5.4 enum ctimer_match_output_control_t

Enumerator

kCTIMER_Output_NoAction No action is taken.
kCTIMER_Output_Clear Clear the EM bit/output to 0.
kCTIMER_Output_Set Set the EM bit/output to 1.
kCTIMER_Output_Toggle Toggle the EM bit/output.

9.5.5 enum ctimer_interrupt_enable_t

Enumerator

kCTIMER_Match0InterruptEnable Match 0 interrupt.
kCTIMER_Match1InterruptEnable Match 1 interrupt.
kCTIMER_Match2InterruptEnable Match 2 interrupt.
kCTIMER_Match3InterruptEnable Match 3 interrupt.

9.5.6 enum ctimer_status_flags_t

Enumerator

kCTIMER_Match0Flag Match 0 interrupt flag.
kCTIMER_Match1Flag Match 1 interrupt flag.
kCTIMER_Match2Flag Match 2 interrupt flag.
kCTIMER_Match3Flag Match 3 interrupt flag.

9.5.7 enum ctimer_callback_type_t

When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Enumerator

kCTIMER_SingleCallback Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently

kCTIMER_MultipleCallback Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

9.6 Function Documentation

9.6.1 void CTIMER_Init (CTIMER_Type * *base*, const ctimer_config_t * *config*)

Note

This API should be called at the beginning of the application before using the driver.

Parameters

base	Ctimer peripheral base address
config	Pointer to the user configuration structure.

9.6.2 void CTIMER_Deinit (CTIMER_Type * *base*)

Parameters

base	Ctimer peripheral base address

9.6.3 void CTIMER_GetDefaultConfig (ctimer_config_t * *config*)

The default values are:

```
*   config->mode = kCTIMER_TimerMode;
*   config->input = kCTIMER_Capture_0;
*   config->prescale = 0;
*
```

Parameters

config	Pointer to the user configuration structure.

9.6.4 status_t CTIMER_SetupPwmPeriod (CTIMER_Type * *base,* ctimer_match_t *matchChannel,* uint32_t *pwmPeriod,* uint32_t *pulsePeriod,* bool *enableInt*)

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function will assign match channel 3 to set the PWM cycle.

Note

When setting PWM output from multiple output pins, all should use the same PWM period

Parameters

base	Ctimer peripheral base address
matchChannel	Match pin to be used to output the PWM signal
pwmPeriod	PWM period match value
pulsePeriod	Pulse width match value
enableInt	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt is generated

Returns

kStatus_Success on success kStatus_Fail If matchChannel passed in is 3; this channel is reserved to set the PWM period

9.6.5 status_t CTIMER_SetupPwm (CTIMER_Type * *base,* ctimer_match_t *matchChannel,* uint8_t *dutyCyclePercent,* uint32_t *pwmFreq_Hz,* uint32_t *srcClock_Hz,* bool *enableInt*)

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function will assign match channel 3 to set the PWM cycle.

Note

When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER_SetupPwmPeriod to set up the PWM with high resolution.

Parameters

base	Ctimer peripheral base address
matchChannel	Match pin to be used to output the PWM signal
dutyCycle-Percent	PWM pulse width; the value should be between 0 to 100
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	Timer counter clock in Hz
enableInt	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt is generated

Returns

kStatus_Success on success kStatus_Fail If matchChannel passed in is 3; this channel is reserved to set the PWM cycle

9.6.6 static void CTIMER_UpdatePwmPulsePeriod (CTIMER_Type ∗ *base,* ctimer_match_t *matchChannel,* uint32_t *pulsePeriod*) [inline], [static]

Parameters

base	Ctimer peripheral base address
matchChannel	Match pin to be used to output the PWM signal
pulsePeriod	New PWM pulse width match value

9.6.7 void CTIMER_UpdatePwmDutycycle (CTIMER_Type ∗ *base,* ctimer_match_t *matchChannel,* uint8_t *dutyCyclePercent*)

Note

Please use CTIMER_UpdatePwmPulsePeriod to update the PWM with high resolution.

Parameters

base	Ctimer peripheral base address
matchChannel	Match pin to be used to output the PWM signal
dutyCyclePercent	New PWM pulse width; the value should be between 0 to 100

9.6.8 void CTIMER_SetupMatch (CTIMER_Type ∗ *base*, ctimer_match_t *matchChannel*, const ctimer_match_config_t ∗ *config*)

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

base	Ctimer peripheral base address
matchChannel	Match register to configure
config	Pointer to the match configuration structure

9.6.9 void CTIMER_SetupCapture (CTIMER_Type ∗ *base*, ctimer_capture_channel_t *capture*, ctimer_capture_edge_t *edge*, bool *enableInt*)

Parameters

base	Ctimer peripheral base address
capture	Capture channel to configure
edge	Edge on the channel that will trigger a capture
enableInt	Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

9.6.10 static uint32_t CTIMER_GetTimerCountValue (CTIMER_Type ∗ *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address.

Returns

return the timer count value.

9.6.11 void CTIMER_RegisterCallBack (CTIMER_Type * *base*, ctimer_callback_t * *cb_func*, ctimer_callback_type_t *cb_type*)

Parameters

base	Ctimer peripheral base address
cb_func	callback function
cb_type	callback function type, singular or multiple

9.6.12 static void CTIMER_EnableInterrupts (CTIMER_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	Ctimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration ctimer-_interrupt_enable_t

9.6.13 static void CTIMER_DisableInterrupts (CTIMER_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	Ctimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration ctimer-_interrupt_enable_t

9.6.14 static uint32_t CTIMER_GetEnabledInterrupts (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration ctimer_interrupt_enable_t

9.6.15 static uint32_t CTIMER_GetStatusFlags (CTIMER_Type ∗ *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

Returns

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

9.6.16 static void CTIMER_ClearStatusFlags (CTIMER_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	Ctimer peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration ctimer_status_flags_t

9.6.17 static void CTIMER_StartTimer (CTIMER_Type ∗ *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

9.6.18 static void CTIMER_StopTimer (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

9.6.19 static void CTIMER_Reset (CTIMER_Type * *base*) [inline], [static]

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

base	Ctimer peripheral base address

Chapter 10
Debug Console

10.1 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.

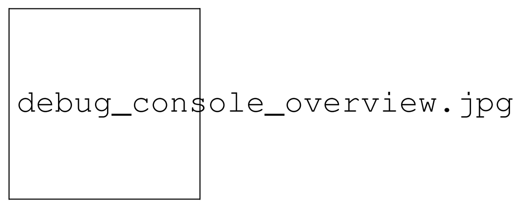

Figure 10.1.1: Debug console overview

10.2 Function groups

10.2.1 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);
```

10.2.2 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
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).
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 ∗

specifier	Qualifying Input	Type of argument
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 */
```

10.3 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.

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.

10.4 Function groups

10.4.1 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);
```

Selects the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_None = 0U,
    kSerialPort_Uart = 1U,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the debug_console_state_t structure, such as shown here.

```
typedef struct DebugConsoleState
{
    uint8_t uartHandleBuffer[HAL_UART_HANDLE_SIZE];
    hal_uart_status_t (*putChar)(hal_uart_handle_t handle, const uint8_t *data, size_t
      length);
    hal_uart_status_t (*getChar)(hal_uart_handle_t handle, uint8_t *data, size_t length);

    serial_port_type_t type;
} debug_console_state_t;
```

This example shows how to call the DbgConsole_Init() given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_USART_INSTANCE, BOARD_DEBUG_USART_BAUDRATE,
      BOARD_DEBUG_USART_TYPE,
                              BOARD_DEBUG_USART_CLK_FREQ);
```

10.4.2 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.

flags	Description
+	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
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).
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 ∗

specifier	Qualifying Input	Type of argument
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(const 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 */
```

10.5 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
  whether provide low level IO implementation to toolchain printf and scanf.
- #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.

10.6 Macro Definition Documentation

10.6.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

10.6.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

10.6.3 #define DEBUGCONSOLE_DISABLE 2U

10.6.4 #define SDK_DEBUGCONSOLE 1U

The macro only support to be redefined in project setting.

10.6.5 #define SDK_DEBUGCONSOLE_UART

For example, within MCUXpresso, if the macro SDK_DEBUGCONSOLE_UART is defined, **sys_write and __sys_readc will be used when __REDLIB** is defined; _write and _read will be used in other cases. If the macro SDK_DEBUGCONSOLE_UART is not defined, the semihost will be used.

10.6.6 #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.

10.7 Function Documentation

10.7.1 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. • kSerialPort_Uart, • kSerialPort_UsbCdc • kSerialPort_UsbCdcVirtual.
clkSrcFreq	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

kStatus_Success	Execution successfully

10.7.2 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.

10.7.3 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.

10.7.4 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.

10.7.5 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_B-LOCKING 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.

10.7.6 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_B-LOCKING 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.

10.7.7 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.

10.7.8 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

10.7.9 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 "".

10.8 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.

10.8.1 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.

10.8.2 Guide Semihosting for Keil µVision

NOTE: Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

10.8.3 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.

10.8.4 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.

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.

10.8.5 Guide Semihosting for IAR

NOTE: After the setting both "printf" and "scanf" are available for debugging.

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. Start the project by choosing Project>Download and Debug.
4. Choose View>Terminal I/O to display the output from the I/O operations.

10.8.6 Guide Semihosting for Keil µVision

NOTE: Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

10.8.7 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.

10.9 SWO

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDEs also support SWO, such IAR and KEIL, both input and output are supported, see below for details.

10.9.1 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

10.9.1.1 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.

10.9.2 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.

10.9.3 Guide SWO for MCUXpresso IDE

NOTE: MCUX support SWO for LPC-Link2 debug probe only.

10.9.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 11
DMA: Direct Memory Access Controller Driver

11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access (DMA) of MCU-Xpresso SDK devices.

11.2 Typical use case

11.2.1 DMA Operation

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

Files

- file fsl_dma.h

Data Structures

- struct dma_descriptor_t
  DMA descriptor structure. More...
- struct dma_xfercfg_t
  DMA transfer configuration. More...
- struct dma_channel_trigger_t
  DMA channel trigger. More...
- struct dma_channel_config_t
  DMA channel trigger. More...
- struct dma_transfer_config_t
  DMA transfer configuration. More...
- struct dma_handle_t
  DMA transfer handle structure. More...

Macros

- #define DMA_MAX_TRANSFER_COUNT 0x400
  DMA max transfer size.
- #define FSL_FEATURE_DMA_NUMBER_OF_CHANNELSn(x) FSL_FEATURE_DMA_NUMBER_OF_CHANNELS
  DMA channel numbers.
- #define FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (16U)
  DMA head link descriptor table align size.
- #define DMA_ALLOCATE_HEAD_DESCRIPTORS(name, number) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)
  DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

- #define DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)
  DMA head descriptor table allocate macro at noncacheable part To simplify user interface, this macro will help allocate descriptor memory at noncacheable part, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_LINK_DESCRIPTORS(name, number) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)
  DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)
  DMA link descriptor table allocate macro at noncacheable part To simplify user interface, this macro will help allocate descriptor memory at noncacheable part, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_COMMON_REG_GET(base, channel, reg) (((volatile uint32_t ∗)(&((base)->COMMON[0].reg)))[DMA_CHANNEL_GROUP(channel)])
  DMA linked descriptor address algin size.
- #define DMA_DESCRIPTOR_END_ADDRESS(start, inc, bytes, width) ((void ∗)((uint32_t)(start) + inc ∗ bytes - inc ∗ width))
  DMA descriptor end address calculate.
- #define DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes)
  DMA channel transfer configurations macro.

Typedefs

- typedef void(∗ dma_callback)(struct _dma_handle ∗handle, void ∗userData, bool transferDone, uint32_t intmode)
  Define Callback function for DMA.

Enumerations

- enum _dma_transfer_status { kStatus_DMA_Busy = MAKE_STATUS(kStatusGroup_DMA, 0) }
  DMA transfer status.
- enum _dma_addr_interleave_size {
  kDMA_AddressInterleave0xWidth = 0U,
  kDMA_AddressInterleave1xWidth = 1U,
  kDMA_AddressInterleave2xWidth = 2U,
  kDMA_AddressInterleave4xWidth = 4U }
  dma address interleave size
- enum _dma_transfer_width {
  kDMA_Transfer8BitWidth = 1U,
  kDMA_Transfer16BitWidth = 2U,
  kDMA_Transfer32BitWidth = 4U }
  dma transfer width
- enum dma_priority_t {

kDMA_ChannelPriority0 = 0,
kDMA_ChannelPriority1,
kDMA_ChannelPriority2,
kDMA_ChannelPriority3,
kDMA_ChannelPriority4,
kDMA_ChannelPriority5,
kDMA_ChannelPriority6,
kDMA_ChannelPriority7 }
DMA channel priority.
- enum dma_irq_t {
kDMA_IntA,
kDMA_IntB,
kDMA_IntError }
DMA interrupt flags.
- enum dma_trigger_type_t {
kDMA_NoTrigger = 0,
kDMA_LowLevelTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1) | DMA_CHANNEL_CFG-_TRIGTYPE(1),
kDMA_HighLevelTrigger,
kDMA_FallingEdgeTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1),
kDMA_RisingEdgeTrigger }
DMA trigger type.
- enum _dma_burst_size {
kDMA_BurstSize1 = 0U,
kDMA_BurstSize2 = 1U,
kDMA_BurstSize4 = 2U,
kDMA_BurstSize8 = 3U,
kDMA_BurstSize16 = 4U,
kDMA_BurstSize32 = 5U,
kDMA_BurstSize64 = 6U,
kDMA_BurstSize128 = 7U,
kDMA_BurstSize256 = 8U,
kDMA_BurstSize512 = 9U,
kDMA_BurstSize1024 = 10U }
DMA burst size.
- enum dma_trigger_burst_t {

kDMA_SingleTransfer = 0,
kDMA_LevelBurstTransfer = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer1 = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer2,
kDMA_EdgeBurstTransfer4,
kDMA_EdgeBurstTransfer8,
kDMA_EdgeBurstTransfer16,
kDMA_EdgeBurstTransfer32,
kDMA_EdgeBurstTransfer64,
kDMA_EdgeBurstTransfer128,
kDMA_EdgeBurstTransfer256,
kDMA_EdgeBurstTransfer512,
kDMA_EdgeBurstTransfer1024 }
DMA trigger burst.
- enum dma_burst_wrap_t {
kDMA_NoWrap = 0,
kDMA_SrcWrap = DMA_CHANNEL_CFG_SRCBURSTWRAP(1),
kDMA_DstWrap = DMA_CHANNEL_CFG_DSTBURSTWRAP(1),
kDMA_SrcAndDstWrap }
DMA burst wrapping.
- enum dma_transfer_type_t {
kDMA_MemoryToMemory = 0x0U,
kDMA_PeripheralToMemory,
kDMA_MemoryToPeripheral,
kDMA_StaticToStatic }
DMA transfer type.

Driver version

- #define FSL_DMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
DMA driver version.

DMA initialization and De-initialization

- void DMA_Init (DMA_Type *base)
Initializes DMA peripheral.
- void DMA_Deinit (DMA_Type *base)
Deinitializes DMA peripheral.
- void DMA_InstallDescriptorMemory (DMA_Type *base, void *addr)
Install DMA descriptor memory.

DMA Channel Operation

- static bool DMA_ChannelIsActive (DMA_Type *base, uint32_t channel)
Return whether DMA channel is processing transfer.
- static bool DMA_ChannelIsBusy (DMA_Type *base, uint32_t channel)
Return whether DMA channel is busy.
- static void DMA_EnableChannelInterrupts (DMA_Type *base, uint32_t channel)

Enables the interrupt source for the DMA transfer.
- static void DMA_DisableChannelInterrupts (DMA_Type ∗base, uint32_t channel)
  Disables the interrupt source for the DMA transfer.
- static void DMA_EnableChannel (DMA_Type ∗base, uint32_t channel)
  Enable DMA channel.
- static void DMA_DisableChannel (DMA_Type ∗base, uint32_t channel)
  Disable DMA channel.
- static void DMA_EnableChannelPeriphRq (DMA_Type ∗base, uint32_t channel)
  Set PERIPHREQEN of channel configuration register.
- static void DMA_DisableChannelPeriphRq (DMA_Type ∗base, uint32_t channel)
  Get PERIPHREQEN value of channel configuration register.
- void DMA_ConfigureChannelTrigger (DMA_Type ∗base, uint32_t channel, dma_channel_trigger_t ∗trigger)
  Set trigger settings of DMA channel.
- void DMA_SetChannelConfig (DMA_Type ∗base, uint32_t channel, dma_channel_trigger_t ∗trigger, bool isPeriph)
  set channel config.
- uint32_t DMA_GetRemainingBytes (DMA_Type ∗base, uint32_t channel)
  Gets the remaining bytes of the current DMA descriptor transfer.
- static void DMA_SetChannelPriority (DMA_Type ∗base, uint32_t channel, dma_priority_t priority)
  Set priority of channel configuration register.
- static dma_priority_t DMA_GetChannelPriority (DMA_Type ∗base, uint32_t channel)
  Get priority of channel configuration register.
- static void DMA_SetChannelConfigValid (DMA_Type ∗base, uint32_t channel)
  Set channel configuration valid.
- static void DMA_DoChannelSoftwareTrigger (DMA_Type ∗base, uint32_t channel)
  Do software trigger for the channel.
- static void DMA_LoadChannelTransferConfig (DMA_Type ∗base, uint32_t channel, uint32_t xfer)
  Load channel transfer configurations.
- void DMA_CreateDescriptor (dma_descriptor_t ∗desc, dma_xfercfg_t ∗xfercfg, void ∗srcAddr, void ∗dstAddr, void ∗nextDesc)
  Create application specific DMA descriptor to be used in a chain in transfer.
- void DMA_SetupDescriptor (dma_descriptor_t ∗desc, uint32_t xfercfg, void ∗srcStartAddr, void ∗dstStartAddr, void ∗nextDesc)
  setup dma descriptor
- void DMA_SetupChannelDescriptor (dma_descriptor_t ∗desc, uint32_t xfercfg, void ∗srcStartAddr, void ∗dstStartAddr, void ∗nextDesc, dma_burst_wrap_t wrapType, uint32_t burstSize)
  setup dma channel descriptor
- void DMA_LoadChannelDescriptor (DMA_Type ∗base, uint32_t channel, dma_descriptor_t ∗descriptor)
  load channel transfer decriptor.

DMA Transactional Operation

- void DMA_AbortTransfer (dma_handle_t ∗handle)
  Abort running transfer by handle.
- void DMA_CreateHandle (dma_handle_t ∗handle, DMA_Type ∗base, uint32_t channel)
  Creates the DMA handle.
- void DMA_SetCallback (dma_handle_t ∗handle, dma_callback callback, void ∗userData)
  Installs a callback function for the DMA transfer.

- void DMA_PrepareTransfer (dma_transfer_config_t *config, void *srcAddr, void *dstAddr, uint32_t byteWidth, uint32_t transferBytes, dma_transfer_type_t type, void *nextDesc)
  Prepares the DMA transfer structure.
- void DMA_PrepareChannelTransfer (dma_channel_config_t *config, void *srcStartAddr, void *dstStartAddr, uint32_t xferCfg, dma_transfer_type_t type, dma_channel_trigger_t *trigger, void *nextDesc)
  Prepare channel transfer configurations.
- status_t DMA_SubmitTransfer (dma_handle_t *handle, dma_transfer_config_t *config)
  Submits the DMA transfer request.
- void DMA_SubmitChannelTransferParameter (dma_handle_t *handle, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)
  Submit channel transfer paramter directly.
- void DMA_SubmitChannelDescriptor (dma_handle_t *handle, dma_descriptor_t *descriptor)
  Submit channel descriptor.
- status_t DMA_SubmitChannelTransfer (dma_handle_t *handle, dma_channel_config_t *config)
  Submits the DMA channel transfer request.
- void DMA_StartTransfer (dma_handle_t *handle)
  DMA start transfer.
- void DMA_IRQHandle (DMA_Type *base)
  DMA IRQ handler for descriptor transfer complete.

11.3 Data Structure Documentation

11.3.1 struct dma_descriptor_t

Data Fields

- volatile uint32_t xfercfg
  Transfer configuration.
- void * srcEndAddr
  Last source address of DMA transfer.
- void * dstEndAddr
  Last destination address of DMA transfer.
- void * linkToNextDesc
  Address of next DMA descriptor in chain.

11.3.2 struct dma_xfercfg_t

Data Fields

- bool valid
  Descriptor is ready to transfer.
- bool reload
  Reload channel configuration register after current descriptor is exhausted.
- bool swtrig
  Perform software trigger.
- bool clrtrig

Clear trigger.
- bool intA
  Raises IRQ when transfer is done and set IRQA status register flag.
- bool intB
  Raises IRQ when transfer is done and set IRQB status register flag.
- uint8_t byteWidth
  Byte width of data to transfer.
- uint8_t srcInc
  Increment source address by 'srcInc' x 'byteWidth'.
- uint8_t dstInc
  Increment destination address by 'dstInc' x 'byteWidth'.
- uint16_t transferCount
  Number of transfers.

11.3.2.0.0.4 Field Documentation

11.3.2.0.0.4.1 bool dma_xfercfg_t::swtrig

Transfer if fired when 'valid' is set

11.3.3 struct dma_channel_trigger_t

Data Fields

- dma_trigger_type_t type
  Select hardware trigger as edge triggered or level triggered.
- dma_trigger_burst_t burst
  Select whether hardware triggers cause a single or burst transfer.
- dma_burst_wrap_t wrap
  Select wrap type, source wrap or dest wrap, or both.

11.3.3.0.0.5 Field Documentation

11.3.3.0.0.5.1 dma_trigger_type_t dma_channel_trigger_t::type

11.3.3.0.0.5.2 dma_trigger_burst_t dma_channel_trigger_t::burst

11.3.3.0.0.5.3 dma_burst_wrap_t dma_channel_trigger_t::wrap

11.3.4 struct dma_channel_config_t

Data Fields

- void ∗ srcStartAddr
  Source data address.
- void ∗ dstStartAddr
  Destination data address.
- void ∗ nextDesc

Chain custom descriptor.
- uint32_t xferCfg
  channel transfer configurations
- dma_channel_trigger_t ∗ trigger
  DMA trigger type.
- bool isPeriph
  select the request type

11.3.5 struct dma_transfer_config_t

Data Fields

- uint8_t ∗ srcAddr
  Source data address.
- uint8_t ∗ dstAddr
  Destination data address.
- uint8_t ∗ nextDesc
  Chain custom descriptor.
- dma_xfercfg_t xfercfg
  Transfer options.
- bool isPeriph
  DMA transfer is driven by peripheral.

11.3.6 struct dma_handle_t

Data Fields

- dma_callback callback
  Callback function.
- void ∗ userData
  Callback function parameter.
- DMA_Type ∗ base
  DMA peripheral base address.
- uint8_t channel
  DMA channel number.

11.3.6.0.0.6 Field Documentation

11.3.6.0.0.6.1 dma_callback dma_handle_t::callback

Invoked when transfer of descriptor with interrupt flag finishes

11.4 Macro Definition Documentation

11.4.1 #define FSL_DMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

Version 2.4.0.

11.4.2 #define DMA_ALLOCATE_HEAD_DESCRIPTORS(*name, number*) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)

Parameters

name,allocate	decriptor name.
number,number	of descriptor to be allocated.

11.4.3 #define DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE(*name, number*) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)

Parameters

name,allocate	decriptor name.
number,number	of descriptor to be allocated.

11.4.4 #define DMA_ALLOCATE_LINK_DESCRIPTORS(*name, number*) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)

Parameters

name,allocate	decriptor name.
number,number	of descriptor to be allocated.

11.4.5 #define DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE(*name, number*) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SI-ZE)

Parameters

name,allocate	decriptor name.
number,number	of descriptor to be allocated.

11.4.6 #define DMA_DESCRIPTOR_END_ADDRESS(*start, inc, bytes, width*) ((void ∗)((uint32_t)(start) + inc ∗ bytes - inc ∗ width))

Parameters

start,start	address
inc,address	interleave size
bytes,transfer	bytes
width,transfer	width

11.4.7 #define DMA_CHANNEL_XFER(*reload*, *clrTrig*, *intA*, *intB*, *width*, *srcInc*, *dstInc*, *bytes*)

Value:

```
DMA_CHANNEL_XFERCFG_CFGVALID_MASK | DMA_CHANNEL_XFERCFG_RELOAD(reload) | DMA_CHANNEL_XFERCFG_CLRTRIG(
      clrTrig) | \
        DMA_CHANNEL_XFERCFG_SETINTA(intA) | DMA_CHANNEL_XFERCFG_SETINTB(intB) |
               \
        DMA_CHANNEL_XFERCFG_WIDTH(width == 4 ? 2 : (width - 1)) |
               \
        DMA_CHANNEL_XFERCFG_SRCINC(srcInc == 4 ? (srcInc - 1) : srcInc) |
               \
        DMA_CHANNEL_XFERCFG_DSTINC(dstInc == 4 ? (dstInc - 1) : dstInc) |
               \
        DMA_CHANNEL_XFERCFG_XFERCOUNT(bytes / width - 1)
```

Parameters

reload,true	is reload link descriptor after current exhaust, false is not
clrTrig,true	is clear trigger status, wait software trigger, false is not
intA,enable	interruptA
intB,enable	interruptB
width,transfer	width
srcInc,source	address interleave size
dst-Inc,destination	address interleave size
bytes,transfer	bytes

11.5 Typedef Documentation

11.5.1 typedef void(∗ dma_callback)(struct _dma_handle ∗handle, void ∗userData, bool transferDone, uint32_t intmode)

11.6 Enumeration Type Documentation

11.6.1 enum _dma_transfer_status

Enumerator

kStatus_DMA_Busy Channel is busy and can't handle the transfer request.

11.6.2 enum _dma_addr_interleave_size

Enumerator

kDMA_AddressInterleave0xWidth dma source/destination address no interleave
kDMA_AddressInterleave1xWidth dma source/destination address interleave 1xwidth
kDMA_AddressInterleave2xWidth dma source/destination address interleave 2xwidth
kDMA_AddressInterleave4xWidth dma source/destination address interleave 3xwidth

11.6.3 enum _dma_transfer_width

Enumerator

kDMA_Transfer8BitWidth dma channel transfer bit width is 8 bit
kDMA_Transfer16BitWidth dma channel transfer bit width is 16 bit
kDMA_Transfer32BitWidth dma channel transfer bit width is 32 bit

11.6.4 enum dma_priority_t

Enumerator

kDMA_ChannelPriority0 Highest channel priority - priority 0.
kDMA_ChannelPriority1 Channel priority 1.
kDMA_ChannelPriority2 Channel priority 2.
kDMA_ChannelPriority3 Channel priority 3.
kDMA_ChannelPriority4 Channel priority 4.
kDMA_ChannelPriority5 Channel priority 5.
kDMA_ChannelPriority6 Channel priority 6.
kDMA_ChannelPriority7 Lowest channel priority - priority 7.

11.6.5 enum dma_irq_t

Enumerator

kDMA_IntA DMA interrupt flag A.
kDMA_IntB DMA interrupt flag B.
kDMA_IntError DMA interrupt flag error.

11.6.6 enum dma_trigger_type_t

Enumerator

kDMA_NoTrigger Trigger is disabled.
kDMA_LowLevelTrigger Low level active trigger.
kDMA_HighLevelTrigger High level active trigger.
kDMA_FallingEdgeTrigger Falling edge active trigger.
kDMA_RisingEdgeTrigger Rising edge active trigger.

11.6.7 enum _dma_burst_size

Enumerator

kDMA_BurstSize1 burst size 1 transfer
kDMA_BurstSize2 burst size 2 transfer
kDMA_BurstSize4 burst size 4 transfer
kDMA_BurstSize8 burst size 8 transfer
kDMA_BurstSize16 burst size 16 transfer
kDMA_BurstSize32 burst size 32 transfer
kDMA_BurstSize64 burst size 64 transfer
kDMA_BurstSize128 burst size 128 transfer
kDMA_BurstSize256 burst size 256 transfer
kDMA_BurstSize512 burst size 512 transfer
kDMA_BurstSize1024 burst size 1024 transfer

11.6.8 enum dma_trigger_burst_t

Enumerator

kDMA_SingleTransfer Single transfer.
kDMA_LevelBurstTransfer Burst transfer driven by level trigger.
kDMA_EdgeBurstTransfer1 Perform 1 transfer by edge trigger.
kDMA_EdgeBurstTransfer2 Perform 2 transfers by edge trigger.

kDMA_EdgeBurstTransfer4 Perform 4 transfers by edge trigger.
kDMA_EdgeBurstTransfer8 Perform 8 transfers by edge trigger.
kDMA_EdgeBurstTransfer16 Perform 16 transfers by edge trigger.
kDMA_EdgeBurstTransfer32 Perform 32 transfers by edge trigger.
kDMA_EdgeBurstTransfer64 Perform 64 transfers by edge trigger.
kDMA_EdgeBurstTransfer128 Perform 128 transfers by edge trigger.
kDMA_EdgeBurstTransfer256 Perform 256 transfers by edge trigger.
kDMA_EdgeBurstTransfer512 Perform 512 transfers by edge trigger.
kDMA_EdgeBurstTransfer1024 Perform 1024 transfers by edge trigger.

11.6.9 enum dma_burst_wrap_t

Enumerator

kDMA_NoWrap Wrapping is disabled.
kDMA_SrcWrap Wrapping is enabled for source.
kDMA_DstWrap Wrapping is enabled for destination.
kDMA_SrcAndDstWrap Wrapping is enabled for source and destination.

11.6.10 enum dma_transfer_type_t

Enumerator

kDMA_MemoryToMemory Transfer from memory to memory (increment source and destination)
kDMA_PeripheralToMemory Transfer from peripheral to memory (increment only destination)
kDMA_MemoryToPeripheral Transfer from memory to peripheral (increment only source)
kDMA_StaticToStatic Peripheral to static memory (do not increment source or destination)

11.7 Function Documentation

11.7.1 void DMA_Init (DMA_Type * *base*)

This function enable the DMA clock, set descriptor table and enable DMA peripheral.

Parameters

base	DMA peripheral base address.

11.7.2 void DMA_Deinit (DMA_Type * *base*)

This function gates the DMA clock.

Parameters

base	DMA peripheral base address.

11.7.3 void DMA_InstallDescriptorMemory (DMA_Type * *base*, void * *addr*)

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, althrough current DMA driver has a default DMA descriptor buffer, but it support one DMA descriptor for one channel only.

Parameters

base	DMA base address.
addr	DMA descriptor address

11.7.4 static bool DMA_ChannelIsActive (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for active state, false otherwise.

11.7.5 static bool DMA_ChannelIsBusy (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.

channel	DMA channel number.

Returns

True for busy state, false otherwise.

11.7.6 static void DMA_EnableChannelInterrupts (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

11.7.7 static void DMA_DisableChannelInterrupts (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

11.7.8 static void DMA_EnableChannel (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

11.7.9 static void DMA_DisableChannel (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

11.7.10 static void DMA_EnableChannelPeriphRq (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

11.7.11 static void DMA_DisableChannelPeriphRq (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for enabled PeriphRq, false for disabled.

11.7.12 void DMA_ConfigureChannelTrigger (DMA_Type * *base*, uint32_t *channel*, dma_channel_trigger_t * *trigger*)

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
trigger	trigger configuration.

11.7.13 void DMA_SetChannelConfig (DMA_Type * *base*, uint32_t *channel*, dma_channel_trigger_t * *trigger*, bool *isPeriph*)

This function provide a interface to configure channel configuration reisters.

Parameters

base	DMA base address.
channel	DMA channel number.
trigger	channel configurations structure.
isPeriph	true is periph request, false is not.

11.7.14 uint32_t DMA_GetRemainingBytes (DMA_Type * *base*, uint32_t *channel*)

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

The number of bytes which have not been transferred yet.

11.7.15 static void DMA_SetChannelPriority (DMA_Type * *base*, uint32_t *channel*, dma_priority_t *priority*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
priority	Channel priority value.

11.7.16 static dma_priority_t DMA_GetChannelPriority (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

Channel priority value.

11.7.17 static void DMA_SetChannelConfigValid (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

11.7.18 static void DMA_DoChannelSoftwareTrigger (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

11.7.19 static void DMA_LoadChannelTransferConfig (DMA_Type * *base,* uint32_t *channel,* uint32_t *xfer*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
xfer	transfer configurations.

11.7.20 void DMA_CreateDescriptor (dma_descriptor_t * *desc,* dma_xfercfg_t * *xfercfg,* void * *srcAddr,* void * *dstAddr,* void * *nextDesc*)

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcAddr	Address of last item to transmit
dstAddr	Address of last item to receive.
nextDesc	Address of next descriptor in chain.

11.7.21 void DMA_SetupDescriptor (dma_descriptor_t * *desc,* uint32_t *xfercfg,* void * *srcStartAddr,* void * *dstStartAddr,* void * *nextDesc*)

Note: This function do not support configure wrap descriptor.

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcStartAddr	Start address of source address.
dstStartAddr	Start address of destination address.
nextDesc	Address of next descriptor in chain.

11.7.22 void DMA_SetupChannelDescriptor (dma_descriptor_t * *desc,* uint32_t *xfercfg,* void * *srcStartAddr,* void * *dstStartAddr,* void * *nextDesc,* dma_burst_wrap_t *wrapType,* uint32_t *burstSize*)

Note: This function support configure wrap descriptor.

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcStartAddr	Start address of source address.
dstStartAddr	Start address of destination address.
nextDesc	Address of next descriptor in chain.
wrapType	burst wrap type.
burstSize	burst size, reference _dma_burst_size.

11.7.23 void DMA_LoadChannelDescriptor (DMA_Type * *base*, uint32_t *channel*, dma_descriptor_t * *descriptor*)

This function can be used to load desscriptor to driver internal channel descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
*    DMA_Init(DMA0);
*    DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);
*    DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);
*    DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (
       dma_descriptor_t *)desc);
*    DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);
*    while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))
*    {}
*
```

Parameters

base	DMA base address.
channel	DMA channel.
descriptor	configured DMA descriptor.

11.7.24 void DMA_AbortTransfer (dma_handle_t * *handle*)

This function aborts DMA transfer specified by handle.

Parameters

handle	DMA handle pointer.

11.7.25 void DMA_CreateHandle (dma_handle_t * *handle*, DMA_Type * *base*, uint32_t *channel*)

This function is called if using transaction API for DMA. This function initializes the internal state of DMA handle.

Parameters

handle	DMA handle pointer. The DMA handle stores callback function and parameters.
base	DMA peripheral base address.
channel	DMA channel number.

11.7.26 void DMA_SetCallback (dma_handle_t ∗ *handle*, dma_callback *callback*, void ∗ *userData*)

This callback is called in DMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

handle	DMA handle pointer.
callback	DMA callback function pointer.
userData	Parameter for callback function.

11.7.27 void DMA_PrepareTransfer (dma_transfer_config_t ∗ *config,* void ∗ *srcAddr,* void ∗ *dstAddr,* uint32_t *byteWidth,* uint32_t *transferBytes,* dma_transfer_type_t *type,* void ∗ *nextDesc*)

Parameters

config	The user configuration structure of type dma_transfer_t.
srcAddr	DMA transfer source address.
dstAddr	DMA transfer destination address.
byteWidth	DMA transfer destination address width(bytes).
transferBytes	DMA transfer bytes to be transferred.
type	DMA transfer type.
nextDesc	Chain custom descriptor to transfer.

Note

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

11.7.28 void DMA_PrepareChannelTransfer (dma_channel_config_t ∗ *config,* void ∗ *srcStartAddr,* void ∗ *dstStartAddr,* uint32_t *xferCfg,* dma_transfer_type_t *type,* dma_channel_trigger_t ∗ *trigger,* void ∗ *nextDesc*)

This function used to prepare channel transfer configurations.

Parameters

config	Pointer to DMA channel transfer configuration structure.
srcStartAddr	source start address.
dstStartAddr	destination start address.
xferCfg	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
type	transfer type.
trigger	DMA channel trigger configurations.
nextDesc	address of next descriptor.

11.7.29 status_t DMA_SubmitTransfer (dma_handle_t ∗ *handle*, dma_transfer_config_t ∗ *config*)

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

handle	DMA handle pointer.
config	Pointer to DMA transfer configuration structure.

Return values

kStatus_DMA_Success	It means submit transfer request succeed.
kStatus_DMA_QueueFull	It means TCD queue is full. Submit transfer request is not allowed.
kStatus_DMA_Busy	It means the given channel is busy, need to submit request later.

11.7.30 void DMA_SubmitChannelTransferParameter (dma_handle_t ∗ *handle*, uint32_t *xfercfg*, void ∗ *srcStartAddr*, void ∗ *dstStartAddr*, void ∗ *nextDesc*)

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle,
   DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
```

```
bytes), srcStartAddr, dstStartAddr, NULL);
   DMA_StartTransfer(handle)
*
```

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   //define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
   DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelTransferParameter(handle,
      DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
   DMA_StartTransfer(handle);
*
```

Parameters

handle	Pointer to DMA handle.
xferCfg	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
srcStartAddr	source start address.
dstStartAddr	destination start address.
nextDesc	address of next descriptor.

11.7.31 void DMA_SubmitChannelDescriptor (dma_handle_t ∗ *handle*, dma_descriptor_t ∗ *descriptor*)

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this functiono is typical for the ping pong case:

1. for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```
   //define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
```

```
srcStartAddr, dstStartAddr, nextDesc0);
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelDescriptor(handle,  nextDesc0);
   DMA_StartTransfer(handle);
*
```

Parameters

handle	Pointer to DMA handle.
descriptor	descriptor to submit.

11.7.32 status_t DMA_SubmitChannelTransfer (dma_handle_t ∗ *handle,* dma_channel_config_t ∗ *config*)

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,NULL);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   //define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
   DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,nextDesc0);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

3. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
    //define link descriptor table in application with macro
    DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

    DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
       intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
    DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
       intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
    DMA_CreateHandle(handle, base, channel)
    DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
       trigger,nextDesc0);
    DMA_SubmitChannelTransfer(handle, config)
    DMA_StartTransfer(handle)
*
```

Parameters

handle	DMA handle pointer.
config	Pointer to DMA transfer configuration structure.

Return values

kStatus_DMA_Success	It means submit transfer request succeed.
kStatus_DMA_QueueFull	It means TCD queue is full. Submit transfer request is not allowed.
kStatus_DMA_Busy	It means the given channel is busy, need to submit request later.

11.7.33 void DMA_StartTransfer (dma_handle_t * *handle*)

This function enables the channel request. User can call this function after submitting the transfer request It will trigger transfer start with software trigger only when hardware trigger is not used.

Parameters

handle	DMA handle pointer.

11.7.34 void DMA_IRQHandle (DMA_Type * *base*)

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

Parameters

base	DMA base address.

Chapter 12
DMIC: Digital Microphone

12.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Digital Microphone (DMIC) module.

The DMIC driver is created to help the user more easily operate the DMIC module. This driver can be used to performed basic and advanced DMIC operations. The driver can be used to transfer data from DMIC to memory using DMA as well as in interrupt mode. The DMIC and DMA transfer in pingpong mode is preferred as DMIC is a streaming device.

12.2 Function groups

12.2.1 Initialization and deinitialization

This function group implements DMIC initialization and deinitialization API. DMIC_Init() function Enables the clock to the DMIC register interface. DMIC_Dinit() function Disables the clock to the DMIC register interface.

12.2.2 Configuration

This function group implements DMIC configration API. DMIC_ConfigIO()function configures the use of PDM (Pulse Density moulation) pins. DMIC_SetOperationMode()function configures the mode of operation either in DMA or in interrupt. DMIC_ConfigChannel() function configures the various property of a DMIC channel. DMIC_Use2fs()function configures the clock scaling used for PCM data output. DMIC_EnableChannnel() function enables a particualr DMIC channel. DMIC_FifoChannel() function configures FIFO settings for a DMIC channel.

12.2.3 DMIC Data and status

This function group implements the API to get data and status of DMIC FIFO. DMIC_FifoGetStatus() function gives the status of a DMIC FIFO. DMIC_ClearStatus() function clears the status of a DMIC FIFO. DMIC_FifoGetData() function gets data from a DMIC FIFO.

12.2.4 DMIC Interrupt Functions

DMIC_EnablebleIntCallback() enables the interrupt for the selected DMIC peripheral. DMIC_DisableIntCallback() disables the interrupt for the selected DMIC peripheral.

12.2.5 DMIC HWVAD Functions

This function group implements the API for HWVAD. DMIC_SetGainNoiseEstHwvad() Sets the gain value for the noise estimator. DMIC_SetGainSignalEstHwvad() Sets the gain value for the signal estimator. DMIC_SetFilterCtrlHwvad() Sets the HWVAD filter cutoff frequency parameter. DMIC_SetInputGainHwvad() Sets the input gain of HWVAD. DMIC_CtrlClrIntrHwvad() Clears HWVAD internal interrupt flag. DMIC_FilterResetHwvad() Resets HWVAD filters. DMIC_GetNoiseEnvlpEst() Gets the value from output of the filter z7.

12.2.6 DMIC HWVAD Interrupt Functions

DMIC_HwvadEnableIntCallback() enables the HWVAD interrupt for the selected DMIC peripheral. DMIC_HwvadDisableIntCallback() disables the HWVAD interrupt for the selected DMIC peripheral.

12.3 Typical use case

12.3.1 DMIC DMA Configuration

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

12.3.2 DMIC use case

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

Modules

- DMIC DMA Driver
- DMIC Driver

12.4 DMIC Driver

12.4.1 Overview

Files

- file fsl_dmic.h

Data Structures

- struct dmic_channel_config_t
  DMIC Channel configuration structure. More...

Typedefs

- typedef void(∗ dmic_callback_t)(void)
  DMIC Callback function.
- typedef void(∗ dmic_hwvad_callback_t)(void)
  HWVAD Callback function.

Enumerations

- enum _dmic_status {
  kStatus_DMIC_Busy = MAKE_STATUS(kStatusGroup_DMIC, 0),
  kStatus_DMIC_Idle = MAKE_STATUS(kStatusGroup_DMIC, 1),
  kStatus_DMIC_OverRunError = MAKE_STATUS(kStatusGroup_DMIC, 2),
  kStatus_DMIC_UnderRunError = MAKE_STATUS(kStatusGroup_DMIC, 3) }
  DMIC transfer status.
- enum operation_mode_t {
  kDMIC_OperationModeInterrupt = 1U,
  kDMIC_OperationModeDma = 2U }
  DMIC different operation modes.
- enum stereo_side_t {
  kDMIC_Left = 0U,
  kDMIC_Right = 1U }
  DMIC left/right values.
- enum pdm_div_t {

kDMIC_PdmDiv1 = 0U,
kDMIC_PdmDiv2 = 1U,
kDMIC_PdmDiv3 = 2U,
kDMIC_PdmDiv4 = 3U,
kDMIC_PdmDiv6 = 4U,
kDMIC_PdmDiv8 = 5U,
kDMIC_PdmDiv12 = 6U,
kDMIC_PdmDiv16 = 7U,
kDMIC_PdmDiv24 = 8U,
kDMIC_PdmDiv32 = 9U,
kDMIC_PdmDiv48 = 10U,
kDMIC_PdmDiv64 = 11U,
kDMIC_PdmDiv96 = 12U,
kDMIC_PdmDiv128 = 13U }
DMIC Clock pre-divider values.
- enum compensation_t {
kDMIC_CompValueZero = 0U,
kDMIC_CompValueNegativePoint16 = 1U,
kDMIC_CompValueNegativePoint15 = 2U,
kDMIC_CompValueNegativePoint13 = 3U }
Pre-emphasis Filter coefficient value for 2FS and 4FS modes.
- enum dc_removal_t {
kDMIC_DcNoRemove = 0U,
kDMIC_DcCut155 = 1U,
kDMIC_DcCut78 = 2U,
kDMIC_DcCut39 = 3U }
DMIC DC filter control values.
- enum dmic_io_t {
kDMIC_PdmDual = 0,
kDMIC_PdmStereo = 4 }
DMIC IO configiration.
- enum dmic_channel_t {
kDMIC_Channel0 = 0U,
kDMIC_Channel1 = 1U }
DMIC Channel number.
- enum _dmic_channel_mask {
kDMIC_EnableChannel0 = 1 << 0U,
kDMIC_EnableChannel1 = 1 << 1U }
DMIC Channel mask.
- enum dmic_phy_sample_rate_t {
kDMIC_PhyFullSpeed = 0U,
kDMIC_PhyHalfSpeed = 1U }
DMIC and decimator sample rates.

DMIC version

- #define FSL_DMIC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  DMIC driver version 2.1.1.

Initialization and deinitialization

- uint32_t DMIC_GetInstance (DMIC_Type ∗base)
  Get the DMIC instance from peripheral base address.
- void DMIC_Init (DMIC_Type ∗base)
  Turns DMIC Clock on.
- void DMIC_DeInit (DMIC_Type ∗base)
  Turns DMIC Clock off.
- void DMIC_ConfigIO (DMIC_Type ∗base, dmic_io_t config)
  Configure DMIC io.
- static void DMIC_SetIOCFG (DMIC_Type ∗base, uint32_t sel)
  Stereo PDM select.
- void DMIC_SetOperationMode (DMIC_Type ∗base, operation_mode_t mode)
  Set DMIC operating mode.
- void DMIC_Use2fs (DMIC_Type ∗base, bool use2fs)
  Configure Clock scaling.

Channel configuration

- void DMIC_CfgChannelDc (DMIC_Type ∗base, dmic_channel_t channel, dc_removal_t dc_cut_-level, uint32_t post_dc_gain_reduce, bool saturate16bit)
  Configure DMIC channel.
- void DMIC_ConfigChannel (DMIC_Type ∗base, dmic_channel_t channel, stereo_side_t side, dmic-_channel_config_t ∗channel_config)
  Configure DMIC channel.
- void DMIC_EnableChannnel (DMIC_Type ∗base, uint32_t channelmask)
  Enable a particualr channel.
- void DMIC_FifoChannel (DMIC_Type ∗base, uint32_t channel, uint32_t trig_level, uint32_t en-able, uint32_t resetn)
  Configure fifo settings for DMIC channel.
- static void DMIC_EnableChannelInterrupt (DMIC_Type ∗base, dmic_channel_t channel, bool en-able)
  Enable a particualr channel interrupt request.
- static void DMIC_EnableChannelDma (DMIC_Type ∗base, dmic_channel_t channel, bool enable)
  Enable a particualr channel dma request.
- static void DMIC_EnableChannelFifo (DMIC_Type ∗base, dmic_channel_t channel, bool enable)
  Enable a particualr channel fifo.
- static void DMIC_DoFifoReset (DMIC_Type ∗base, dmic_channel_t channel)
  Channel fifo reset.
- static uint32_t DMIC_FifoGetStatus (DMIC_Type ∗base, uint32_t channel)
  Get FIFO status.
- static void DMIC_FifoClearStatus (DMIC_Type ∗base, uint32_t channel, uint32_t mask)
  Clear FIFO status.

- static uint32_t DMIC_FifoGetData (DMIC_Type *base, uint32_t channel)
  Get FIFO data.
- static uint32_t DMIC_FifoGetAddress (DMIC_Type *base, uint32_t channel)
  Get FIFO address.

Register callback.

- void DMIC_EnableIntCallback (DMIC_Type *base, dmic_callback_t cb)
  Enable callback.
- void DMIC_DisableIntCallback (DMIC_Type *base, dmic_callback_t cb)
  Disable callback.

HWVAD

- static void DMIC_SetGainNoiseEstHwvad (DMIC_Type *base, uint32_t value)
  Sets the gain value for the noise estimator.
- static void DMIC_SetGainSignalEstHwvad (DMIC_Type *base, uint32_t value)
  Sets the gain value for the signal estimator.
- static void DMIC_SetFilterCtrlHwvad (DMIC_Type *base, uint32_t value)
  Sets the hwvad filter cutoff frequency parameter.
- static void DMIC_SetInputGainHwvad (DMIC_Type *base, uint32_t value)
  Sets the input gain of hwvad.
- static void DMIC_CtrlClrIntrHwvad (DMIC_Type *base, bool st10)
  Clears hwvad internal interrupt flag.
- static void DMIC_FilterResetHwvad (DMIC_Type *base, bool rstt)
  Resets hwvad filters.
- static uint16_t DMIC_GetNoiseEnvlpEst (DMIC_Type *base)
  Gets the value from output of the filter z7.
- void DMIC_HwvadEnableIntCallback (DMIC_Type *base, dmic_hwvad_callback_t vadcb)
  Enable hwvad callback.
- void DMIC_HwvadDisableIntCallback (DMIC_Type *base, dmic_hwvad_callback_t vadcb)
  Disable callback.

12.4.2 Data Structure Documentation

12.4.2.1 struct dmic_channel_config_t

Data Fields

- pdm_div_t divhfclk
  DMIC Clock pre-divider values.
- uint32_t osr
  oversampling rate(CIC decimation rate) for PCM
- int32_t gainshft
  4FS PCM data gain control
- compensation_t preac2coef

Pre-emphasis Filter coefficient value for 2FS.
- compensation_t preac4coef
  Pre-emphasis Filter coefficient value for 4FS.
- dc_removal_t dc_cut_level
  DMIC DC filter control values.
- uint32_t post_dc_gain_reduce
  Fine gain adjustment in the form of a number of bits to downshift.
- dmic_phy_sample_rate_t sample_rate
  DMIC and decimator sample rates.
- bool saturate16bit
  Selects 16-bit saturation.

12.4.2.1.0.7 Field Documentation

12.4.2.1.0.7.1 dc_removal_t dmic_channel_config_t::dc_cut_level

12.4.2.1.0.7.2 bool dmic_channel_config_t::saturate16bit

0 means results roll over if out range and do not saturate. 1 means if the result overflows, it saturates at 0xFFFF for positive overflow and 0x8000 for negative overflow.

12.4.3 Macro Definition Documentation

12.4.3.1 #define FSL_DMIC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

12.4.4 Typedef Documentation

12.4.4.1 typedef void(* dmic_callback_t)(void)

12.4.4.2 typedef void(* dmic_hwvad_callback_t)(void)

12.4.5 Enumeration Type Documentation

12.4.5.1 enum _dmic_status

Enumerator

kStatus_DMIC_Busy DMIC is busy.
kStatus_DMIC_Idle DMIC is idle.
kStatus_DMIC_OverRunError DMIC over run Error.
kStatus_DMIC_UnderRunError DMIC under run Error.

12.4.5.2 enum operation_mode_t

Enumerator

kDMIC_OperationModeInterrupt Interrupt mode.
kDMIC_OperationModeDma DMA mode.

12.4.5.3 enum stereo_side_t

Enumerator

kDMIC_Left Left Stereo channel.
kDMIC_Right Right Stereo channel.

12.4.5.4 enum pdm_div_t

Enumerator

kDMIC_PdmDiv1 DMIC pre-divider set in divide by 1.
kDMIC_PdmDiv2 DMIC pre-divider set in divide by 2.
kDMIC_PdmDiv3 DMIC pre-divider set in divide by 3.
kDMIC_PdmDiv4 DMIC pre-divider set in divide by 4.
kDMIC_PdmDiv6 DMIC pre-divider set in divide by 6.
kDMIC_PdmDiv8 DMIC pre-divider set in divide by 8.
kDMIC_PdmDiv12 DMIC pre-divider set in divide by 12.
kDMIC_PdmDiv16 DMIC pre-divider set in divide by 16.
kDMIC_PdmDiv24 DMIC pre-divider set in divide by 24.
kDMIC_PdmDiv32 DMIC pre-divider set in divide by 32.
kDMIC_PdmDiv48 DMIC pre-divider set in divide by 48.
kDMIC_PdmDiv64 DMIC pre-divider set in divide by 64.
kDMIC_PdmDiv96 DMIC pre-divider set in divide by 96.
kDMIC_PdmDiv128 DMIC pre-divider set in divide by 128.

12.4.5.5 enum compensation_t

Enumerator

kDMIC_CompValueZero Compensation 0.
kDMIC_CompValueNegativePoint16 Compensation -0.16.
kDMIC_CompValueNegativePoint15 Compensation -0.15.
kDMIC_CompValueNegativePoint13 Compensation -0.13.

12.4.5.6 enum dc_removal_t

Enumerator

kDMIC_DcNoRemove Flat response no filter.
kDMIC_DcCut155 Cut off Frequency is 155 Hz.
kDMIC_DcCut78 Cut off Frequency is 78 Hz.
kDMIC_DcCut39 Cut off Frequency is 39 Hz.

12.4.5.7 enum dmic_io_t

Enumerator

kDMIC_PdmDual Two separate pairs of PDM wires.
kDMIC_PdmStereo Stereo data0.

12.4.5.8 enum dmic_channel_t

Enumerator

kDMIC_Channel0 DMIC channel 0.
kDMIC_Channel1 DMIC channel 1.

12.4.5.9 enum _dmic_channel_mask

Enumerator

kDMIC_EnableChannel0 DMIC channel 0 mask.
kDMIC_EnableChannel1 DMIC channel 1 mask.

12.4.5.10 enum dmic_phy_sample_rate_t

Enumerator

kDMIC_PhyFullSpeed Decimator gets one sample per each chosen clock edge of PDM interface.
kDMIC_PhyHalfSpeed PDM clock to Microphone is halved, decimator receives each sample twice.

12.4.6 Function Documentation

12.4.6.1 uint32_t DMIC_GetInstance (DMIC_Type * *base*)

Parameters

base	DMIC peripheral base address.

Returns

DMIC instance.

12.4.6.2 void DMIC_Init (DMIC_Type * *base*)

Parameters

base	: DMIC base

Returns

Nothing

12.4.6.3 void DMIC_DeInit (DMIC_Type * *base*)

Parameters

base	: DMIC base

Returns

Nothing

12.4.6.4 void DMIC_ConfigIO (DMIC_Type * *base*, dmic_io_t *config*)

Parameters

base	: The base address of DMIC interface
config	: DMIC io configuration

Returns

Nothing

12.4.6.5 static void DMIC_SetIOCFG (DMIC_Type * *base*, uint32_t *sel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
sel	: Reference dmic_io_t, can be a single or combination value of dmic_io_t.

Returns

Nothing

12.4.6.6 void DMIC_SetOperationMode (DMIC_Type * *base*, operation_mode_t *mode*)

Parameters

base	: The base address of DMIC interface
mode	: DMIC mode

Returns

Nothing

12.4.6.7 void DMIC_Use2fs (DMIC_Type * *base*, bool *use2fs*)

Parameters

base	: The base address of DMIC interface
use2fs	: clock scaling

Returns

Nothing

12.4.6.8 void DMIC_CfgChannelDc (DMIC_Type * *base*, dmic_channel_t *channel*, dc_removal_t *dc_cut_level*, uint32_t *post_dc_gain_reduce*, bool *saturate16bit*)

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
dc_cut_level	: dc_removal_t, Cut off Frequency
post_dc_gain_-reduce	: Fine gain adjustment in the form of a number of bits to downshift.
saturate16bit	: If selects 16-bit saturation.

12.4.6.9 void DMIC_ConfigChannel (DMIC_Type ∗ *base*, dmic_channel_t *channel*, stereo_side_t *side*, dmic_channel_config_t ∗ *channel_config*)

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
side	: stereo_side_t, choice of left or right
channel_config	: Channel configuration

Returns

Nothing

12.4.6.10 void DMIC_EnableChannnel (DMIC_Type ∗ *base*, uint32_t *channelmask*)

Parameters

base	: The base address of DMIC interface
channel-mask,reference	_dmic_channel_mask

Returns

Nothing

12.4.6.11 void DMIC_FifoChannel (DMIC_Type ∗ *base*, uint32_t *channel*, uint32_t *trig_level*, uint32_t *enable*, uint32_t *resetn*)

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
trig_level	: FIFO trigger level
enable	: FIFO level
resetn	: FIFO reset

Returns

Nothing

12.4.6.12 static void DMIC_EnableChannelInterrupt (DMIC_Type * *base*, dmic_channel_t *channel*, bool *enable*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection

12.4.6.13 static void DMIC_EnableChannelDma (DMIC_Type * *base*, dmic_channel_t *channel*, bool *enable*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection
enable	: true is enable, false is disable

12.4.6.14 static void DMIC_EnableChannelFifo (DMIC_Type * *base*, dmic_channel_t *channel*, bool *enable*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection
enable	: true is enable, false is disable

12.4.6.15 static void DMIC_DoFifoReset (DMIC_Type * *base*, dmic_channel_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection

12.4.6.16 static uint32_t DMIC_FifoGetStatus (DMIC_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel

Returns

FIFO status

12.4.6.17 static void DMIC_FifoClearStatus (DMIC_Type * *base*, uint32_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
mask	: Bits to be cleared

Returns

FIFO status

12.4.6.18 static uint32_t DMIC_FifoGetData (DMIC_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel

Returns

FIFO data

12.4.6.19 static uint32_t DMIC_FifoGetAddress (DMIC_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel

Returns

FIFO data

12.4.6.20 void DMIC_EnableIntCallback (DMIC_Type * *base*, dmic_callback_t *cb*)

This function enables the interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

base	Base address of the DMIC peripheral.
cb	callback Pointer to store callback function.

Return values

None.	

12.4.6.21 void DMIC_DisableIntCallback (DMIC_Type * *base*, dmic_callback_t *cb*)

This function disables the interrupt for the selected DMIC peripheral.

Parameters

base	Base address of the DMIC peripheral.
cb	callback Pointer to store callback function..

Return values

None.	

12.4.6.22 static void DMIC_SetGainNoiseEstHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	gain value for the noise estimator.

Return values

None.	

12.4.6.23 static void DMIC_SetGainSignalEstHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	gain value for the signal estimator.

Return values

None.	

12.4.6.24 static void DMIC_SetFilterCtrlHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	cut off frequency value.

Return values

None.	

12.4.6.25 static void DMIC_SetInputGainHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	input gain value for hwvad.

Return values

None.	

12.4.6.26 static void DMIC_CtrlClrIntrHwvad (DMIC_Type * *base,* bool *st10*) [inline], [static]

Parameters

base	DMIC base pointer
st10	bit value.

Return values

None.	

12.4.6.27 static void DMIC_FilterResetHwvad (DMIC_Type * *base,* bool *rstt*) [inline], [static]

Parameters

base	DMIC base pointer
rstt	Reset bit value.

Return values

None.	

12.4.6.28 static uint16_t DMIC_GetNoiseEnvlpEst (DMIC_Type ∗ *base*) [inline], [static]

Parameters

base	DMIC base pointer

Return values

output	of filter z7.

12.4.6.29 void DMIC_HwvadEnableIntCallback (DMIC_Type ∗ *base*, dmic_hwvad_callback_t *vadcb*)

This function enables the hwvad interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

base	Base address of the DMIC peripheral.
vadcb	callback Pointer to store callback function.

Return values

None.	

12.4.6.30 void DMIC_HwvadDisableIntCallback (DMIC_Type ∗ *base*, dmic_hwvad_callback_t *vadcb*)

This function disables the hwvad interrupt for the selected DMIC peripheral.

Parameters

base	Base address of the DMIC peripheral.
vadcb	callback Pointer to store callback function..

Return values

None.	

12.5 DMIC DMA Driver

12.5.1 Overview

Files

- file fsl_dmic_dma.h

Data Structures

- struct dmic_transfer_t
  DMIC transfer structure. More...
- struct dmic_dma_handle_t
  DMIC DMA handle. More...

Typedefs

- typedef void(∗ dmic_dma_transfer_callback_t)(DMIC_Type ∗base, dmic_dma_handle_t ∗handle, status_t status, void ∗userData)
  DMIC transfer callback function.

DMIC DMA version

- #define FSL_DMIC_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  DMIC DMA driver version 2.1.1.

DMA transactional

- status_t DMIC_TransferCreateHandleDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle, dmic_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗rxDmaHandle)
  Initializes the DMIC handle which is used in transactional functions.
- status_t DMIC_TransferReceiveDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle, dmic_transfer_t ∗xfer, uint32_t dmic_channel)
  Receives data using DMA.
- void DMIC_TransferAbortReceiveDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t DMIC_TransferGetReceiveCountDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle, uint32_t ∗count)
  Get the number of bytes that have been received.
- void DMIC_InstallDMADescriptorMemory (dmic_dma_handle_t ∗handle, void ∗linkAddr, size_t linkNum)
  Install DMA descriptor memory.

12.5.2 Data Structure Documentation

12.5.2.1 struct dmic_transfer_t

Data Fields

- void * data
  The buffer of data to be transfer.
- uint8_t dataWidth
  DMIC support 16bit/32bit.
- size_t dataSize
  The byte count to be transfer.
- uint8_t dataAddrInterleaveSize
  destination address interleave size
- struct _dmic_transfer * linkTransfer
  use to support link transfer

12.5.2.1.0.8 Field Documentation

12.5.2.1.0.8.1 void* dmic_transfer_t::data

12.5.2.1.0.8.2 size_t dmic_transfer_t::dataSize

12.5.2.2 struct _dmic_dma_handle

Data Fields

- DMIC_Type * base
  DMIC peripheral base address.
- dma_handle_t * rxDmaHandle
  The DMA RX channel used.
- dmic_dma_transfer_callback_t callback
  Callback function.
- void * userData
  DMIC callback function parameter.
- size_t transferSize
  Size of the data to receive.
- volatile uint8_t state
  Internal state of DMIC DMA transfer.
- dma_descriptor_t * desLink
  descriptor pool pointer
- size_t linkNum
  number of descriptor in descriptors pool

12.5.2.2.0.9 Field Documentation

12.5.2.2.0.9.1 DMIC_Type* dmic_dma_handle_t::base

12.5.2.2.0.9.2 dma_handle_t* dmic_dma_handle_t::rxDmaHandle

12.5.2.2.0.9.3 dmic_dma_transfer_callback_t dmic_dma_handle_t::callback

12.5.2.2.0.9.4 void* dmic_dma_handle_t::userData

12.5.2.2.0.9.5 size_t dmic_dma_handle_t::transferSize

12.5.3 Macro Definition Documentation

12.5.3.1 #define FSL_DMIC_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

12.5.4 Typedef Documentation

12.5.4.1 typedef void(* dmic_dma_transfer_callback_t)(DMIC_Type *base, dmic_dma_handle_t *handle, status_t status, void *userData)

12.5.5 Function Documentation

12.5.5.1 status_t DMIC_TransferCreateHandleDMA (DMIC_Type * *base*, dmic_dma_handle_t * *handle*, dmic_dma_transfer_callback_t *callback*, void * *userData*, dma_handle_t * *rxDmaHandle*)

Parameters

base	DMIC peripheral base address.
handle	Pointer to dmic_dma_handle_t structure.
callback	Callback function.
userData	User data.
rxDmaHandle	User-requested DMA handle for RX DMA transfer.

12.5.5.2 status_t DMIC_TransferReceiveDMA (DMIC_Type ∗ *base,* dmic_dma_handle_t ∗ *handle,* dmic_transfer_t ∗ *xfer,* uint32_t *dmic_channel*)

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	USART peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	DMIC DMA transfer structure. See dmic_transfer_t.
dmic_channel	DMIC start channel number.

Return values

kStatus_Success	

12.5.5.3 void DMIC_TransferAbortReceiveDMA (DMIC_Type ∗ *base,* dmic_dma_handle_t ∗ *handle*)

This function aborts the received data using DMA.

Parameters

base	DMIC peripheral base address
handle	Pointer to dmic_dma_handle_t structure

12.5.5.4 status_t DMIC_TransferGetReceiveCountDMA (DMIC_Type ∗ *base,* dmic_dma_handle_t ∗ *handle,* uint32_t ∗ *count*)

This function gets the number of bytes that have been received.

Parameters

base	DMIC peripheral base address.
handle	DMIC handle pointer.
count	Receive bytes count.

Return values

kStatus_NoTransferIn-Progress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter count;

12.5.5.5 void DMIC_InstallDMADescriptorMemory (dmic_dma_handle_t ∗ *handle,* void ∗ *linkAddr,* size_t *linkNum*)

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, it should be called after DMI-C_TransferCreateHandleDMA. User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

handle	Pointer to DMA channel transfer handle.
linkAddr	DMA link descriptor address.
num	DMA link descriptor number.

Chapter 13
FLASH: Flash driver

13.1 Overview

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

Files

- file flash_header.h
- file fsl_flash.h

Data Structures

- struct IMG_HEADER_T
  Image header. *More...*
- struct BOOT_BLOCK_T
  Boot block. *More...*
- struct flash_config_t
  Flash configuration information. *More...*

Macros

- #define FLASH_FAIL (1 << 0)
  FLASH INT_STATUS register definitions.
- #define FLASH_FAIL (1 << 0)
  FLASH INT_STATUS register definitions.
- #define FLASH_ERR (1 << 1)
  Illegal command.
- #define FLASH_ERR (1 << 1)
  Illegal command.
- #define FLASH_DONE (1 << 2)
  Command complete.
- #define FLASH_DONE (1 << 2)
  Command complete.
- #define FLASH_ECC_ERR (1 << 3)
  ECC error detected.
- #define FLASH_ECC_ERR (1 << 3)
  ECC error detected.

Enumerations

- enum flash_status_t {
  kStatus_FLASH_Success = FLASH_DONE,
  kStatus_FLASH_Fail = FLASH_DONE | FLASH_FAIL,
  kStatus_FLASH_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
  kStatus_FLASH_AlignmentError = MAKE_STATUS(kStatusGroup_FLASH, 6),
  kStatus_FLASH_EccError = FLASH_DONE | FLASH_ECC_ERR,
  kStatus_FLASH_Error = FLASH_DONE | FLASH_ERR }
- enum flash_read_mode_t { , FLASH_ReadModeNormalEccOff = (FLASH_READ_MODE_NORMAL << FLASH_READ_MODE_SHIFT)|(1<<FLASH_READ_MODE_ECC_OFF_SHIFT) }

Functions

- void FLASH_Init (FLASH_Type ∗pFLASH)
  Enable the FLASH.
- void FLASH_Powerdown (FLASH_Type ∗pFLASH)
  Power down the FLASH.
- int FLASH_Wait (FLASH_Type ∗pFLASH)
  Wait for FLASH command to complete.
- int FLASH_Erase (FLASH_Type ∗pFLASH, uint8_t ∗pu8Start, uint8_t ∗pu8End)
  Erase page.
- int FLASH_ErasePages (FLASH_Type ∗pFLASH, uint32_t u32StartPage, uint32_t u32PageCount)
  Erase multiple pages.
- int FLASH_BlankCheck (FLASH_Type ∗pFLASH, uint8_t ∗pu8Start, uint8_t ∗pu8End)
  Page Blank check.
- int FLASH_MarginCheck (FLASH_Type ∗pFLASH, uint8_t ∗pu8Start, uint8_t ∗pu8End)
  Margin Check.
- int FLASH_Program (FLASH_Type ∗pFLASH, uint32_t ∗pu32Start, uint32_t ∗pu32Data, uint32_t u32Length)
  Program page.
- int FLASH_Checksum (FLASH_Type ∗pFLASH, uint8_t ∗pu8Start, uint8_t ∗pu8End, uint32_t au32Checksum[4])
  Page Checksum.
- int FLASH_Read (FLASH_Type ∗pFLASH, uint8_t ∗pu8Start, uint32_t u32ReadMode, uint32_t au32Data[4])
  Read flash word (16 byte worth of data)
- void FLASH_SetReadMode (FLASH_Type ∗pFLASH, bool freq_48M_not_32M)
  Configure the flash wait state depending of the elwe mode and CPU frequency. When the CPU clock frequency is decreased, the Set Read command shall be called after the frequency change. When the CPU clock frequency is increased, the Set Read command shall be called before the frequency change.
- void FLASH_CalculateChecksum (const uint32_t ∗input, size_t nb_128b_words, uint32_t ∗misr, int init)
  Calculate checksum using the same checksum algorithm as the CMD_CHECKSUM implementation of the Flash controller. When executed over a 512 byte page (page size) must return the same value as FLASH_Checksum.
- int FLASH_ConfigPageVerifyPageChecksum (const uint32_t ∗page_buffer, uint32_t ∗misr)
  Calculate checksum over page (N-2) aka CONFIG page and check it matches the expected value.
- int FLASH_ConfigPageVerifyGpoChecksum (const uint32_t ∗page_buffer, uint32_t ∗misr)
  Calculate checksum over GPO array of CONFIG page and check it matches the expected value.

- void FLASH_ConfigPageUpdate (uint32_t *page_ram_buffer, uint32_t *gpo_chksum, uint32_t *page_chksum)
  Configure the flash wait state depending of the elwe mode and CPU frequency. When the CPU clock frequency is decreased, the Set Read command shall be called after the frequency change. When the CPU clock frequency is increased, the Set Read command shall be called before the frequency change.
- int FLASH_GetStatus (FLASH_Type *pFLASH)
  Return unfiltered FLASH INT_STATUS. In normal operation FLASH_DONE rises systematically but other status bits may rise at the same time or have risen before to notify of an error. Usually testing the value returned by FLASH_Wait is sufficionet but in some special cases the raw value may be needed.

Variables

- uint32_t IMG_HEADER_T::vectors [NUMBER_CCSUM_VECTORS]
  critical vectors protected by csum
- uint32_t IMG_HEADER_T::vectorCsum
  csum of vectors 0-7
- uint32_t IMG_HEADER_T::imageSignature
  image signature
- uint32_t IMG_HEADER_T::bootBlockOffset
  offset of boot block structure
- uint32_t IMG_HEADER_T::header_crc
  the CRC of header
- uint32_t BOOT_BLOCK_T::header_marker
  Image header marker should always be set to 0xbb0110bb+/-2.
- uint32_t BOOT_BLOCK_T::img_type
  Image check type, with or without optional CRC.
- uint32_t BOOT_BLOCK_T::target_addr
  Target address.
- uint32_t BOOT_BLOCK_T::img_len
  Image length or the length of image CRC check should be done.
- uint32_t BOOT_BLOCK_T::stated_size
  max size of any subsequent image : AppSize0 = 2 x stated_size
- uint32_t BOOT_BLOCK_T::certificate_offset
  Offset of the certificate list.
- uint32_t BOOT_BLOCK_T::compatibility_offset
  Offset of the compatibility list.
- uint32_t BOOT_BLOCK_T::version
  Image version for multi-image support.

13.2 Data Structure Documentation

13.2.1 struct IMG_HEADER_T

Be very cautious when modifying the IMG_HEADER_T and the BOOT_BLOCK_T structures (alignment) as these structures are used in the image_tool.py (which does not take care of alignment).

Data Fields

- uint32_t vectors [NUMBER_CCSUM_VECTORS]

critical vectors protected by csum
- uint32_t vectorCsum
  csum of vectors 0-7
- uint32_t imageSignature
  image signature
- uint32_t bootBlockOffset
  offset of boot block structure
- uint32_t header_crc
  the CRC of header

13.2.2 struct BOOT_BLOCK_T

For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0_SE4a and ADC0_SE4b are the different channels that share the same channel number.

Data Fields

- uint32_t header_marker
  Image header marker should always be set to 0xbb0110bb+/-2.
- uint32_t img_type
  Image check type, with or without optional CRC.
- uint32_t target_addr
  Target address.
- uint32_t img_len
  Image length or the length of image CRC check should be done.
- uint32_t stated_size
  max size of any subsequent image : AppSize0 = 2 x stated_size
- uint32_t certificate_offset
  Offset of the certificate list.
- uint32_t compatibility_offset
  Offset of the compatibility list.
- uint32_t version
  Image version for multi-image support.

13.2.3 struct flash_config_t

An instance of this structure is allocated by the user of the flash driver and at initialization.

Data Fields

- uint32_t PFlashBlockBase
  A base address of the first PFlash block.
- uint32_t PFlashTotalSize
  The size of the combined PFlash block.
- uint32_t PFlashSectorSize

The size in bytes of a sector of PFlash.

13.2.3.0.0.10 Field Documentation

13.2.3.0.0.10.1 uint32_t flash_config_t::PFlashTotalSize

13.2.3.0.0.10.2 uint32_t flash_config_t::PFlashSectorSize

13.3 Macro Definition Documentation

13.3.1 #define FLASH_FAIL (1 << 0)

FLASH INT_ENABLE register definitions.

Command failed

13.3.2 #define FLASH_FAIL (1 << 0)

FLASH INT_ENABLE register definitions.

Command failed

13.4 Enumeration Type Documentation

13.4.1 enum flash_status_t

Enumerator

kStatus_FLASH_Success flash operation is successful
kStatus_FLASH_Fail flash operation is not successful
kStatus_FLASH_InvalidArgument Invalid argument.
kStatus_FLASH_AlignmentError Alignment Error.
kStatus_FLASH_EccError ECC error detected.
kStatus_FLASH_Error Illegal command.

13.4.2 enum flash_read_mode_t

Enumerator

FLASH_ReadModeNormalEccOff flash operation is not successful

13.5 Function Documentation

13.5.1 void FLASH_Init (FLASH_Type * *pFLASH*)

Parameters

pFLASH	Pointer to selected FLASHx peripheral

Returns

Nothing

13.5.2 void FLASH_Powerdown (FLASH_Type * *pFLASH*)

Parameters

pFLASH	Pointer to selected FLASHx peripheral

Returns

Nothing

13.5.3 int FLASH_Wait (FLASH_Type * *pFLASH*)

Parameters

pFLASH	Pointer to selected FLASHx peripheral

Returns

INT_STATUS with ECC_ERR bit masked out

13.5.4 int FLASH_Erase (FLASH_Type * *pFLASH,* uint8_t * *pu8Start,* uint8_t * *pu8End*)

Parameters

	pFLASH	Pointer to selected FLASH peripheral
in	*pu8Start*	Start address with page to inspect
in	*pu8End*	End address (included in check)

Returns

INT_STATUS with ECC_ERR bit masked out

See Also

flash_status_t

13.5.5 int FLASH_ErasePages (FLASH_Type * *pFLASH,* uint32_t *u32StartPage,* uint32_t *u32PageCount*)

Parameters

	pFLASH	Pointer to selected FLASH peripheral
in	*u32StartPage*	Index of page to start erasing from
in	*u32PageCount*	Number of pages to erase

Returns

INT_STATUS with ECC_ERR bit masked out

See Also

flash_status_t

13.5.6 int FLASH_BlankCheck (FLASH_Type * *pFLASH,* uint8_t * *pu8Start,* uint8_t * *pu8End*)

Parameters

	pFLASH	Pointer to selected FLASH peripheral
in	*pu8Start*	Start address with page to inspect
in	*pu8End*	End address (included in check)

Returns

INT_STATUS with ECC_ERR bit masked out

See Also

flash_status_t

13.5.7 int FLASH_MarginCheck (FLASH_Type * *pFLASH*, uint8_t * *pu8Start*, uint8_t * *pu8End*)

Parameters

	pFLASH	Pointer to selected FLASH peripheral
in	*pu8Start*	Start address with page to inspect
in	*pu8End*	End address (included in check)

Returns

INT_STATUS with ECC_ERR bit masked out

See Also

flash_status_t

13.5.8 int FLASH_Program (FLASH_Type * *pFLASH*, uint32_t * *pu32Start*, uint32_t * *pu32Data*, uint32_t *u32Length*)

Parameters

in	*pFLASH*	Pointer to selected FLASH peripheral
out	*pu32Start*	Pointer location that must be programmed in flash
in	*pu32Data*	Pointer to source buffer being written to flash
in	*u32Length*	Number of bytes to be programmed

Returns

INT_STATUS with ECC_ERR bit masked out

See Also

flash_status_t

13.5.9 int FLASH_Checksum (FLASH_Type ∗ *pFLASH,* uint8_t ∗ *pu8Start,* uint8_t ∗ *pu8End,* uint32_t *au32Checksum[4]*)

Parameters

	pFLASH	Pointer to selected FLASH peripheral
in	*pu8Start*	Pointer to data within starting page page checksum must be computed
in	*pu8End*	Pointer to data whose page is the last of the checksum calculation
out	*au32Checksum*	Four 32bit word array to store checksum calculation result

Returns

INT_STATUS with ECC_ERR bit masked out

See Also

flash_status_t

13.5.10 int FLASH_Read (FLASH_Type ∗ *pFLASH,* uint8_t ∗ *pu8Start,* uint32_t *u32ReadMode,* uint32_t *au32Data[4]*)

Parameters

	pFLASH	Pointer to selected FLASH peripheral
in	*pu8Start*	Pointer to data to be read
in	*u32ReadMode*	Read mode see also flash_read_mode_t
out	*au32Data*	Four 32bit word array to store read result

Returns

INT_STATUS with ECC_ERR bit masked out

See Also

flash_status_t

13.5.11 void FLASH_SetReadMode (FLASH_Type * *pFLASH*, bool *freq_48M_not_32M*)

Parameters

pFLASH	Pointer to selected FLASHx peripheral
freq_48M_not-_32M	CPU clock frequency @48MHz - lower or equal to 32Mhz if 0

Returns

Nothing

13.5.12 void FLASH_CalculateChecksum (const uint32_t * *input*, size_t *nb_128b_words*, uint32_t * *misr*, int *init*)

Parameters

in	*input*	Pointer to data over which checksum calculation must be executed.
in	*nb_128b_-words*	Number of 16 byte words on flash.
out	*misr*	Pointer on a four 32bit word array to store calculated checksum.
in	*init*	Set to true to clear the misr buffer.

Returns

Nothing

13.5.13 int FLASH_ConfigPageVerifyPageChecksum (const uint32_t * *page_buffer*, uint32_t * *misr*)

Parameters

in	*page_buffer*	Pointer to data over which checksum calculation must be executed.
out	*misr*	Pointer on a four 32bit word array to store calculated checksum. Note: this buffer is only useful for debugging purposes.

Returns

Result of the page checksum verification:
- 0: Verify successfully.
- -1: Verification failed.

13.5.14 int FLASH_ConfigPageVerifyGpoChecksum (const uint32_t * *page_buffer*, uint32_t * *misr*)

Parameters

in	*page_buffer*	Pointer to data over which checksum calculation must be executed.
out	*misr*	Pointer on a 4 32bit word array to store calculated checksum. Note: this buffer is only useful for debugging purposes.

Returns

Result of the GPO array checksum verification:
- 0: Verify successfully.
- -1: Verification failed.

13.5.15 void FLASH_ConfigPageUpdate (uint32_t * *page_ram_buffer,* uint32_t * *gpo_chksum,* uint32_t * *page_chksum*)

Parameters

page_ram_-buffer	Pointer to RAM page buffer in which the read-modify-write of page (N-2) is performed
gpo_chksum	Pointer on a four 32bit word array to store calculated checksum.
page_chksum	Pointer on a four 32bit word array to store calculated checksum.

Returns

Nothing

13.5.16 int FLASH_GetStatus (FLASH_Type * *pFLASH*)

Parameters

pFLASH	Pointer to selected FLASHx peripheral.

Returns

INT_STATUS raw value.

See Also

flash_status_t

13.6 Variable Documentation

13.6.1 uint32_t BOOT_BLOCK_T::img_len

For faster boot application could set a smaller length than actual image. For Secure boot images, this MUST be the entire image length

Chapter 14
FLEXCOMM: FLEXCOMM Driver

14.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FLEXCOMM drivers for the FLEXCOMM module of MCUXpresso SDK devices.

Modules

- FLEXCOMM Driver

14.2 FLEXCOMM Driver

14.2.1 Overview

Typedefs

- typedef void(∗ flexcomm_irq_handler_t)(void ∗base, void ∗handle)
  Typedef for interrupt handler.

Enumerations

- enum FLEXCOMM_PERIPH_T {
  FLEXCOMM_PERIPH_NONE,
  FLEXCOMM_PERIPH_USART,
  FLEXCOMM_PERIPH_SPI,
  FLEXCOMM_PERIPH_I2C,
  FLEXCOMM_PERIPH_I2S_TX,
  FLEXCOMM_PERIPH_I2S_RX }
  FLEXCOMM peripheral modes.

Functions

- uint32_t FLEXCOMM_GetInstance (void ∗base)
  Returns instance number for FLEXCOMM module with given base address.
- status_t FLEXCOMM_Init (void ∗base, FLEXCOMM_PERIPH_T periph)
  Initializes FLEXCOMM and selects peripheral mode according to the second parameter.
- void FLEXCOMM_SetIRQHandler (void ∗base, flexcomm_irq_handler_t handler, void ∗handle)
  Sets IRQ handler for given FLEXCOMM module.

Variables

- IRQn_Type const kFlexcommIrqs [ ]
  Array with IRQ number for each FLEXCOMM module.

Driver version

- #define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  FlexCOMM driver version 2.0.2.

14.2.2 Macro Definition Documentation

14.2.2.1 #define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

14.2.3 Typedef Documentation

14.2.3.1 typedef void(* flexcomm_irq_handler_t)(void *base, void *handle)

14.2.4 Enumeration Type Documentation

14.2.4.1 enum FLEXCOMM_PERIPH_T

Enumerator

FLEXCOMM_PERIPH_NONE No peripheral.
FLEXCOMM_PERIPH_USART USART peripheral.
FLEXCOMM_PERIPH_SPI SPI Peripheral.
FLEXCOMM_PERIPH_I2C I2C Peripheral.
FLEXCOMM_PERIPH_I2S_TX I2S TX Peripheral.
FLEXCOMM_PERIPH_I2S_RX I2S RX Peripheral.

14.2.5 Function Documentation

14.2.5.1 uint32_t FLEXCOMM_GetInstance (void * *base*)

14.2.5.2 status_t FLEXCOMM_Init (void * *base*, FLEXCOMM_PERIPH_T *periph*)

14.2.5.3 void FLEXCOMM_SetIRQHandler (void * *base*, flexcomm_irq_handler_t *handler*, void * *handle*)

It is used by drivers register IRQ handler according to FLEXCOMM mode

14.2.6 Variable Documentation

14.2.6.1 IRQn_Type const kFlexcommIrqs[ ]

Chapter 15
I2C: Inter-Integrated Circuit Driver

15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C 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 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 asynchronous transfer. This means that the functions I2C_MasterTransferNonBlocking() set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

15.2 Typical use case

15.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Send start and slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
      kI2C_Write/kI2C_Read);

/* Wait address sent out. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR)) & kI2C_IntPendingFlag))
{

}

if(status & kI2C_ReceiveNakFlag)
{
    return kStatus_I2C_Nak;
}
```

```
result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE);

if(result)
{
    /* If error occours, send STOP. */
    I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
    return result;
}

while(!(I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR) & kI2C_IntPendingFlag))
{

}

/* Wait all data sent out, send STOP. */
I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
```

15.2.2 Master Operation in interrupt transactional method

```
i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle,
      status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
      i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &
      masterXfer);

/*  Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

15.2.3 Master Operation in DMA transactional method

```
i2c_master_dma_handle_t g_m_dma_handle;
dma_handle_t dmaHandle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_dma_handle_t *handle,
      status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

DMA_EnableChannel(EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);
DMA_CreateHandle(&dmaHandle, EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);

I2C_MasterTransferCreateHandleDMA(EXAMPLE_I2C_MASTER_BASEADDR, &
      g_m_dma_handle, i2c_master_callback, NULL, &dmaHandle);
I2C_MasterTransferDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, &masterXfer);

/*  Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

15.2.4 Slave Operation in functional method

```
i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
       mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

/* Wait address match. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_SLAVE_BASEADDR)) & kI2C_AddressMatchFlag))
{

}
```

```
/* Slave transmit, master reading from slave. */
if (status & kI2C_TransferDirectionFlag)
{
    result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}
else
{
    I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}

return result;
```

15.2.5 Slave Operation in interrupt transactional method

```
i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
      userData)
{
    switch (xfer->event)
    {
        /*  Transmit request */
        case kI2C_SlaveTransmitEvent:
            /*  Update information for transmit process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Receive request */
        case kI2C_SlaveReceiveEvent:
            /*  Update information for received process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Transfer done */
        case kI2C_SlaveCompletionEvent:
            g_SlaveCompletionFlag = true;
            break;

        default:
            g_SlaveCompletionFlag = true;
            break;
    }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
       mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      kI2C_SlaveCompletionEvent);

/*  Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
```

```
g_SlaveCompletionFlag = false;
```

Modules

- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver
- I2C Master Driver
- I2C Slave Driver

15.3 I2C Driver

15.3.1 Overview

Files

- file fsl_i2c.h

Macros

- #define I2C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.
- #define I2C_STAT_MSTCODE_IDLE (0)
  Master Idle State Code.
- #define I2C_STAT_MSTCODE_RXREADY (1)
  Master Receive Ready State Code.
- #define I2C_STAT_MSTCODE_TXREADY (2)
  Master Transmit Ready State Code.
- #define I2C_STAT_MSTCODE_NACKADR (3)
  Master NACK by slave on address State Code.
- #define I2C_STAT_MSTCODE_NACKDAT (4)
  Master NACK by slave on data State Code.

Enumerations

- enum _i2c_status {
  kStatus_I2C_Busy = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 0),
  kStatus_I2C_Idle = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 1),
  kStatus_I2C_Nak,
  kStatus_I2C_InvalidParameter,
  kStatus_I2C_BitError = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 4),
  kStatus_I2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 5),
  kStatus_I2C_NoTransferInProgress,
  kStatus_I2C_DmaRequestFail = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 7) ,
  kStatus_I2C_Timeout = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 10),
  kStatus_I2C_Addr_Nak = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 11) }
  I2C status return codes.

Driver version

- #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))
  I2C driver version 2.0.6.

15.3.2 Macro Definition Documentation

15.3.2.1 #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))

15.3.2.2 #define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

15.3.3 Enumeration Type Documentation

15.3.3.1 enum _i2c_status

Enumerator

kStatus_I2C_Busy The master is already performing a transfer.
kStatus_I2C_Idle The slave driver is idle.
kStatus_I2C_Nak The slave device sent a NAK in response to a byte.
kStatus_I2C_InvalidParameter Unable to proceed due to invalid parameter.
kStatus_I2C_BitError Transferred bit was not seen on the bus.
kStatus_I2C_ArbitrationLost Arbitration lost error.
kStatus_I2C_NoTransferInProgress Attempt to abort a transfer when one is not in progress.
kStatus_I2C_DmaRequestFail DMA request failed.
kStatus_I2C_Timeout Timeout poling status flags.
kStatus_I2C_Addr_Nak NAK received for Address.

15.4 I2C Master Driver

15.4.1 Overview

Data Structures

- struct i2c_master_config_t
  Structure with settings to initialize the I2C master module. More...
- struct i2c_master_transfer_t
  Non-blocking transfer descriptor structure. More...
- struct i2c_master_handle_t
  Driver handle for master non-blocking APIs. More...

Typedefs

- typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Master completion callback function pointer type.

Enumerations

- enum _i2c_master_flags {
  kI2C_MasterPendingFlag = I2C_STAT_MSTPENDING_MASK,
  kI2C_MasterArbitrationLostFlag,
  kI2C_MasterStartStopErrorFlag }
  I2C master peripheral flags.
- enum i2c_direction_t {
  kI2C_Write = 0U,
  kI2C_Read = 1U }
  Direction of master and slave transfers.
- enum _i2c_master_transfer_flags {
  kI2C_TransferDefaultFlag = 0x00U,
  kI2C_TransferNoStartFlag = 0x01U,
  kI2C_TransferRepeatedStartFlag = 0x02U,
  kI2C_TransferNoStopFlag = 0x04U }
  Transfer option flags.
- enum _i2c_transfer_states
  States for the state machine used by transactional APIs.

Initialization and deinitialization

- void I2C_MasterGetDefaultConfig (i2c_master_config_t ∗masterConfig)
  Provides a default configuration for the I2C master peripheral.
- void I2C_MasterInit (I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)

Initializes the I2C master peripheral.
- void I2C_MasterDeinit (I2C_Type *base)
  Deinitializes the I2C master peripheral.
- uint32_t I2C_GetInstance (I2C_Type *base)
  Returns an instance number given a base address.
- static void I2C_MasterReset (I2C_Type *base)
  Performs a software reset.
- static void I2C_MasterEnable (I2C_Type *base, bool enable)
  Enables or disables the I2C module as master.

Status

- static uint32_t I2C_GetStatusFlags (I2C_Type *base)
  Gets the I2C status flags.
- static void I2C_MasterClearStatusFlags (I2C_Type *base, uint32_t statusMask)
  Clears the I2C master status flag state.

Interrupts

- static void I2C_EnableInterrupts (I2C_Type *base, uint32_t interruptMask)
  Enables the I2C master interrupt requests.
- static void I2C_DisableInterrupts (I2C_Type *base, uint32_t interruptMask)
  Disables the I2C master interrupt requests.
- static uint32_t I2C_GetEnabledInterrupts (I2C_Type *base)
  Returns the set of currently enabled I2C master interrupt requests.

Bus operations

- void I2C_MasterSetBaudRate (I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the I2C bus frequency for master transactions.
- static bool I2C_MasterGetBusIdleState (I2C_Type *base)
  Returns whether the bus is idle.
- status_t I2C_MasterStart (I2C_Type *base, uint8_t address, i2c_direction_t direction)
  Sends a START on the I2C bus.
- status_t I2C_MasterStop (I2C_Type *base)
  Sends a STOP signal on the I2C bus.
- static status_t I2C_MasterRepeatedStart (I2C_Type *base, uint8_t address, i2c_direction_t direction)
  Sends a REPEATED START on the I2C bus.
- status_t I2C_MasterWriteBlocking (I2C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
  Performs a polling send transfer on the I2C bus.
- status_t I2C_MasterReadBlocking (I2C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)
  Performs a polling receive transfer on the I2C bus.
- status_t I2C_MasterTransferBlocking (I2C_Type *base, i2c_master_transfer_t *xfer)
  Performs a master polling transfer on the I2C bus.

Non-blocking

- void I2C_MasterTransferCreateHandle (I2C_Type ∗base, i2c_master_handle_t ∗handle, i2c_master_transfer_callback_t callback, void ∗userData)
  Creates a new handle for the I2C master non-blocking APIs.
- status_t I2C_MasterTransferNonBlocking (I2C_Type ∗base, i2c_master_handle_t ∗handle, i2c_master_transfer_t ∗xfer)
  Performs a non-blocking transaction on the I2C bus.
- status_t I2C_MasterTransferGetCount (I2C_Type ∗base, i2c_master_handle_t ∗handle, size_t ∗count)
  Returns number of bytes transferred so far.
- status_t I2C_MasterTransferAbort (I2C_Type ∗base, i2c_master_handle_t ∗handle)
  Terminates a non-blocking I2C master transmission early.

IRQ handler

- void I2C_MasterTransferHandleIRQ (I2C_Type ∗base, i2c_master_handle_t ∗handle)
  Reusable routine to handle master interrupts.

15.4.2 Data Structure Documentation

15.4.2.1 struct i2c_master_config_t

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the I2C_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.
- uint32_t baudRate_Bps
  Desired baud rate in bits per second.
- bool enableTimeout
  Enable internal timeout function.

15.4.2.1.0.11 Field Documentation

15.4.2.1.0.11.1 bool i2c_master_config_t::enableMaster

15.4.2.1.0.11.2 uint32_t i2c_master_config_t::baudRate_Bps

15.4.2.1.0.11.3 bool i2c_master_config_t::enableTimeout

15.4.2.2 struct _i2c_master_transfer

I2C master transfer typedef.

This structure is used to pass transaction parameters to the I2C_MasterTransferNonBlocking() API.

Data Fields

- uint32_t flags
  Bit mask of options for the transfer.
- uint16_t slaveAddress
  The 7-bit slave address.
- i2c_direction_t direction
  Either kI2C_Read or kI2C_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.

15.4.2.2.0.12 Field Documentation

15.4.2.2.0.12.1 uint32_t i2c_master_transfer_t::flags

See enumeration _i2c_master_transfer_flags for available options. Set to 0 or kI2C_TransferDefaultFlag for normal transfers.

15.4.2.2.0.12.2 uint16_t i2c_master_transfer_t::slaveAddress

15.4.2.2.0.12.3 i2c_direction_t i2c_master_transfer_t::direction

15.4.2.2.0.12.4 uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

15.4.2.2.0.12.5 size_t i2c_master_transfer_t::subaddressSize

Maximum size is 4 bytes.

15.4.2.2.0.12.6 void* i2c_master_transfer_t::data

15.4.2.2.0.12.7 size_t i2c_master_transfer_t::dataSize

15.4.2.3 struct _i2c_master_handle

I2C master handle typedef.

Note

The contents of this structure are private and subject to change.

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint32_t transferCount
  Indicates progress of the transfer.
- uint32_t remainingBytes
  Remaining byte count in current state.
- uint8_t * buf
  Buffer pointer for current state.
- i2c_master_transfer_t transfer
  Copy of the current transfer info.
- i2c_master_transfer_callback_t completionCallback
  Callback function pointer.
- void * userData
  Application data passed to callback.

15.4.2.3.0.13 Field Documentation

15.4.2.3.0.13.1 uint8_t i2c_master_handle_t::state

15.4.2.3.0.13.2 uint32_t i2c_master_handle_t::remainingBytes

15.4.2.3.0.13.3 uint8_t* i2c_master_handle_t::buf

15.4.2.3.0.13.4 i2c_master_transfer_t i2c_master_handle_t::transfer

15.4.2.3.0.13.5 i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

15.4.2.3.0.13.6 void* i2c_master_handle_t::userData

15.4.3 Typedef Documentation

15.4.3.1 typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base, i2c_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 I2C_MasterTransferCreateHandle().

Parameters

base	The I2C 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.

15.4.4 Enumeration Type Documentation

15.4.4.1 enum _i2c_master_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI2C_MasterPendingFlag The I2C module is waiting for software interaction.
kI2C_MasterArbitrationLostFlag The arbitration of the bus was lost. There was collision on the bus
kI2C_MasterStartStopErrorFlag There was an error during start or stop phase of the transaction.

15.4.4.2 enum i2c_direction_t

Enumerator

kI2C_Write Master transmit.
kI2C_Read Master receive.

15.4.4.3 enum _i2c_master_transfer_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the _i2c_master_transfer::flags field.

Enumerator

kI2C_TransferDefaultFlag Transfer starts with a start signal, stops with a stop signal.
kI2C_TransferNoStartFlag Don't send a start condition, address, and sub address.
kI2C_TransferRepeatedStartFlag Send a repeated start condition.
kI2C_TransferNoStopFlag Don't send a stop condition.

15.4.4.4 enum _i2c_transfer_states

15.4.5 Function Documentation

15.4.5.1 void I2C_MasterGetDefaultConfig (i2c_master_config_t * *masterConfig*)

This function provides the following default configuration for the I2C master peripheral:

```
*   masterConfig->enableMaster              = true;
*   masterConfig->baudRate_Bps              = 100000U;
*   masterConfig->enableTimeout             = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with I2C_MasterInit().

Parameters

out	*masterConfig*	User provided configuration structure for default values. Refer to i2c_-master_config_t.

15.4.5.2 void I2C_MasterInit (I2C_Type * *base*, const i2c_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

base	The I2C peripheral base address.
masterConfig	User provided peripheral configuration. Use I2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
srcClock_Hz	Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

15.4.5.3 void I2C_MasterDeinit (I2C_Type * *base*)

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

15.4.5.4 uint32_t I2C_GetInstance (I2C_Type * *base*)

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

base	The I2C peripheral base address.

Returns

I2C instance number starting from 0.

15.4.5.5 static void I2C_MasterReset (I2C_Type * *base*) [inline], [static]

Restores the I2C master peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

15.4.5.6 static void I2C_MasterEnable (I2C_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	The I2C peripheral base address.
enable	Pass true to enable or false to disable the specified I2C as master.

15.4.5.7 static uint32_t I2C_GetStatusFlags (I2C_Type * *base*) [inline], [static]

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I2C peripheral base address.

Returns

State of the status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i2c_master_flags

15.4.5.8 static void I2C_MasterClearStatusFlags (I2C_Type ∗ *base,* uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared:

- kI2C_MasterArbitrationLostFlag
- kI2C_MasterStartStopErrorFlag

Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i2c_master_flags enumerators OR'd together. You may pass the result of a previous call to I2C_GetStatusFlags().

See Also

_i2c_master_flags.

15.4.5.9 static void I2C_EnableInterrupts (I2C_Type ∗ *base,* uint32_t *interruptMask*) [inline], [static]

Parameters

base	The I2C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _i2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

15.4.5.10 static void I2C_DisableInterrupts (I2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	The I2C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _i2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

15.4.5.11 static uint32_t I2C_GetEnabledInterrupts (I2C_Type * *base*) [inline], [static]

Parameters

base	The I2C peripheral base address.

Returns

A bitmask composed of _i2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

15.4.5.12 void I2C_MasterSetBaudRate (I2C_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

The I2C 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.

Parameters

base	The I2C peripheral base address.
srcClock_Hz	I2C functional clock frequency in Hertz.
baudRate_Bps	Requested bus frequency in bits per second.

15.4.5.13 static bool I2C_MasterGetBusIdleState (I2C_Type * *base*) [inline], [static]

Requires the master mode to be enabled.

Parameters

base	The I2C peripheral base address.

Return values

true	Bus is busy.
false	Bus is idle.

15.4.5.14 status_t I2C_MasterStart (I2C_Type * *base*, uint8_t *address*, i2c_direction_t *direction*)

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

base	I2C peripheral base pointer
address	7-bit slave device address.
direction	Master transfer directions(transmit/receive).

Return values

kStatus_Success	Successfully send the start signal.
kStatus_I2C_Busy	Current bus is busy.

15.4.5.15 status_t I2C_MasterStop (I2C_Type * *base*)

Return values

kStatus_Success	Successfully send the stop signal.
kStatus_I2C_Timeout	Send stop signal failed, timeout.

15.4.5.16 static status_t I2C_MasterRepeatedStart (I2C_Type * *base*, uint8_t *address*, i2c_direction_t *direction*) [inline], [static]

Parameters

base	I2C peripheral base pointer
address	7-bit slave device address.
direction	Master transfer directions(transmit/receive).

Return values

kStatus_Success	Successfully send the start signal.
kStatus_I2C_Busy	Current bus is busy but not occupied by current I2C master.

15.4.5.17 status_t I2C_MasterWriteBlocking (I2C_Type * *base*, const void * *txBuff*, size_t *txSize*, uint32_t *flags*)

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_I2C_Nak.

Parameters

base	The I2C peripheral base address.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.
flags	Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

kStatus_Success	Data was sent successfully.
kStatus_I2C_Busy	Another master is currently utilizing the bus.
kStatus_I2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_I2C_Arbitration-Lost	Arbitration lost error.

15.4.5.18 status_t I2C_MasterReadBlocking (I2C_Type ∗ *base*, void ∗ *rxBuff*, size_t *rxSize*, uint32_t *flags*)

Parameters

base	The I2C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.
flags	Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

kStatus_Success	Data was received successfully.
kStatus_I2C_Busy	Another master is currently utilizing the bus.
kStatus_I2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_I2C_Arbitration-Lost	Arbitration lost error.

15.4.5.19 status_t I2C_MasterTransferBlocking (I2C_Type ∗ *base*, i2c_master_transfer_t ∗ *xfer*)

Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

base	I2C peripheral base address.
xfer	Pointer to the transfer structure.

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Busy	Previous transmission still not finished.
kStatus_I2C_Timeout	Transfer error, wait signal timeout.
kStatus_I2C_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

15.4.5.20 void I2C_MasterTransferCreateHandle (I2C_Type * *base*, i2c_master_handle_t * *handle*, i2c_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 I2C_MasterTransferAbort() API shall be called.

Parameters

	base	The I2C peripheral base address.
`out`	*handle*	Pointer to the I2C master driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

15.4.5.21 status_t I2C_MasterTransferNonBlocking (I2C_Type * *base*, i2c_master_handle_t * *handle*, i2c_master_transfer_t * *xfer*)

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.
xfer	The pointer to the transfer descriptor.

Return values

kStatus_Success	The transaction was started successfully.
kStatus_I2C_Busy	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

15.4.5.22 status_t I2C_MasterTransferGetCount (I2C_Type * *base*, i2c_master_handle_t * *handle*, size_t * *count*)

Parameters

	base	The I2C peripheral base address.
	handle	Pointer to the I2C master driver handle.
out	*count*	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_I2C_Busy	

15.4.5.23 status_t I2C_MasterTransferAbort (I2C_Type * *base*, i2c_master_handle_t * *handle*)

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.

Return values

kStatus_Success	A transaction was successfully aborted.
kStatus_I2C_Timeout	Timeout during polling for flags.

15.4.5.24 void I2C_MasterTransferHandleIRQ (I2C_Type * *base*, i2c_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 I2C peripheral base address.
handle	Pointer to the I2C master driver handle.

15.5 I2C Slave Driver

15.5.1 Overview

Data Structures

- struct i2c_slave_address_t
  Data structure with 7-bit Slave address and Slave address disable. More...
- struct i2c_slave_config_t
  Structure with settings to initialize the I2C slave module. More...
- struct i2c_slave_transfer_t
  I2C slave transfer structure. More...
- struct i2c_slave_handle_t
  I2C slave handle structure. More...

Typedefs

- typedef void(∗ i2c_slave_transfer_callback_t)(I2C_Type ∗base, volatile i2c_slave_transfer_t ∗transfer, void ∗userData)
  Slave event callback function pointer type.

Enumerations

- enum _i2c_slave_flags {
  kI2C_SlavePendingFlag = I2C_STAT_SLVPENDING_MASK,
  kI2C_SlaveNotStretching,
  kI2C_SlaveSelected = I2C_STAT_SLVSEL_MASK,
  kI2C_SaveDeselected }
  I2C slave peripheral flags.
- enum i2c_slave_address_register_t {
  kI2C_SlaveAddressRegister0 = 0U,
  kI2C_SlaveAddressRegister1 = 1U,
  kI2C_SlaveAddressRegister2 = 2U,
  kI2C_SlaveAddressRegister3 = 3U }
  I2C slave address register.
- enum i2c_slave_address_qual_mode_t {
  kI2C_QualModeMask = 0U,
  kI2C_QualModeExtend }
  I2C slave address match options.
- enum i2c_slave_bus_speed_t
  I2C slave bus speed options.
- enum i2c_slave_transfer_event_t {

kI2C_SlaveAddressMatchEvent = 0x01U,
kI2C_SlaveTransmitEvent = 0x02U,
kI2C_SlaveReceiveEvent = 0x04U,
kI2C_SlaveCompletionEvent = 0x20U,
kI2C_SlaveDeselectedEvent,
kI2C_SlaveAllEvents }
Set of events sent to the callback for non blocking slave transfers.
- enum i2c_slave_fsm_t
  I2C slave software finite state machine states.

Slave initialization and deinitialization

- void I2C_SlaveGetDefaultConfig (i2c_slave_config_t ∗slaveConfig)
  Provides a default configuration for the I2C slave peripheral.
- status_t I2C_SlaveInit (I2C_Type ∗base, const i2c_slave_config_t ∗slaveConfig, uint32_t srcClock-_Hz)
  Initializes the I2C slave peripheral.
- void I2C_SlaveSetAddress (I2C_Type ∗base, i2c_slave_address_register_t addressRegister, uint8_t address, bool addressDisable)
  Configures Slave Address n register.
- void I2C_SlaveDeinit (I2C_Type ∗base)
  Deinitializes the I2C slave peripheral.
- static void I2C_SlaveEnable (I2C_Type ∗base, bool enable)
  Enables or disables the I2C module as slave.

Slave status

- static void I2C_SlaveClearStatusFlags (I2C_Type ∗base, uint32_t statusMask)
  Clears the I2C status flag state.

Slave bus operations

- status_t I2C_SlaveWriteBlocking (I2C_Type ∗base, const uint8_t ∗txBuff, size_t txSize)
  Performs a polling send transfer on the I2C bus.
- status_t I2C_SlaveReadBlocking (I2C_Type ∗base, uint8_t ∗rxBuff, size_t rxSize)
  Performs a polling receive transfer on the I2C bus.

Slave non-blocking

- void I2C_SlaveTransferCreateHandle (I2C_Type ∗base, i2c_slave_handle_t ∗handle, i2c_slave_-transfer_callback_t callback, void ∗userData)
  Creates a new handle for the I2C slave non-blocking APIs.
- status_t I2C_SlaveTransferNonBlocking (I2C_Type ∗base, i2c_slave_handle_t ∗handle, uint32_t eventMask)

Starts accepting slave transfers.
- status_t I2C_SlaveSetSendBuffer (I2C_Type ∗base, volatile i2c_slave_transfer_t ∗transfer, const void ∗txData, size_t txSize, uint32_t eventMask)
  Starts accepting master read from slave requests.
- status_t I2C_SlaveSetReceiveBuffer (I2C_Type ∗base, volatile i2c_slave_transfer_t ∗transfer, void ∗rxData, size_t rxSize, uint32_t eventMask)
  Starts accepting master write to slave requests.
- static uint32_t I2C_SlaveGetReceivedAddress (I2C_Type ∗base, volatile i2c_slave_transfer_t ∗transfer)
  Returns the slave address sent by the I2C master.
- void I2C_SlaveTransferAbort (I2C_Type ∗base, i2c_slave_handle_t ∗handle)
  Aborts the slave non-blocking transfers.
- status_t I2C_SlaveTransferGetCount (I2C_Type ∗base, i2c_slave_handle_t ∗handle, size_t ∗count)
  Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

Slave IRQ handler

- void I2C_SlaveTransferHandleIRQ (I2C_Type ∗base, i2c_slave_handle_t ∗handle)
  Reusable routine to handle slave interrupts.

15.5.2 Data Structure Documentation

15.5.2.1 struct i2c_slave_address_t

Data Fields

- uint8_t address
  7-bit Slave address SLVADR.
- bool addressDisable
  Slave address disable SADISABLE.

15.5.2.1.0.14 Field Documentation

15.5.2.1.0.14.1 uint8_t i2c_slave_address_t::address

15.5.2.1.0.14.2 bool i2c_slave_address_t::addressDisable

15.5.2.2 struct i2c_slave_config_t

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the I2C_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

- i2c_slave_address_t address0
  Slave's 7-bit address and disable.
- i2c_slave_address_t address1
  Alternate slave 7-bit address and disable.
- i2c_slave_address_t address2
  Alternate slave 7-bit address and disable.
- i2c_slave_address_t address3
  Alternate slave 7-bit address and disable.
- i2c_slave_address_qual_mode_t qualMode
  Qualify mode for slave address 0.
- uint8_t qualAddress
  Slave address qualifier for address 0.
- i2c_slave_bus_speed_t busSpeed
  Slave bus speed mode.
- bool enableSlave
  Enable slave mode.

15.5.2.2.0.15 Field Documentation

15.5.2.2.0.15.1 i2c_slave_address_t i2c_slave_config_t::address0

15.5.2.2.0.15.2 i2c_slave_address_t i2c_slave_config_t::address1

15.5.2.2.0.15.3 i2c_slave_address_t i2c_slave_config_t::address2

15.5.2.2.0.15.4 i2c_slave_address_t i2c_slave_config_t::address3

15.5.2.2.0.15.5 i2c_slave_address_qual_mode_t i2c_slave_config_t::qualMode

15.5.2.2.0.15.6 uint8_t i2c_slave_config_t::qualAddress

15.5.2.2.0.15.7 i2c_slave_bus_speed_t i2c_slave_config_t::busSpeed

If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The busSpeed value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the busSpeed mode is unknown at compile time, use the longest data setup time kI2C_SlaveStandardMode (250 ns)

15.5.2.2.0.15.8 bool i2c_slave_config_t::enableSlave

15.5.2.3 struct i2c_slave_transfer_t

Data Fields

- i2c_slave_handle_t * handle
  Pointer to handle that contains this transfer.
- i2c_slave_transfer_event_t event

Reason the callback is being invoked.
- uint8_t receivedAddress
  Matching address send by master.
- uint32_t eventMask
  Mask of enabled events.
- uint8_t ∗ rxData
  Transfer buffer for receive data.
- const uint8_t ∗ txData
  Transfer buffer for transmit data.
- size_t txSize
  Transfer size.
- size_t rxSize
  Transfer size.
- size_t transferredCount
  Number of bytes transferred during this transfer.
- status_t completionStatus
  Success or error code describing how the transfer completed.

15.5.2.3.0.16 Field Documentation

15.5.2.3.0.16.1 i2c_slave_handle_t∗ i2c_slave_transfer_t::handle

15.5.2.3.0.16.2 i2c_slave_transfer_event_t i2c_slave_transfer_t::event

15.5.2.3.0.16.3 uint8_t i2c_slave_transfer_t::receivedAddress

7-bits plus R/nW bit0

15.5.2.3.0.16.4 uint32_t i2c_slave_transfer_t::eventMask

15.5.2.3.0.16.5 size_t i2c_slave_transfer_t::transferredCount

15.5.2.3.0.16.6 status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

15.5.2.4 struct _i2c_slave_handle

I2C slave handle typedef.

Note

The contents of this structure are private and subject to change.

Data Fields

- volatile i2c_slave_transfer_t transfer
  I2C slave transfer.
- volatile bool isBusy

Whether transfer is busy.
- volatile i2c_slave_fsm_t slaveFsm
  slave transfer state machine.
- i2c_slave_transfer_callback_t callback
  Callback function called at transfer event.
- void * userData
  Callback parameter passed to callback.

15.5.2.4.0.17 Field Documentation

15.5.2.4.0.17.1 volatile i2c_slave_transfer_t i2c_slave_handle_t::transfer

15.5.2.4.0.17.2 volatile bool i2c_slave_handle_t::isBusy

15.5.2.4.0.17.3 volatile i2c_slave_fsm_t i2c_slave_handle_t::slaveFsm

15.5.2.4.0.17.4 i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

15.5.2.4.0.17.5 void* i2c_slave_handle_t::userData

15.5.3 Typedef Documentation

15.5.3.1 typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *userData)

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I2C_-SlaveSetCallback() function after you have created a handle.

Parameters

base	Base address for the I2C 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.

15.5.4 Enumeration Type Documentation

15.5.4.1 enum _i2c_slave_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI2C_SlavePendingFlag The I2C module is waiting for software interaction.
kI2C_SlaveNotStretching Indicates whether the slave is currently stretching clock (0 = yes, 1 = no).

kI2C_SlaveSelected Indicates whether the slave is selected by an address match.
kI2C_SaveDeselected Indicates that slave was previously deselected (deselect event took place, w1c).

15.5.4.2 enum i2c_slave_address_register_t

Enumerator

kI2C_SlaveAddressRegister0 Slave Address 0 register.
kI2C_SlaveAddressRegister1 Slave Address 1 register.
kI2C_SlaveAddressRegister2 Slave Address 2 register.
kI2C_SlaveAddressRegister3 Slave Address 3 register.

15.5.4.3 enum i2c_slave_address_qual_mode_t

Enumerator

kI2C_QualModeMask The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.
kI2C_QualModeExtend The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

15.5.4.4 enum i2c_slave_bus_speed_t

15.5.4.5 enum i2c_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 I2C_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

kI2C_SlaveAddressMatchEvent Received the slave address after a start or repeated start.
kI2C_SlaveTransmitEvent Callback is requested to provide data to transmit (slave-transmitter role).

kI2C_SlaveReceiveEvent Callback is requested to provide a buffer in which to place received data (slave-receiver role).
kI2C_SlaveCompletionEvent All data in the active transfer have been consumed.
kI2C_SlaveDeselectedEvent The slave function has become deselected (SLVSEL flag changing from 1 to 0.
kI2C_SlaveAllEvents Bit mask of all available events.

15.5.5 Function Documentation

15.5.5.1 void I2C_SlaveGetDefaultConfig (i2c_slave_config_t * *slaveConfig*)

This function provides the following default configuration for the I2C slave peripheral:

```
*   slaveConfig->enableSlave = true;
*   slaveConfig->address0.disable = false;
*   slaveConfig->address0.address = 0u;
*   slaveConfig->address1.disable = true;
*   slaveConfig->address2.disable = true;
*   slaveConfig->address3.disable = true;
*   slaveConfig->busSpeed = kI2C_SlaveStandardMode;
*
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with I2C_SlaveInit(). Be sure to override at least the *address0.address* 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 i2c_slave_config_t.

15.5.5.2 status_t I2C_SlaveInit (I2C_Type * *base*, const i2c_slave_config_t * *slaveConfig*, uint32_t *srcClock_Hz*)

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

Parameters

base	The I2C peripheral base address.
slaveConfig	User provided peripheral configuration. Use I2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
srcClock_Hz	Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock.

15.5.5.3 void I2C_SlaveSetAddress (I2C_Type * *base*, i2c_slave_address_register_t *addressRegister*, uint8_t *address*, bool *addressDisable*)

This function writes new value to Slave Address register.

Parameters

base	The I2C peripheral base address.
address-Register	The module supports multiple address registers. The parameter determines which one shall be changed.
address	The slave address to be stored to the address register for matching.
addressDisable	Disable matching of the specified address register.

15.5.5.4 void I2C_SlaveDeinit (I2C_Type * *base*)

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

15.5.5.5 static void I2C_SlaveEnable (I2C_Type * *base,* bool *enable*) `[inline]`, `[static]`

Parameters

base	The I2C peripheral base address.
enable	True to enable or flase to disable.

15.5.5.6 static void I2C_SlaveClearStatusFlags (I2C_Type * *base,* uint32_t *statusMask*) `[inline]`, `[static]`

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i2c_-slave_flags enumerators OR'd together. You may pass the result of a previous call to I2C_SlaveGetStatusFlags().

See Also

_i2c_slave_flags.

15.5.5.7 status_t I2C_SlaveWriteBlocking (I2C_Type ∗ *base*, const uint8_t ∗ *txBuff*, size_t *txSize*)

The function executes blocking address phase and blocking data phase.

Parameters

base	The I2C peripheral base address.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.

Returns

kStatus_Success Data has been sent.
kStatus_Fail Unexpected slave state (master data write while master read from slave is expected).

15.5.5.8 status_t I2C_SlaveReadBlocking (I2C_Type ∗ *base*, uint8_t ∗ *rxBuff*, size_t *rxSize*)

The function executes blocking address phase and blocking data phase.

Parameters

base	The I2C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.

Returns

kStatus_Success Data has been received.
kStatus_Fail Unexpected slave state (master data read while master write to slave is expected).

15.5.5.9 void I2C_SlaveTransferCreateHandle (I2C_Type * *base*, i2c_slave_handle_t * *handle*, i2c_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 I2C_SlaveTransferAbort() API shall be called.

Parameters

	base	The I2C peripheral base address.
out	*handle*	Pointer to the I2C slave driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

15.5.5.10 status_t I2C_SlaveTransferNonBlocking (I2C_Type * *base*, i2c_slave_handle_t * *handle*, uint32_t *eventMask*)

Call this API after calling I2C_SlaveInit() and I2C_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 I2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes kI2C_SlaveTransmitEvent callback. If no slave Rx transfer is busy, a master write to slave request invokes kI2C_SlaveReceiveEvent callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_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 kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.
eventMask	Bit mask formed by OR'ing together i2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kI2C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

15.5.5.11 status_t I2C_SlaveSetSendBuffer (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*, const void * *txData*, size_t *txSize*, uint32_t *eventMask*)

The function can be called in response to kI2C_SlaveTransmitEvent callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The k-I2C_SlaveTransmitEvent and kI2C_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 kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
transfer	Pointer to i2c_slave_transfer_t structure.
txData	Pointer to data to send to master.
txSize	Size of txData in bytes.
eventMask	Bit mask formed by OR'ing together i2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kI2C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

15.5.5.12 status_t I2C_SlaveSetReceiveBuffer (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*, void * *rxData*, size_t *rxSize*, uint32_t *eventMask*)

The function can be called in response to kI2C_SlaveReceiveEvent callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_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 kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
transfer	Pointer to i2c_slave_transfer_t structure.
rxData	Pointer to data to store data from master.
rxSize	Size of rxData in bytes.
eventMask	Bit mask formed by OR'ing together i2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kI2C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

15.5.5.13 static uint32_t I2C_SlaveGetReceivedAddress (I2C_Type * *base,* volatile i2c_slave_transfer_t * *transfer*) [inline], [static]

This function should only be called from the address match event callback kI2C_SlaveAddressMatchEvent.

Parameters

base	The I2C peripheral base address.
transfer	The I2C slave transfer.

Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

15.5.5.14 void I2C_SlaveTransferAbort (I2C_Type * *base,* i2c_slave_handle_t * *handle*)

Note

This API could be called at any time to stop slave for handling the bus events.

Parameters

base	The I2C peripheral base address.
handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.

Return values

kStatus_Success	
kStatus_I2C_Idle	

15.5.5.15 status_t I2C_SlaveTransferGetCount (I2C_Type * *base*, i2c_slave_handle_t * *handle*, size_t * *count*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_slave_handle_t structure.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

15.5.5.16 void I2C_SlaveTransferHandleIRQ (I2C_Type * *base*, i2c_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 I2C peripheral base address.
handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.

15.6 I2C DMA Driver

15.6.1 Overview

Data Structures

- struct i2c_master_dma_handle_t
  I2C master dma transfer structure. More...

Macros

- #define I2C_MAX_DMA_TRANSFER_COUNT 1024
  Maximum lenght of single DMA transfer (determined by capability of the DMA engine)

Typedefs

- typedef void(∗ i2c_master_dma_transfer_callback_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, status_t status, void ∗userData)
  I2C master dma transfer callback typedef.

Driver version

- #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  I2C DMA driver version 2.0.5.

I2C Block DMA Transfer Operation

- void I2C_MasterTransferCreateHandleDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, i2c_master_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗dmaHandle)
  Init the I2C handle which is used in transactional functions.
- status_t I2C_MasterTransferDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, i2c_master_transfer_t ∗xfer)
  Performs a master dma non-blocking transfer on the I2C bus.
- status_t I2C_MasterTransferGetCountDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, size_t ∗count)
  Get master transfer status during a dma non-blocking transfer.
- void I2C_MasterTransferAbortDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle)
  Abort a master dma non-blocking transfer in a early time.

15.6.2 Data Structure Documentation

15.6.2.1 struct _i2c_master_dma_handle

I2C master dma handle typedef.

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint32_t transferCount
  Indicates progress of the transfer.
- uint32_t remainingBytesDMA
  Remaining byte count to be transferred using DMA.
- uint8_t * buf
  Buffer pointer for current state.
- dma_handle_t * dmaHandle
  The DMA handler used.
- i2c_master_transfer_t transfer
  Copy of the current transfer info.
- i2c_master_dma_transfer_callback_t completionCallback
  Callback function called after dma transfer finished.
- void * userData
  Callback parameter passed to callback function.

15.6.2.1.0.18 Field Documentation

15.6.2.1.0.18.1 uint8_t i2c_master_dma_handle_t::state

15.6.2.1.0.18.2 uint32_t i2c_master_dma_handle_t::remainingBytesDMA

15.6.2.1.0.18.3 uint8_t* i2c_master_dma_handle_t::buf

15.6.2.1.0.18.4 dma_handle_t* i2c_master_dma_handle_t::dmaHandle

15.6.2.1.0.18.5 i2c_master_transfer_t i2c_master_dma_handle_t::transfer

15.6.2.1.0.18.6 i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completion-Callback

15.6.2.1.0.18.7 void* i2c_master_dma_handle_t::userData

15.6.3 Macro Definition Documentation

15.6.3.1 #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

15.6.4 Typedef Documentation

15.6.4.1 typedef void(* i2c_master_dma_transfer_callback_t)(I2C_Type *base, i2c_master_dma_handle_t *handle, status_t status, void *userData)

15.6.5 Function Documentation

15.6.5.1 void I2C_MasterTransferCreateHandleDMA (I2C_Type * *base,* i2c_master_dma_handle_t * *handle,* i2c_master_dma_transfer_callback_t *callback,* void * *userData,* dma_handle_t * *dmaHandle*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure
callback	pointer to user callback function
userData	user param passed to the callback function
dmaHandle	DMA handle pointer

15.6.5.2 status_t I2C_MasterTransferDMA (I2C_Type * *base,* i2c_master_dma_handle_t * *handle,* i2c_master_transfer_t * *xfer*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure
xfer	pointer to transfer structure of i2c_master_transfer_t

Return values

kStatus_Success	Sucessully complete the data transmission.
kStatus_I2C_Busy	Previous transmission still not finished.
kStatus_I2C_Timeout	Transfer error, wait signal timeout.
kStatus_I2C_ArbitrationLost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive Nak during transfer.

15.6.5.3 status_t I2C_MasterTransferGetCountDMA (I2C_Type * *base,* i2c_master_dma_handle_t * *handle,* size_t * *count*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure

count	Number of bytes transferred so far by the non-blocking transaction.

15.6.5.4 void I2C_MasterTransferAbortDMA (I2C_Type * *base*, i2c_master_dma_handle_t * *handle*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure

15.7 I2C FreeRTOS Driver

15.7.1 Overview

Data Structures

- struct i2c_rtos_handle_t
  I2C FreeRTOS handle. More...

Driver version

- #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  I2C freertos driver version 2.0.5.

I2C RTOS Operation

- status_t I2C_RTOS_Init (i2c_rtos_handle_t ∗handle, I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes I2C.
- status_t I2C_RTOS_Deinit (i2c_rtos_handle_t ∗handle)
  Deinitializes the I2C.
- status_t I2C_RTOS_Transfer (i2c_rtos_handle_t ∗handle, i2c_master_transfer_t ∗transfer)
  Performs I2C transfer.

15.7.2 Data Structure Documentation

15.7.2.1 struct i2c_rtos_handle_t

Data Fields

- I2C_Type ∗ base
  I2C base address.
- i2c_master_handle_t drv_handle
  A handle of the underlying driver, treated as opaque by the RTOS layer.
- status_t async_status
  Transactional state of the underlying driver.
- SemaphoreHandle_t mutex
  A mutex to lock the handle during a transfer.
- SemaphoreHandle_t semaphore
  A semaphore to notify and unblock task when the transfer ends.

15.7.3 Macro Definition Documentation

15.7.3.1 #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

15.7.4 Function Documentation

15.7.4.1 status_t I2C_RTOS_Init (i2c_rtos_handle_t * *handle,* I2C_Type * *base,* const i2c_master_config_t * *masterConfig,* uint32_t *srcClock_Hz*)

This function initializes the I2C module and the related RTOS context.

Parameters

handle	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the I2C instance to initialize.
masterConfig	Configuration structure to set-up I2C in master mode.
srcClock_Hz	Frequency of input clock of the I2C module.

Returns

status of the operation.

15.7.4.2 status_t I2C_RTOS_Deinit (i2c_rtos_handle_t * *handle*)

This function deinitializes the I2C module and the related RTOS context.

Parameters

handle	The RTOS I2C handle.

15.7.4.3 status_t I2C_RTOS_Transfer (i2c_rtos_handle_t * *handle*, i2c_master_transfer_t * *transfer*)

This function performs an I2C transfer according to data given in the transfer structure.

Parameters

handle	The RTOS I2C handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

Chapter 16
SPI: Serial Peripheral Interface Driver

16.1 Overview

SPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPI 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 spi_handle_t as the first parameter. Initialize the handle by calling the SPI_MasterTransferCreateHandle() or SPI_SlaveTransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SPI_MasterTransferNonBlocking() and SPI_SlaveTransferNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_SPI_Idle status.

16.2 Typical use case

16.2.1 SPI master transfer using an interrupt method

```
#define BUFFER_LEN (64)
spi_master_handle_t spiHandle;
spi_master_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished = false;

const uint8_t sendData[BUFFER_LEN] = [......];
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_master_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    SPI_MasterGetDefaultConfig(&masterConfig);

    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);
    SPI_MasterTransferCreateHandle(SPI0, &spiHandle, SPI_UserCallback, NULL);

    // Prepare to send.
    xfer.txData = sendData;
    xfer.rxData = receiveBuff;
```

```
    xfer.dataSize = sizeof(sendData);

    // Send out.
    SPI_MasterTransferNonBlocking(SPI0, &spiHandle, &xfer);

    // Wait send finished.
    while (!isFinished)
    {
    }

    // ...
}
```

16.2.2 SPI Send/receive using a DMA method

```
#define BUFFER_LEN (64)
spi_dma_handle_t spiHandle;
dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
spi_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished;

uint8_t sendData[BUFFER_LEN] = ...;
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    // Initialize DMA peripheral
    DMA_Init(DMA0);

    // Initialize SPI peripheral
    SPI_MasterGetDefaultConfig(&masterConfig);
    masterConfig.sselNum = SPI_SSEL;
    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);

    // Enable DMA channels connected to SPI0 Tx/SPI0 Rx request lines
    DMA_EnableChannel(SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_EnableChannel(SPI0, SPI_MASTER_RX_CHANNEL);

    // Set DMA channels priority
    DMA_SetChannelPriority(SPI0, SPI_MASTER_TX_CHANNEL,
      kDMA_ChannelPriority3);
    DMA_SetChannelPriority(SPI0, SPI_MASTER_RX_CHANNEL,
      kDMA_ChannelPriority2);

    // Creates the DMA handle.
    DMA_CreateHandle(&masterTxHandle, SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_CreateHandle(&masterRxHandle, SPI0, SPI_MASTER_RX_CHANNEL);

    // Create SPI DMA handle
    SPI_MasterTransferCreateHandleDMA(SPI0, spiHandle, SPI_UserCallback,
      NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);

    // Prepares to send.
    xfer.txData = sendData;
    xfer.rxData = receiveBuff;
    xfer.dataSize = sizeof(sendData);
```

```
    // Sends out.
    SPI_MasterTransferDMA(SPI0, &spiHandle, &xfer);

    // Waits for send to complete.
    while (!isFinished)
    {
    }

    // ...
}
```

Modules

- SPI DMA Driver
- SPI Driver
- SPI FreeRTOS driver

16.3 SPI Driver

16.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

Files

- file fsl_spi.h

Data Structures

- struct spi_delay_config_t
  SPI delay time configure structure. More...
- struct spi_master_config_t
  SPI master user configure structure. More...
- struct spi_slave_config_t
  SPI slave user configure structure. More...
- struct spi_transfer_t
  SPI transfer structure. More...
- struct spi_half_duplex_transfer_t
  SPI half-duplex(master only) transfer structure. More...
- struct spi_config_t
  Internal configuration structure used in 'spi' and 'spi_dma' driver. More...
- struct spi_master_handle_t
  SPI transfer handle structure. More...

Macros

- #define SPI_DUMMYDATA (0xFFU)
  SPI dummy transfer data, the data is sent while txBuff is NULL.

Typedefs

- typedef spi_master_handle_t spi_slave_handle_t
  Slave handle type.
- typedef void(∗ spi_master_callback_t)(SPI_Type ∗base, spi_master_handle_t ∗handle, status_t status, void ∗userData)
  SPI master callback for finished transmit.
- typedef void(∗ spi_slave_callback_t)(SPI_Type ∗base, spi_slave_handle_t ∗handle, status_t status, void ∗userData)
  SPI slave callback for finished transmit.

Enumerations

- enum spi_xfer_option_t {
  kSPI_FrameDelay = (SPI_FIFOWR_EOF_MASK),
  kSPI_FrameAssert = (SPI_FIFOWR_EOT_MASK) }
  SPI transfer option.
- enum spi_shift_direction_t {
  kSPI_MsbFirst = 0U,
  kSPI_LsbFirst = 1U }
  SPI data shifter direction options.
- enum spi_clock_polarity_t {
  kSPI_ClockPolarityActiveHigh = 0x0U,
  kSPI_ClockPolarityActiveLow }
  SPI clock polarity configuration.
- enum spi_clock_phase_t {
  kSPI_ClockPhaseFirstEdge = 0x0U,
  kSPI_ClockPhaseSecondEdge }
  SPI clock phase configuration.
- enum spi_txfifo_watermark_t {
  kSPI_TxFifo0 = 0,
  kSPI_TxFifo1 = 1,
  kSPI_TxFifo2 = 2,
  kSPI_TxFifo3 = 3,
  kSPI_TxFifo4 = 4,
  kSPI_TxFifo5 = 5,
  kSPI_TxFifo6 = 6,
  kSPI_TxFifo7 = 7 }
  txFIFO watermark values
- enum spi_rxfifo_watermark_t {
  kSPI_RxFifo1 = 0,
  kSPI_RxFifo2 = 1,
  kSPI_RxFifo3 = 2,
  kSPI_RxFifo4 = 3,
  kSPI_RxFifo5 = 4,
  kSPI_RxFifo6 = 5,
  kSPI_RxFifo7 = 6,
  kSPI_RxFifo8 = 7 }
  rxFIFO watermark values
- enum spi_data_width_t {

kSPI_Data4Bits = 3,
kSPI_Data5Bits = 4,
kSPI_Data6Bits = 5,
kSPI_Data7Bits = 6,
kSPI_Data8Bits = 7,
kSPI_Data9Bits = 8,
kSPI_Data10Bits = 9,
kSPI_Data11Bits = 10,
kSPI_Data12Bits = 11,
kSPI_Data13Bits = 12,
kSPI_Data14Bits = 13,
kSPI_Data15Bits = 14,
kSPI_Data16Bits = 15 }
Transfer data width.
- enum spi_ssel_t {
kSPI_Ssel0 = 0,
kSPI_Ssel1 = 1,
kSPI_Ssel2 = 2,
kSPI_Ssel3 = 3 }
Slave select.
- enum spi_spol_t
ssel polarity
- enum _spi_status {
kStatus_SPI_Busy = MAKE_STATUS(kStatusGroup_LPC_SPI, 0),
kStatus_SPI_Idle = MAKE_STATUS(kStatusGroup_LPC_SPI, 1),
kStatus_SPI_Error = MAKE_STATUS(kStatusGroup_LPC_SPI, 2),
kStatus_SPI_BaudrateNotSupport }
SPI transfer status.
- enum _spi_interrupt_enable {
kSPI_RxLvlIrq = SPI_FIFOINTENSET_RXLVL_MASK,
kSPI_TxLvlIrq = SPI_FIFOINTENSET_TXLVL_MASK }
SPI interrupt sources.
- enum _spi_statusflags {
kSPI_TxEmptyFlag = SPI_FIFOSTAT_TXEMPTY_MASK,
kSPI_TxNotFullFlag = SPI_FIFOSTAT_TXNOTFULL_MASK,
kSPI_RxNotEmptyFlag = SPI_FIFOSTAT_RXNOTEMPTY_MASK,
kSPI_RxFullFlag = SPI_FIFOSTAT_RXFULL_MASK }
SPI status flags.

Functions

- uint32_t SPI_GetInstance (SPI_Type ∗base)
Returns instance number for SPI peripheral base address.

Variables

- volatile uint8_t s_dummyData [ ]
  Global variable for dummy data value setting.

Driver version

- #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  SPI driver version 2.0.4.

Initialization and deinitialization

- void SPI_MasterGetDefaultConfig (spi_master_config_t *config)
  Sets the SPI master configuration structure to default values.
- status_t SPI_MasterInit (SPI_Type *base, const spi_master_config_t *config, uint32_t srcClock_-Hz)
  Initializes the SPI with master configuration.
- void SPI_SlaveGetDefaultConfig (spi_slave_config_t *config)
  Sets the SPI slave configuration structure to default values.
- status_t SPI_SlaveInit (SPI_Type *base, const spi_slave_config_t *config)
  Initializes the SPI with slave configuration.
- void SPI_Deinit (SPI_Type *base)
  De-initializes the SPI.
- static void SPI_Enable (SPI_Type *base, bool enable)
  Enable or disable the SPI Master or Slave.

Status

- static uint32_t SPI_GetStatusFlags (SPI_Type *base)
  Gets the status flag.

Interrupts

- static void SPI_EnableInterrupts (SPI_Type *base, uint32_t irqs)
  Enables the interrupt for the SPI.
- static void SPI_DisableInterrupts (SPI_Type *base, uint32_t irqs)
  Disables the interrupt for the SPI.

DMA Control

- void SPI_EnableTxDMA (SPI_Type *base, bool enable)
  Enables the DMA request from SPI txFIFO.
- void SPI_EnableRxDMA (SPI_Type *base, bool enable)
  Enables the DMA request from SPI rxFIFO.

Bus Operations

- void ∗ SPI_GetConfig (SPI_Type ∗base)
  Returns the configurations.
- status_t SPI_MasterSetBaud (SPI_Type ∗base, uint32_t baudrate_Bps, uint32_t srcClock_Hz)
  Sets the baud rate for SPI transfer.
- void SPI_WriteData (SPI_Type ∗base, uint16_t data, uint32_t configFlags)
  Writes a data into the SPI data register.
- static uint32_t SPI_ReadData (SPI_Type ∗base)
  Gets a data from the SPI data register.
- static void SPI_SetTransferDelay (SPI_Type ∗base, const spi_delay_config_t ∗config)
  Set delay time for transfer.
- void SPI_SetDummyData (SPI_Type ∗base, uint8_t dummyData)
  Set up the dummy data.

Transactional

- status_t SPI_MasterTransferCreateHandle (SPI_Type ∗base, spi_master_handle_t ∗handle, spi_master_callback_t callback, void ∗userData)
  Initializes the SPI master handle.
- status_t SPI_MasterTransferBlocking (SPI_Type ∗base, spi_transfer_t ∗xfer)
  Transfers a block of data using a polling method.
- status_t SPI_MasterTransferNonBlocking (SPI_Type ∗base, spi_master_handle_t ∗handle, spi_transfer_t ∗xfer)
  Performs a non-blocking SPI interrupt transfer.
- status_t SPI_MasterHalfDuplexTransferBlocking (SPI_Type ∗base, spi_half_duplex_transfer_t ∗xfer)
  Transfers a block of data using a polling method.
- status_t SPI_MasterHalfDuplexTransferNonBlocking (SPI_Type ∗base, spi_master_handle_t ∗handle, spi_half_duplex_transfer_t ∗xfer)
  Performs a non-blocking SPI interrupt transfer.
- status_t SPI_MasterTransferGetCount (SPI_Type ∗base, spi_master_handle_t ∗handle, size_t ∗count)
  Gets the master transfer count.
- void SPI_MasterTransferAbort (SPI_Type ∗base, spi_master_handle_t ∗handle)
  SPI master aborts a transfer using an interrupt.
- void SPI_MasterTransferHandleIRQ (SPI_Type ∗base, spi_master_handle_t ∗handle)
  Interrupts the handler for the SPI.
- static status_t SPI_SlaveTransferCreateHandle (SPI_Type ∗base, spi_slave_handle_t ∗handle, spi_slave_callback_t callback, void ∗userData)
  Initializes the SPI slave handle.
- static status_t SPI_SlaveTransferNonBlocking (SPI_Type ∗base, spi_slave_handle_t ∗handle, spi_transfer_t ∗xfer)
  Performs a non-blocking SPI slave interrupt transfer.
- static status_t SPI_SlaveTransferGetCount (SPI_Type ∗base, spi_slave_handle_t ∗handle, size_t ∗count)
  Gets the slave transfer count.
- static void SPI_SlaveTransferAbort (SPI_Type ∗base, spi_slave_handle_t ∗handle)
  SPI slave aborts a transfer using an interrupt.

- static void SPI_SlaveTransferHandleIRQ (SPI_Type ∗base, spi_slave_handle_t ∗handle)
  Interrupts a handler for the SPI slave.

16.3.2 Data Structure Documentation

16.3.2.1 struct spi_delay_config_t

Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maxinun value of these delay time is 15.

Data Fields

- uint8_t preDelay
  Delay between SSEL assertion and the beginning of transfer.
- uint8_t postDelay
  Delay between the end of transfer and SSEL deassertion.
- uint8_t frameDelay
  Delay between frame to frame.
- uint8_t transferDelay
  Delay between transfer to transfer.

16.3.2.1.0.19 Field Documentation

16.3.2.1.0.19.1 uint8_t spi_delay_config_t::preDelay

16.3.2.1.0.19.2 uint8_t spi_delay_config_t::postDelay

16.3.2.1.0.19.3 uint8_t spi_delay_config_t::frameDelay

16.3.2.1.0.19.4 uint8_t spi_delay_config_t::transferDelay

16.3.2.2 struct spi_master_config_t

Data Fields

- bool enableLoopback
  Enable loopback for test purpose.
- bool enableMaster
  Enable SPI at initialization time.
- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- uint32_t baudRate_Bps
  Baud Rate for SPI in Hz.
- spi_data_width_t dataWidth

Width of the data.
- spi_ssel_t sselNum
  Slave select number.
- spi_spol_t sselPol
  Configure active CS polarity.
- spi_txfifo_watermark_t txWatermark
  txFIFO watermark
- spi_rxfifo_watermark_t rxWatermark
  rxFIFO watermark
- spi_delay_config_t delayConfig
  Delay configuration.

16.3.2.2.0.20 Field Documentation

16.3.2.2.0.20.1 spi_delay_config_t spi_master_config_t::delayConfig

16.3.2.3 struct spi_slave_config_t

Data Fields

- bool enableSlave
  Enable SPI at initialization time.
- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- spi_data_width_t dataWidth
  Width of the data.
- spi_spol_t sselPol
  Configure active CS polarity.
- spi_txfifo_watermark_t txWatermark
  txFIFO watermark
- spi_rxfifo_watermark_t rxWatermark
  rxFIFO watermark

16.3.2.4 struct spi_transfer_t

Data Fields

- uint8_t ∗ txData
  Send buffer.
- uint8_t ∗ rxData
  Receive buffer.
- uint32_t configFlags
  Additional option to control transfer, spi_xfer_option_t.
- size_t dataSize
  Transfer bytes.

16.3.2.4.0.21 Field Documentation

16.3.2.4.0.21.1 uint32_t spi_transfer_t::configFlags

16.3.2.5 struct spi_half_duplex_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- size_t txDataSize
  Transfer bytes for transmit.
- size_t rxDataSize
  Transfer bytes.
- uint32_t configFlags
  Transfer configuration flags, spi_xfer_option_t.
- bool isPcsAssertInTransfer
  If PCS pin keep assert between transmit and receive.
- bool isTransmitFirst
  True for transmit first and false for receive first.

16.3.2.5.0.22 Field Documentation

16.3.2.5.0.22.1 uint32_t spi_half_duplex_transfer_t::configFlags

16.3.2.5.0.22.2 bool spi_half_duplex_transfer_t::isPcsAssertInTransfer

true for assert and false for deassert.

16.3.2.5.0.22.3 bool spi_half_duplex_transfer_t::isTransmitFirst

16.3.2.6 struct spi_config_t

16.3.2.7 struct _spi_master_handle

Master handle type.

Data Fields

- uint8_t *volatile txData
  Transfer buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t txRemainingBytes
  Number of data to be transmitted [in bytes].
- volatile size_t rxRemainingBytes
  Number of data to be received [in bytes].
- volatile size_t toReceiveCount

Receive data remaining in bytes.
- size_t totalByteCount
  A number of transfer bytes.
- volatile uint32_t state
  SPI internal state.
- spi_master_callback_t callback
  SPI callback.
- void * userData
  Callback parameter.
- uint8_t dataWidth
  Width of the data [Valid values: 1 to 16].
- uint8_t sselNum
  Slave select number to be asserted when transferring data [Valid values: 0 to 3].
- uint32_t configFlags
  Additional option to control transfer.
- spi_txfifo_watermark_t txWatermark
  txFIFO watermark
- spi_rxfifo_watermark_t rxWatermark
  rxFIFO watermark

16.3.3 Macro Definition Documentation

16.3.3.1 #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

16.3.3.2 #define SPI_DUMMYDATA (0xFFU)

16.3.4 Enumeration Type Documentation

16.3.4.1 enum spi_xfer_option_t

Enumerator

kSPI_FrameDelay A delay may be inserted, defined in the DLY register.
kSPI_FrameAssert SSEL will be deasserted at the end of a transfer.

16.3.4.2 enum spi_shift_direction_t

Enumerator

kSPI_MsbFirst Data transfers start with most significant bit.
kSPI_LsbFirst Data transfers start with least significant bit.

16.3.4.3 enum spi_clock_polarity_t

Enumerator

kSPI_ClockPolarityActiveHigh Active-high SPI clock (idles low).
kSPI_ClockPolarityActiveLow Active-low SPI clock (idles high).

16.3.4.4 enum spi_clock_phase_t

Enumerator

kSPI_ClockPhaseFirstEdge First edge on SCK occurs at the middle of the first cycle of a data transfer.
kSPI_ClockPhaseSecondEdge First edge on SCK occurs at the start of the first cycle of a data transfer.

16.3.4.5 enum spi_txfifo_watermark_t

Enumerator

kSPI_TxFifo0 SPI tx watermark is empty.
kSPI_TxFifo1 SPI tx watermark at 1 item.
kSPI_TxFifo2 SPI tx watermark at 2 items.
kSPI_TxFifo3 SPI tx watermark at 3 items.
kSPI_TxFifo4 SPI tx watermark at 4 items.
kSPI_TxFifo5 SPI tx watermark at 5 items.
kSPI_TxFifo6 SPI tx watermark at 6 items.
kSPI_TxFifo7 SPI tx watermark at 7 items.

16.3.4.6 enum spi_rxfifo_watermark_t

Enumerator

kSPI_RxFifo1 SPI rx watermark at 1 item.
kSPI_RxFifo2 SPI rx watermark at 2 items.
kSPI_RxFifo3 SPI rx watermark at 3 items.
kSPI_RxFifo4 SPI rx watermark at 4 items.
kSPI_RxFifo5 SPI rx watermark at 5 items.
kSPI_RxFifo6 SPI rx watermark at 6 items.
kSPI_RxFifo7 SPI rx watermark at 7 items.
kSPI_RxFifo8 SPI rx watermark at 8 items.

16.3.4.7 enum spi_data_width_t

Enumerator

kSPI_Data4Bits 4 bits data width
kSPI_Data5Bits 5 bits data width
kSPI_Data6Bits 6 bits data width
kSPI_Data7Bits 7 bits data width
kSPI_Data8Bits 8 bits data width
kSPI_Data9Bits 9 bits data width
kSPI_Data10Bits 10 bits data width
kSPI_Data11Bits 11 bits data width
kSPI_Data12Bits 12 bits data width
kSPI_Data13Bits 13 bits data width
kSPI_Data14Bits 14 bits data width
kSPI_Data15Bits 15 bits data width
kSPI_Data16Bits 16 bits data width

16.3.4.8 enum spi_ssel_t

Enumerator

kSPI_Ssel0 Slave select 0.
kSPI_Ssel1 Slave select 1.
kSPI_Ssel2 Slave select 2.
kSPI_Ssel3 Slave select 3.

16.3.4.9 enum _spi_status

Enumerator

kStatus_SPI_Busy SPI bus is busy.
kStatus_SPI_Idle SPI is idle.
kStatus_SPI_Error SPI error.
kStatus_SPI_BaudrateNotSupport Baudrate is not support in current clock source.

16.3.4.10 enum _spi_interrupt_enable

Enumerator

kSPI_RxLvlIrq Rx level interrupt.
kSPI_TxLvlIrq Tx level interrupt.

16.3.4.11 enum _spi_statusflags

Enumerator

kSPI_TxEmptyFlag txFifo is empty
kSPI_TxNotFullFlag txFifo is not full
kSPI_RxNotEmptyFlag rxFIFO is not empty
kSPI_RxFullFlag rxFIFO is full

16.3.5 Function Documentation

16.3.5.1 uint32_t SPI_GetInstance (SPI_Type * *base*)

16.3.5.2 void SPI_MasterGetDefaultConfig (spi_master_config_t * *config*)

The purpose of this API is to get the configuration structure initialized for use in SPI_MasterInit(). User may use the initialized structure unchanged in SPI_MasterInit(), or modify some fields of the structure before calling SPI_MasterInit(). After calling this API, the master is ready to transfer. Example:

```
spi_master_config_t config;
SPI_MasterGetDefaultConfig(&config);
```

Parameters

config	pointer to master config structure

16.3.5.3 status_t SPI_MasterInit (SPI_Type * *base,* const spi_master_config_t * *config,* uint32_t *srcClock_Hz*)

The configuration structure can be filled by user from scratch, or be set with default values by SPI_MasterGetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
spi_master_config_t config = {
.baudRate_Bps = 400000,
...
};
SPI_MasterInit(SPI0, &config);
```

Parameters

base	SPI base pointer
config	pointer to master configuration structure
srcClock_Hz	Source clock frequency.

16.3.5.4 void SPI_SlaveGetDefaultConfig (spi_slave_config_t * *config*)

The purpose of this API is to get the configuration structure initialized for use in SPI_SlaveInit(). Modify some fields of the structure before calling SPI_SlaveInit(). Example:

```
spi_slave_config_t config;
SPI_SlaveGetDefaultConfig(&config);
```

Parameters

config	pointer to slave configuration structure

16.3.5.5 status_t SPI_SlaveInit (SPI_Type * *base*, const spi_slave_config_t * *config*)

The configuration structure can be filled by user from scratch or be set with default values by SPI_Slave-GetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
spi_slave_config_t config = {
.polarity = flexSPIClockPolarity_ActiveHigh;
.phase = flexSPIClockPhase_FirstEdge;
.direction = flexSPIMsbFirst;
...
};
SPI_SlaveInit(SPI0, &config);
```

Parameters

base	SPI base pointer
config	pointer to slave configuration structure

16.3.5.6 void SPI_Deinit (SPI_Type * *base*)

Calling this API resets the SPI module, gates the SPI clock. The SPI module can't work unless calling the SPI_MasterInit/SPI_SlaveInit to initialize module.

Parameters

base	SPI base pointer

16.3.5.7 static void SPI_Enable (SPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	SPI base pointer
enable	or disable (true = enable, false = disable)

16.3.5.8 static uint32_t SPI_GetStatusFlags (SPI_Type * *base*) [inline], [static]

Parameters

base	SPI base pointer

Returns

SPI Status, use status flag to AND _spi_statusflags could get the related status.

16.3.5.9 static void SPI_EnableInterrupts (SPI_Type * *base*, uint32_t *irqs*) [inline], [static]

Parameters

base	SPI base pointer
irqs	SPI interrupt source. The parameter can be any combination of the following values: • kSPI_RxLvlIrq • kSPI_TxLvlIrq

16.3.5.10 static void SPI_DisableInterrupts (SPI_Type * *base*, uint32_t *irqs*) [inline], [static]

Parameters

base	SPI base pointer
irqs	SPI interrupt source. The parameter can be any combination of the following values: • kSPI_RxLvlIrq • kSPI_TxLvlIrq

16.3.5.11 void SPI_EnableTxDMA (SPI_Type * *base*, bool *enable*)

Parameters

base	SPI base pointer
enable	True means enable DMA, false means disable DMA

16.3.5.12 void SPI_EnableRxDMA (SPI_Type * *base*, bool *enable*)

Parameters

base	SPI base pointer
enable	True means enable DMA, false means disable DMA

16.3.5.13 void* SPI_GetConfig (SPI_Type * *base*)

Parameters

base	SPI peripheral address.

Returns

return configurations which contain datawidth and SSEL numbers. return data type is a pointer of spi_config_t.

16.3.5.14 status_t SPI_MasterSetBaud (SPI_Type * *base*, uint32_t *baudrate_Bps*, uint32_t *srcClock_Hz*)

This is only used in master.

Parameters

base	SPI base pointer
baudrate_Bps	baud rate needed in Hz.
srcClock_Hz	SPI source clock frequency in Hz.

16.3.5.15 void SPI_WriteData (SPI_Type ∗ *base*, uint16_t *data*, uint32_t *configFlags*)

Parameters

base	SPI base pointer
data	needs to be write.
configFlags	transfer configuration options spi_xfer_option_t

16.3.5.16 static uint32_t SPI_ReadData (SPI_Type ∗ *base*) [inline], [static]

Parameters

base	SPI base pointer

Returns

Data in the register.

16.3.5.17 static void SPI_SetTransferDelay (SPI_Type ∗ *base*, const spi_delay_config_t ∗ *config*) [inline], [static]

```
the delay uint is SPI clock time, maximum value is 0xF.
```

Parameters

base	SPI base pointer
config	configuration for delay option spi_delay_config_t.

16.3.5.18 void SPI_SetDummyData (SPI_Type ∗ *base*, uint8_t *dummyData*)

Parameters

base	SPI peripheral address.
dummyData	Data to be transferred when tx buffer is NULL.

16.3.5.19 status_t SPI_MasterTransferCreateHandle (SPI_Type * *base*, spi_master_handle_t * *handle*, spi_master_callback_t *callback*, void * *userData*)

This function initializes the SPI master handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	Callback function.
userData	User data.

16.3.5.20 status_t SPI_MasterTransferBlocking (SPI_Type * *base*, spi_transfer_t * *xfer*)

Parameters

base	SPI base pointer
xfer	pointer to spi_xfer_config_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.

16.3.5.21 status_t SPI_MasterTransferNonBlocking (SPI_Type * *base*, spi_master_handle_t * *handle*, spi_transfer_t * *xfer*)

Parameters

base	SPI peripheral base address.
handle	pointer to spi_master_handle_t structure which stores the transfer state
xfer	pointer to spi_xfer_config_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

16.3.5.22 status_t SPI_MasterHalfDuplexTransferBlocking (SPI_Type * *base*, spi_half_duplex_transfer_t * *xfer*)

This function will do a half-duplex transfer for SPI master, This is a blocking function, which does not retuen until all transfer have been completed. And data transfer mechanism is half-duplex, users can set transmit first or receive first.

Parameters

base	SPI base pointer
xfer	pointer to spi_half_duplex_transfer_t structure

Returns

status of status_t.

16.3.5.23 status_t SPI_MasterHalfDuplexTransferNonBlocking (SPI_Type * *base*, spi_master_handle_t * *handle*, spi_half_duplex_transfer_t * *xfer*)

This function using polling way to do the first half transimission and using interrupts to do the second half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

Parameters

base	SPI peripheral base address.
handle	pointer to spi_master_handle_t structure which stores the transfer state
xfer	pointer to spi_half_duplex_transfer_t structure

Returns

status of status_t.

16.3.5.24 status_t SPI_MasterTransferGetCount (SPI_Type * *base*, spi_master_handle_t * *handle*, size_t * *count*)

This function gets the master transfer count.

Parameters

base	SPI peripheral base address.
handle	Pointer to the spi_master_handle_t structure which stores the transfer state.
count	The number of bytes transferred by using the non-blocking transaction.

Returns

status of status_t.

16.3.5.25 void SPI_MasterTransferAbort (SPI_Type * *base*, spi_master_handle_t * *handle*)

This function aborts a transfer using an interrupt.

Parameters

base	SPI peripheral base address.
handle	Pointer to the spi_master_handle_t structure which stores the transfer state.

16.3.5.26 void SPI_MasterTransferHandleIRQ (SPI_Type * *base*, spi_master_handle_t * *handle*)

Parameters

base	SPI peripheral base address.
handle	pointer to spi_master_handle_t structure which stores the transfer state.

16.3.5.27 static status_t SPI_SlaveTransferCreateHandle (SPI_Type * *base*, spi_slave_handle_t * *handle*, spi_slave_callback_t *callback*, void * *userData*) [inline], [static]

This function initializes the SPI slave handle which can be used for other SPI slave transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	Callback function.
userData	User data.

16.3.5.28 static status_t SPI_SlaveTransferNonBlocking (SPI_Type * *base*, spi_slave_handle_t * *handle*, spi_transfer_t * *xfer*) [inline], [static]

Note

The API returns immediately after the transfer initialization is finished.

Parameters

base	SPI peripheral base address.
handle	pointer to spi_master_handle_t structure which stores the transfer state
xfer	pointer to spi_xfer_config_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

16.3.5.29 static status_t SPI_SlaveTransferGetCount (SPI_Type * base, spi_slave_handle_t * handle, size_t * count) [inline], [static]

This function gets the slave transfer count.

Parameters

base	SPI peripheral base address.
handle	Pointer to the spi_master_handle_t structure which stores the transfer state.
count	The number of bytes transferred by using the non-blocking transaction.

Returns

status of status_t.

16.3.5.30 static void SPI_SlaveTransferAbort (SPI_Type * *base*, spi_slave_handle_t * *handle*) [inline], [static]

This function aborts a transfer using an interrupt.

Parameters

base	SPI peripheral base address.
handle	Pointer to the spi_slave_handle_t structure which stores the transfer state.

16.3.5.31 static void SPI_SlaveTransferHandleIRQ (SPI_Type * *base*, spi_slave_handle_t * *handle*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	pointer to spi_slave_handle_t structure which stores the transfer state

16.3.6 Variable Documentation

16.3.6.1 volatile uint8_t s_dummyData[]

16.4 SPI DMA Driver

16.4.1 Overview

This section describes the programming interface of the SPI DMA driver.

Files

- file fsl_spi_dma.h

Data Structures

- struct spi_dma_handle_t
  SPI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ spi_dma_callback_t)(SPI_Type ∗base, spi_dma_handle_t ∗handle, status_t status, void ∗userData)
  SPI DMA callback called at the end of transfer.

Driver version

- #define FSL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  SPI DMA driver version 2.0.4.

DMA Transactional

- status_t SPI_MasterTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI master DMA handle.
- status_t SPI_MasterTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.
- status_t SPI_MasterHalfDuplexTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_half_duplex_transfer_t ∗xfer)
  Transfers a block of data using a DMA method.
- static status_t SPI_SlaveTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI slave DMA handle.
- static status_t SPI_SlaveTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.

- void SPI_MasterTransferAbortDMA (SPI_Type *base, spi_dma_handle_t *handle)
  Abort a SPI transfer using DMA.
- status_t SPI_MasterTransferGetCountDMA (SPI_Type *base, spi_dma_handle_t *handle, size_t *count)
  Gets the master DMA transfer remaining bytes.
- static void SPI_SlaveTransferAbortDMA (SPI_Type *base, spi_dma_handle_t *handle)
  Abort a SPI transfer using DMA.
- static status_t SPI_SlaveTransferGetCountDMA (SPI_Type *base, spi_dma_handle_t *handle, size_t *count)
  Gets the slave DMA transfer remaining bytes.

16.4.2 Data Structure Documentation

16.4.2.1 struct _spi_dma_handle

Data Fields

- volatile bool txInProgress
  Send transfer finished.
- volatile bool rxInProgress
  Receive transfer finished.
- dma_handle_t * txHandle
  DMA handler for SPI send.
- dma_handle_t * rxHandle
  DMA handler for SPI receive.
- uint8_t bytesPerFrame
  Bytes in a frame for SPI transfer.
- spi_dma_callback_t callback
  Callback for SPI DMA transfer.
- void * userData
  User Data for SPI DMA callback.
- uint32_t state
  Internal state of SPI DMA transfer.
- size_t transferSize
  Bytes need to be transfer.

16.4.3 Macro Definition Documentation

16.4.3.1 #define FSL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

16.4.4 Typedef Documentation

16.4.4.1 typedef void(* spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)

16.4.5 Function Documentation

16.4.5.1 status_t SPI_MasterTransferCreateHandleDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *txHandle*, dma_handle_t * *rxHandle*)

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

16.4.5.2 status_t SPI_MasterTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_transfer_t * *xfer*)

Note

> This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

16.4.5.3 status_t SPI_MasterHalfDuplexTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_half_duplex_transfer_t * *xfer*)

This function using polling way to do the first half transimission and using DMA way to do the srcond half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

Parameters

base	SPI base pointer
handle	A pointer to the spi_master_dma_handle_t structure which stores the transfer state.
transfer	A pointer to the spi_half_duplex_transfer_t structure.

Returns

status of status_t.

16.4.5.4 static status_t SPI_SlaveTransferCreateHandleDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *txHandle*, dma_handle_t * *rxHandle*) [inline], [static]

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

16.4.5.5 static status_t SPI_SlaveTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_transfer_t * *xfer*) [inline], [static]

Note

This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

16.4.5.6 void SPI_MasterTransferAbortDMA (SPI_Type ∗ *base,* spi_dma_handle_t ∗ *handle*)

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

16.4.5.7 status_t SPI_MasterTransferGetCountDMA (SPI_Type ∗ *base,* spi_dma_handle_t ∗ *handle,* size_t ∗ *count*)

This function gets the master DMA transfer remaining bytes.

Parameters

base	SPI peripheral base address.
handle	A pointer to the spi_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

16.4.5.8 static void SPI_SlaveTransferAbortDMA (SPI_Type ∗ *base,* spi_dma_handle_t ∗ *handle*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

16.4.5.9 static status_t SPI_SlaveTransferGetCountDMA (SPI_Type ∗ *base*, spi_dma_handle_t ∗ *handle*, size_t ∗ *count*) [inline], [static]

This function gets the slave DMA transfer remaining bytes.

Parameters

base	SPI peripheral base address.
handle	A pointer to the spi_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

16.5 SPI FreeRTOS driver

16.5.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

Files

- file fsl_spi_freertos.h

Data Structures

- struct spi_rtos_handle_t
  SPI FreeRTOS handle. More...

Driver version

- #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  SPI freertos driver version 2.0.4.

SPI RTOS Operation

- status_t SPI_RTOS_Init (spi_rtos_handle_t ∗handle, SPI_Type ∗base, const spi_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes SPI.
- status_t SPI_RTOS_Deinit (spi_rtos_handle_t ∗handle)
  Deinitializes the SPI.
- status_t SPI_RTOS_Transfer (spi_rtos_handle_t ∗handle, spi_transfer_t ∗transfer)
  Performs SPI transfer.

16.5.2 Data Structure Documentation

16.5.2.1 struct spi_rtos_handle_t

Data Fields

- SPI_Type ∗ base
  SPI base address.
- spi_master_handle_t drv_handle
  Handle of the underlying driver, treated as opaque by the RTOS layer.
- SemaphoreHandle_t mutex
  Mutex to lock the handle during a trasfer.
- SemaphoreHandle_t event

Semaphore to notify and unblock task when transfer ends.

16.5.3 Macro Definition Documentation

16.5.3.1 #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

16.5.4 Function Documentation

16.5.4.1 status_t SPI_RTOS_Init (spi_rtos_handle_t * *handle,* SPI_Type * *base,* const spi_master_config_t * *masterConfig,* uint32_t *srcClock_Hz*)

This function initializes the SPI module and related RTOS context.

Parameters

handle	The RTOS SPI handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the SPI instance to initialize.
masterConfig	Configuration structure to set-up SPI in master mode.
srcClock_Hz	Frequency of input clock of the SPI module.

Returns

status of the operation.

16.5.4.2 status_t SPI_RTOS_Deinit (spi_rtos_handle_t * *handle*)

This function deinitializes the SPI module and related RTOS context.

Parameters

handle	The RTOS SPI handle.

16.5.4.3 status_t SPI_RTOS_Transfer (spi_rtos_handle_t * *handle,* spi_transfer_t * *transfer*)

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

handle	The RTOS SPI handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

Chapter 17
USART: Universal Asynchronous Receiver/Transmitter Driver

17.1 Overview

The MCUXpresso SDK provides a peripheral UART driver for the Universal Synchronous Receiver/-Transmitter (USART) module of MCUXpresso SDK devices. Driver does not support synchronous mode.

The USART driver includes two parts: functional APIs and transactional APIs.

Functional APIs are used for USART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the USART peripheral and know how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. USART functional operation groups provide the functional APIs set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the usart_handle_t as the second parameter. Initialize the handle by calling the USART_Transfer-CreateHandle() API.

Transactional APIs support asynchronous transfer, which means that the functions USART_TransferSend-NonBlocking() and USART_TransferReceiveNonBlocking() set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_USART_-TxIdle and kStatus_USART_RxIdle.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the USART_TransferCreateHandle(). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The USART_TransferReceiveNonBlocking() function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the kStatus_USART_RxIdle.

If the receive ring buffer is full, the upper layer is informed through a callback with the kStatus_USAR-T_RxRingBufferOverrun. In the callback function, the upper layer reads data out from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code:

```
USART_TransferCreateHandle(USART0, &handle, USART_UserCallback, NULL);
```

In this example, the buffer size is 32, but only 31 bytes are used for saving data.

17.2 Typical use case

17.2.1 USART Send/receive using a polling method

```
uint8_t ch;
USART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableTx = true;
user_config.enableRx = true;

USART_Init(USART1,&user_config,120000000U);

while(1)
{
    USART_ReadBlocking(USART1, &ch, 1);
    USART_WriteBlocking(USART1, &ch, 1);
}
```

17.2.2 USART Send/receive using an interrupt method

```
usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);

    // Prepare to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData);
    txFinished = false;

    // Send out.
    USART_TransferSendNonBlocking(USART1, &g_usartHandle, &sendXfer);
```

```
    // Wait send finished.
    while (!txFinished)
    {
    }

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
    rxFinished = false;

    // Receive.
    USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer,
      NULL);

    // Wait receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

17.2.3 USART Receive using the ringbuffer feature

```
#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE     32

usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);
    USART_TransferStartRingBuffer(USART1, &g_usartHandle, ringBuffer,
      RING_BUFFER_SIZE);
    // Now the RX is working in background, receive in to ring buffer.

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
```

```
    rxFinished = false;

    // Receive.
    USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer);

    if (bytesRead = RX_DATA_SIZE) /* Have read enough data. */
    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Wait receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}
```

17.2.4 USART Send/Receive using the DMA method

```
usart_handle_t g_usartHandle;
dma_handle_t g_usartTxDmaHandle;
dma_handle_t g_usartRxDmaHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);

    // Set up the DMA
```

```
    DMA_Init(DMA0);
    DMA_EnableChannel(DMA0, USART_TX_DMA_CHANNEL);
    DMA_EnableChannel(DMA0, USART_RX_DMA_CHANNEL);

    DMA_CreateHandle(&g_usartTxDmaHandle, DMA0, USART_TX_DMA_CHANNEL);
    DMA_CreateHandle(&g_usartRxDmaHandle, DMA0, USART_RX_DMA_CHANNEL);

    USART_TransferCreateHandleDMA(USART1, &g_usartHandle, USART_UserCallback,
      NULL, &g_usartTxDmaHandle, &g_usartRxDmaHandle);

    // Prepare to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData);
    txFinished = false;

    // Send out.
    USART_TransferSendDMA(USART1, &g_usartHandle, &sendXfer);

    // Wait send finished.
    while (!txFinished)
    {
    }

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
    rxFinished = false;

    // Receive.
    USART_TransferReceiveDMA(USART1, &g_usartHandle, &receiveXfer);

    // Wait receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

Modules

- USART DMA Driver
- USART Driver
- USART FreeRTOS Driver

17.3 USART Driver

17.3.1 Overview

Data Structures

- struct usart_config_t
  USART configuration structure. More...
- struct usart_transfer_t
  USART transfer structure. More...
- struct usart_handle_t
  USART handle structure. More...

Typedefs

- typedef void(∗ usart_transfer_callback_t)(USART_Type ∗base, usart_handle_t ∗handle, status_t status, void ∗userData)
  USART transfer callback function.

Enumerations

- enum _usart_status {
  kStatus_USART_TxBusy = MAKE_STATUS(kStatusGroup_LPC_USART, 0),
  kStatus_USART_RxBusy = MAKE_STATUS(kStatusGroup_LPC_USART, 1),
  kStatus_USART_TxIdle = MAKE_STATUS(kStatusGroup_LPC_USART, 2),
  kStatus_USART_RxIdle = MAKE_STATUS(kStatusGroup_LPC_USART, 3),
  kStatus_USART_TxError = MAKE_STATUS(kStatusGroup_LPC_USART, 7),
  kStatus_USART_RxError = MAKE_STATUS(kStatusGroup_LPC_USART, 9),
  kStatus_USART_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_LPC_USART, 8),
  kStatus_USART_NoiseError = MAKE_STATUS(kStatusGroup_LPC_USART, 10),
  kStatus_USART_FramingError = MAKE_STATUS(kStatusGroup_LPC_USART, 11),
  kStatus_USART_ParityError = MAKE_STATUS(kStatusGroup_LPC_USART, 12),
  kStatus_USART_BaudrateNotSupport }
  Error codes for the USART driver.
- enum usart_sync_mode_t {
  kUSART_SyncModeDisabled = 0x0U,
  kUSART_SyncModeSlave = 0x2U,
  kUSART_SyncModeMaster = 0x3U }
  USART synchronous mode.
- enum usart_parity_mode_t {
  kUSART_ParityDisabled = 0x0U,
  kUSART_ParityEven = 0x2U,
  kUSART_ParityOdd = 0x3U }
  USART parity mode.

- enum usart_stop_bit_count_t {
  kUSART_OneStopBit = 0U,
  kUSART_TwoStopBit = 1U }
  USART stop bit count.
- enum usart_data_len_t {
  kUSART_7BitsPerChar = 0U,
  kUSART_8BitsPerChar = 1U }
  USART data size.
- enum usart_clock_polarity_t {
  kUSART_RxSampleOnFallingEdge = 0x0U,
  kUSART_RxSampleOnRisingEdge = 0x1U }
  USART clock polarity configuration, used in sync mode.
- enum usart_txfifo_watermark_t {
  kUSART_TxFifo0 = 0,
  kUSART_TxFifo1 = 1,
  kUSART_TxFifo2 = 2,
  kUSART_TxFifo3 = 3,
  kUSART_TxFifo4 = 4,
  kUSART_TxFifo5 = 5,
  kUSART_TxFifo6 = 6,
  kUSART_TxFifo7 = 7 }
  txFIFO watermark values
- enum usart_rxfifo_watermark_t {
  kUSART_RxFifo1 = 0,
  kUSART_RxFifo2 = 1,
  kUSART_RxFifo3 = 2,
  kUSART_RxFifo4 = 3,
  kUSART_RxFifo5 = 4,
  kUSART_RxFifo6 = 5,
  kUSART_RxFifo7 = 6,
  kUSART_RxFifo8 = 7 }
  rxFIFO watermark values
- enum _usart_interrupt_enable
  USART interrupt configuration structure, default settings all disabled.
- enum _usart_flags {
  kUSART_TxError = (USART_FIFOSTAT_TXERR_MASK),
  kUSART_RxError = (USART_FIFOSTAT_RXERR_MASK),
  kUSART_TxFifoEmptyFlag = (USART_FIFOSTAT_TXEMPTY_MASK),
  kUSART_TxFifoNotFullFlag = (USART_FIFOSTAT_TXNOTFULL_MASK),
  kUSART_RxFifoNotEmptyFlag = (USART_FIFOSTAT_RXNOTEMPTY_MASK),
  kUSART_RxFifoFullFlag = (USART_FIFOSTAT_RXFULL_MASK) }
  USART status flags.

Functions

- uint32_t USART_GetInstance (USART_Type ∗base)

Returns instance number for USART peripheral base address.

Driver version

- #define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  USART driver version 2.1.0.

Initialization and deinitialization

- status_t USART_Init (USART_Type ∗base, const usart_config_t ∗config, uint32_t srcClock_Hz)
  Initializes a USART instance with user configuration structure and peripheral clock.
- void USART_Deinit (USART_Type ∗base)
  Deinitializes a USART instance.
- void USART_GetDefaultConfig (usart_config_t ∗config)
  Gets the default configuration structure.
- status_t USART_SetBaudRate (USART_Type ∗base, uint32_t baudrate_Bps, uint32_t srcClock_-Hz)
  Sets the USART instance baud rate.

Status

- static uint32_t USART_GetStatusFlags (USART_Type ∗base)
  Get USART status flags.
- static void USART_ClearStatusFlags (USART_Type ∗base, uint32_t mask)
  Clear USART status flags.

Interrupts

- static void USART_EnableInterrupts (USART_Type ∗base, uint32_t mask)
  Enables USART interrupts according to the provided mask.
- static void USART_DisableInterrupts (USART_Type ∗base, uint32_t mask)
  Disables USART interrupts according to a provided mask.
- static uint32_t USART_GetEnabledInterrupts (USART_Type ∗base)
  Returns enabled USART interrupts.
- static void USART_EnableTxDMA (USART_Type ∗base, bool enable)
  Enable DMA for Tx.
- static void USART_EnableRxDMA (USART_Type ∗base, bool enable)
  Enable DMA for Rx.
- static void USART_EnableCTS (USART_Type ∗base, bool enable)
  Enable CTS.
- static void USART_EnableContinuousSCLK (USART_Type ∗base, bool enable)
  Continuous Clock generation.
- static void USART_EnableAutoClearSCLK (USART_Type ∗base, bool enable)
  Enable Continuous Clock generation bit auto clear.

Bus Operations

- static void USART_WriteByte (USART_Type ∗base, uint8_t data)
  Writes to the FIFOWR register.
- static uint8_t USART_ReadByte (USART_Type ∗base)
  Reads the FIFORD register directly.
- void USART_WriteBlocking (USART_Type ∗base, const uint8_t ∗data, size_t length)
  Writes to the TX register using a blocking method.
- status_t USART_ReadBlocking (USART_Type ∗base, uint8_t ∗data, size_t length)
  Read RX data register using a blocking method.

Transactional

- status_t USART_TransferCreateHandle (USART_Type ∗base, usart_handle_t ∗handle, usart_transfer_callback_t callback, void ∗userData)
  Initializes the USART handle.
- status_t USART_TransferSendNonBlocking (USART_Type ∗base, usart_handle_t ∗handle, usart_transfer_t ∗xfer)
  Transmits a buffer of data using the interrupt method.
- void USART_TransferStartRingBuffer (USART_Type ∗base, usart_handle_t ∗handle, uint8_t ∗ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void USART_TransferStopRingBuffer (USART_Type ∗base, usart_handle_t ∗handle)
  Aborts the background transfer and uninstalls the ring buffer.
- size_t USART_TransferGetRxRingBufferLength (usart_handle_t ∗handle)
  Get the length of received data in RX ring buffer.
- void USART_TransferAbortSend (USART_Type ∗base, usart_handle_t ∗handle)
  Aborts the interrupt-driven data transmit.
- status_t USART_TransferGetSendCount (USART_Type ∗base, usart_handle_t ∗handle, uint32_t ∗count)
  Get the number of bytes that have been written to USART TX register.
- status_t USART_TransferReceiveNonBlocking (USART_Type ∗base, usart_handle_t ∗handle, usart_transfer_t ∗xfer, size_t ∗receivedBytes)
  Receives a buffer of data using an interrupt method.
- void USART_TransferAbortReceive (USART_Type ∗base, usart_handle_t ∗handle)
  Aborts the interrupt-driven data receiving.
- status_t USART_TransferGetReceiveCount (USART_Type ∗base, usart_handle_t ∗handle, uint32_t ∗count)
  Get the number of bytes that have been received.
- void USART_TransferHandleIRQ (USART_Type ∗base, usart_handle_t ∗handle)
  USART IRQ handle function.

17.3.2 Data Structure Documentation

17.3.2.1 struct usart_config_t

Data Fields

- uint32_t baudRate_Bps
  USART baud rate.
- usart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- usart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- usart_data_len_t bitCountPerChar
  Data length - 7 bit, 8 bit.
- bool loopback
  Enable peripheral loopback.
- bool enableRx
  Enable RX.
- bool enableTx
  Enable TX.
- bool enableContinuousSCLK
  USART continuous Clock generation enable in synchronous master mode.
- usart_txfifo_watermark_t txWatermark
  txFIFO watermark
- usart_rxfifo_watermark_t rxWatermark
  rxFIFO watermark
- usart_sync_mode_t syncMode
  Transfer mode select - asynchronous, synchronous master, synchronous slave.
- usart_clock_polarity_t clockPolarity
  Selects the clock polarity and sampling edge in synchronous mode.

17.3.2.1.0.23 Field Documentation

17.3.2.1.0.23.1 bool usart_config_t::enableContinuousSCLK

17.3.2.1.0.23.2 usart_sync_mode_t usart_config_t::syncMode

17.3.2.1.0.23.3 usart_clock_polarity_t usart_config_t::clockPolarity

17.3.2.2 struct usart_transfer_t

Data Fields

- uint8_t ∗ data
  The buffer of data to be transfer.
- size_t dataSize
  The byte count to be transfer.

17.3.2.2.0.24 Field Documentation

17.3.2.2.0.24.1 uint8_t* usart_transfer_t::data

17.3.2.2.0.24.2 size_t usart_transfer_t::dataSize

17.3.2.3 struct _usart_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.
- usart_transfer_callback_t callback
  Callback function.
- void * userData
  USART callback function parameter.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.
- usart_txfifo_watermark_t txWatermark
  txFIFO watermark
- usart_rxfifo_watermark_t rxWatermark
  rxFIFO watermark

17.3.2.3.0.25 Field Documentation

17.3.2.3.0.25.1 uint8_t* volatile usart_handle_t::txData

17.3.2.3.0.25.2 volatile size_t usart_handle_t::txDataSize

17.3.2.3.0.25.3 size_t usart_handle_t::txDataSizeAll

17.3.2.3.0.25.4 uint8_t* volatile usart_handle_t::rxData

17.3.2.3.0.25.5 volatile size_t usart_handle_t::rxDataSize

17.3.2.3.0.25.6 size_t usart_handle_t::rxDataSizeAll

17.3.2.3.0.25.7 uint8_t* usart_handle_t::rxRingBuffer

17.3.2.3.0.25.8 size_t usart_handle_t::rxRingBufferSize

17.3.2.3.0.25.9 volatile uint16_t usart_handle_t::rxRingBufferHead

17.3.2.3.0.25.10 volatile uint16_t usart_handle_t::rxRingBufferTail

17.3.2.3.0.25.11 usart_transfer_callback_t usart_handle_t::callback

17.3.2.3.0.25.12 void* usart_handle_t::userData

17.3.2.3.0.25.13 volatile uint8_t usart_handle_t::txState

17.3.3 Macro Definition Documentation

17.3.3.1 #define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

17.3.4 Typedef Documentation

17.3.4.1 typedef void(* usart_transfer_callback_t)(USART_Type *base, usart_handle_t *handle, status_t status, void *userData)

17.3.5 Enumeration Type Documentation

17.3.5.1 enum _usart_status

Enumerator

- ***kStatus_USART_TxBusy*** Transmitter is busy.
- ***kStatus_USART_RxBusy*** Receiver is busy.
- ***kStatus_USART_TxIdle*** USART transmitter is idle.
- ***kStatus_USART_RxIdle*** USART receiver is idle.
- ***kStatus_USART_TxError*** Error happens on txFIFO.

kStatus_USART_RxError Error happens on rxFIFO.
kStatus_USART_RxRingBufferOverrun Error happens on rx ring buffer.
kStatus_USART_NoiseError USART noise error.
kStatus_USART_FramingError USART framing error.
kStatus_USART_ParityError USART parity error.
kStatus_USART_BaudrateNotSupport Baudrate is not support in current clock source.

17.3.5.2 enum usart_sync_mode_t

Enumerator

kUSART_SyncModeDisabled Asynchronous mode.
kUSART_SyncModeSlave Synchronous slave mode.
kUSART_SyncModeMaster Synchronous master mode.

17.3.5.3 enum usart_parity_mode_t

Enumerator

kUSART_ParityDisabled Parity disabled.
kUSART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kUSART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

17.3.5.4 enum usart_stop_bit_count_t

Enumerator

kUSART_OneStopBit One stop bit.
kUSART_TwoStopBit Two stop bits.

17.3.5.5 enum usart_data_len_t

Enumerator

kUSART_7BitsPerChar Seven bit mode.
kUSART_8BitsPerChar Eight bit mode.

17.3.5.6 enum usart_clock_polarity_t

Enumerator

kUSART_RxSampleOnFallingEdge Un_RXD is sampled on the falling edge of SCLK.
kUSART_RxSampleOnRisingEdge Un_RXD is sampled on the rising edge of SCLK.

17.3.5.7 enum usart_txfifo_watermark_t

Enumerator

kUSART_TxFifo0 USART tx watermark is empty.
kUSART_TxFifo1 USART tx watermark at 1 item.
kUSART_TxFifo2 USART tx watermark at 2 items.
kUSART_TxFifo3 USART tx watermark at 3 items.
kUSART_TxFifo4 USART tx watermark at 4 items.
kUSART_TxFifo5 USART tx watermark at 5 items.
kUSART_TxFifo6 USART tx watermark at 6 items.
kUSART_TxFifo7 USART tx watermark at 7 items.

17.3.5.8 enum usart_rxfifo_watermark_t

Enumerator

kUSART_RxFifo1 USART rx watermark at 1 item.
kUSART_RxFifo2 USART rx watermark at 2 items.
kUSART_RxFifo3 USART rx watermark at 3 items.
kUSART_RxFifo4 USART rx watermark at 4 items.
kUSART_RxFifo5 USART rx watermark at 5 items.
kUSART_RxFifo6 USART rx watermark at 6 items.
kUSART_RxFifo7 USART rx watermark at 7 items.
kUSART_RxFifo8 USART rx watermark at 8 items.

17.3.5.9 enum _usart_flags

This provides constants for the USART status flags for use in the USART functions.

Enumerator

kUSART_TxError TEERR bit, sets if TX buffer is error.
kUSART_RxError RXERR bit, sets if RX buffer is error.
kUSART_TxFifoEmptyFlag TXEMPTY bit, sets if TX buffer is empty.
kUSART_TxFifoNotFullFlag TXNOTFULL bit, sets if TX buffer is not full.
kUSART_RxFifoNotEmptyFlag RXNOEMPTY bit, sets if RX buffer is not empty.
kUSART_RxFifoFullFlag RXFULL bit, sets if RX buffer is full.

17.3.6 Function Documentation

17.3.6.1 uint32_t USART_GetInstance (USART_Type ∗ *base*)

17.3.6.2 status_t USART_Init (USART_Type ∗ *base*, const usart_config_t ∗ *config*, uint32_t *srcClock_Hz*)

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the USART_GetDefaultConfig() function. Example below shows how to use this API to configure USART.

```
*  usart_config_t usartConfig;
*  usartConfig.baudRate_Bps = 115200U;
*  usartConfig.parityMode = kUSART_ParityDisabled;
*  usartConfig.stopBitCount = kUSART_OneStopBit;
*  USART_Init(USART1, &usartConfig, 20000000U);
*
```

Parameters

base	USART peripheral base address.
config	Pointer to user-defined configuration structure.
srcClock_Hz	USART clock source frequency in HZ.

Return values

kStatus_USART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_InvalidArgument	USART base address is not valid
kStatus_Success	Status USART initialize succeed

17.3.6.3 void USART_Deinit (USART_Type ∗ *base*)

This function waits for TX complete, disables TX and RX, and disables the USART clock.

Parameters

base	USART peripheral base address.

17.3.6.4 void USART_GetDefaultConfig (usart_config_t ∗ *config*)

This function initializes the USART configuration structure to a default value. The default values are: usartConfig->baudRate_Bps = 115200U; usartConfig->parityMode = kUSART_ParityDisabled; usart-

Config->stopBitCount = kUSART_OneStopBit; usartConfig->bitCountPerChar = kUSART_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;

Parameters

config	Pointer to configuration structure.

17.3.6.5 status_t USART_SetBaudRate (USART_Type * *base*, uint32_t *baudrate_Bps*, uint32_t *srcClock_Hz*)

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART_Init.

```
*  USART_SetBaudRate(USART1, 115200U, 20000000U);
*
```

Parameters

base	USART peripheral base address.
baudrate_Bps	USART baudrate to be set.
srcClock_Hz	USART clock source frequency in HZ.

Return values

kStatus_USART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	Set baudrate succeed.
kStatus_InvalidArgument	One or more arguments are invalid.

17.3.6.6 static uint32_t USART_GetStatusFlags (USART_Type * *base*) [inline], [static]

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators _usart_flags. To check a specific status, compare the return value with enumerators in _usart_flags. For example, to check whether the TX is empty:

```
*     if (kUSART_TxFifoNotFullFlag &
      USART_GetStatusFlags(USART1))
*     {
*         ...
*     }
*
```

Parameters

base	USART peripheral base address.

Returns

USART status flags which are ORed by the enumerators in the _usart_flags.

17.3.6.7 static void USART_ClearStatusFlags (USART_Type * *base,* uint32_t *mask*) [inline], [static]

This function clear supported USART status flags Flags that can be cleared or set are: kUSART_TxError kUSART_RxError For example:

```
*     USART_ClearStatusFlags(USART1, kUSART_TxError |
      kUSART_RxError)
*
```

Parameters

base	USART peripheral base address.
mask	status flags to be cleared.

17.3.6.8 static void USART_EnableInterrupts (USART_Type * *base,* uint32_t *mask*) [inline], [static]

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _usart_interrupt_enable. For example, to enable TX empty interrupt and RX full interrupt:

```
*     USART_EnableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
      kUSART_RxLevelInterruptEnable);
*
```

Parameters

base	USART peripheral base address.

mask	The interrupts to enable. Logical OR of _usart_interrupt_enable.

17.3.6.9 static void USART_DisableInterrupts (USART_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _usart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     USART_DisableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
      kUSART_RxLevelInterruptEnable);
*
```

Parameters

base	USART peripheral base address.
mask	The interrupts to disable. Logical OR of _usart_interrupt_enable.

17.3.6.10 static uint32_t USART_GetEnabledInterrupts (USART_Type * *base*) [inline], [static]

This function returns the enabled USART interrupts.

Parameters

base	USART peripheral base address.

17.3.6.11 static void USART_EnableCTS (USART_Type * *base*, bool *enable*) [inline], [static]

This function will determine whether CTS is used for flow control.

Parameters

base	USART peripheral base address.
enable	Enable CTS or not, true for enable and false for disable.

17.3.6.12 static void USART_EnableContinuousSCLK (USART_Type * *base*, bool *enable*) [inline], [static]

By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this funciton, SCLK will run continuously in synchronous mode, allowing characters to be received on Un_RxD independently from transmission on Un_TXD).

Parameters

base	USART peripheral base address.
enable	Enable Continuous Clock generation mode or not, true for enable and false for disable.

17.3.6.13 static void USART_EnableAutoClearSCLK (USART_Type * *base*, bool *enable*) [inline], [static]

While enable this cuntion, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

base	USART peripheral base address.
enable	Enable auto clear or not, true for enable and false for disable.

17.3.6.14 static void USART_WriteByte (USART_Type * *base*, uint8_t *data*) [inline], [static]

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

Parameters

base	USART peripheral base address.
data	The byte to write.

17.3.6.15 static uint8_t USART_ReadByte (USART_Type * *base*) [inline], [static]

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

Parameters

base	USART peripheral base address.

Returns

The byte read from USART data register.

17.3.6.16 void USART_WriteBlocking (USART_Type ∗ *base*, 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.

Parameters

base	USART peripheral base address.
data	Start address of the data to write.
length	Size of the data to write.

17.3.6.17 status_t USART_ReadBlocking (USART_Type ∗ *base*, 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 read data from the TX register.

Parameters

base	USART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_USART_-FramingError	Receiver overrun happened while receiving data.
kStatus_USART_Parity-Error	Noise error happened while receiving data.
kStatus_USART_Noise-Error	Framing error happened while receiving data.
kStatus_USART_RxError	Overflow or underflow rxFIFO happened.
kStatus_Success	Successfully received all data.

17.3.6.18 status_t USART_TransferCreateHandle (USART_Type * *base*, usart_handle_t * *handle*, usart_transfer_callback_t *callback*, void * *userData*)

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

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

17.3.6.19 status_t USART_TransferSendNonBlocking (USART_Type * *base*, usart_handle_t * *handle*, usart_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 the IRQ handler, the USART driver calls the callback function and passes the kStatus_USART_TxIdle as status parameter.

Note

> The kStatus_USART_TxIdle is passed to the upper layer when all data is written to the TX register. However it does not ensure that all data are sent out. Before disabling the TX, check the kUSART-_TransmissionCompleteFlag to ensure that the TX is finished.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
xfer	USART transfer structure. See usart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_USART_TxBusy	Previous transmission still not finished, data not all written to TX register yet.
kStatus_InvalidArgument	Invalid argument.

17.3.6.20 void USART_TransferStartRingBuffer (USART_Type * *base*, usart_handle_t * *handle*, uint8_t * *ringBuffer*, size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the USART_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 the 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	USART peripheral base address.
handle	USART handle pointer.
ringBuffer	Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	size of the ring buffer.

17.3.6.21 void USART_TransferStopRingBuffer (USART_Type * *base*, usart_handle_t * *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

17.3.6.22 size_t USART_TransferGetRxRingBufferLength (usart_handle_t * *handle*)

Parameters

handle	USART handle pointer.

Returns

Length of received data in RX ring buffer.

17.3.6.23 void USART_TransferAbortSend (USART_Type * *base*, usart_handle_t * *handle*)

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are still not sent out.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

17.3.6.24 status_t USART_TransferGetSendCount (USART_Type ∗ *base*, usart_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes that have been written to USART TX register by interrupt method.

Parameters

base	USART peripheral base address.
handle	USART 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`;

17.3.6.25 status_t USART_TransferReceiveNonBlocking (USART_Type ∗ *base*, usart_handle_t ∗ *handle*, usart_transfer_t ∗ *xfer*, size_t ∗ *receivedBytes*)

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter kStatus_USART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the xfer->data and this function returns with the parameter `receivedBytes` set to 5. For the left 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the USART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the xfer->data. When all data is received, the upper layer is notified.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
xfer	USART transfer structure, see usart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into transmit queue.
kStatus_USART_RxBusy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

17.3.6.26 void USART_TransferAbortReceive (USART_Type ∗ *base*, usart_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	USART peripheral base address.
handle	USART handle pointer.

17.3.6.27 status_t USART_TransferGetReceiveCount (USART_Type ∗ *base*, usart_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes that have been received.

Parameters

base	USART peripheral base address.
handle	USART 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`;

17.3.6.28 void USART_TransferHandleIRQ (USART_Type * *base*, usart_handle_t * *handle*)

This function handles the USART transmit and receive IRQ request.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

17.4 USART DMA Driver

17.4.1 Overview

Files

- file fsl_usart_dma.h

Data Structures

- struct usart_dma_handle_t
  UART DMA handle. More...

Typedefs

- typedef void(∗ usart_dma_transfer_callback_t)(USART_Type ∗base, usart_dma_handle_t ∗handle, status_t status, void ∗userData)
  UART transfer callback function.

Driver version

- #define FSL_USART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  USART dma driver version 2.0.1.

DMA transactional

- status_t USART_TransferCreateHandleDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the USART handle which is used in transactional functions.
- status_t USART_TransferSendDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_transfer_t ∗xfer)
  Sends data using DMA.
- status_t USART_TransferReceiveDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_transfer_t ∗xfer)
  Receives data using DMA.
- void USART_TransferAbortSendDMA (USART_Type ∗base, usart_dma_handle_t ∗handle)
  Aborts the sent data using DMA.
- void USART_TransferAbortReceiveDMA (USART_Type ∗base, usart_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t USART_TransferGetReceiveCountDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, uint32_t ∗count)
  Get the number of bytes that have been received.

17.4.2 Data Structure Documentation

17.4.2.1 struct _usart_dma_handle

Data Fields

- USART_Type * base
  UART peripheral base address.
- usart_dma_transfer_callback_t callback
  Callback function.
- void * userData
  UART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- 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.

17.4.2.1.0.26 Field Documentation

17.4.2.1.0.26.1 USART_Type* usart_dma_handle_t::base

17.4.2.1.0.26.2 usart_dma_transfer_callback_t usart_dma_handle_t::callback

17.4.2.1.0.26.3 void* usart_dma_handle_t::userData

17.4.2.1.0.26.4 size_t usart_dma_handle_t::rxDataSizeAll

17.4.2.1.0.26.5 size_t usart_dma_handle_t::txDataSizeAll

17.4.2.1.0.26.6 dma_handle_t* usart_dma_handle_t::txDmaHandle

17.4.2.1.0.26.7 dma_handle_t* usart_dma_handle_t::rxDmaHandle

17.4.2.1.0.26.8 volatile uint8_t usart_dma_handle_t::txState

17.4.3 Macro Definition Documentation

17.4.3.1 #define FSL_USART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

17.4.4 Typedef Documentation

17.4.4.1 typedef void(* usart_dma_transfer_callback_t)(USART_Type *base, usart_dma_handle_t *handle, status_t status, void *userData)

17.4.5 Function Documentation

17.4.5.1 status_t USART_TransferCreateHandleDMA (USART_Type * *base,* usart_dma_handle_t * *handle,* usart_dma_transfer_callback_t *callback,* void * *userData,* dma_handle_t * *txDmaHandle,* dma_handle_t * *rxDmaHandle*)

Parameters

base	USART peripheral base address.
handle	Pointer to usart_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.

17.4.5.2 status_t USART_TransferSendDMA (USART_Type ∗ *base,* usart_dma_handle_t ∗ *handle,* usart_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	USART peripheral base address.
handle	USART handle pointer.
xfer	USART DMA transfer structure. See usart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_USART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

17.4.5.3 status_t USART_TransferReceiveDMA (USART_Type ∗ *base,* usart_dma_handle_t ∗ *handle,* usart_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	USART peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	USART DMA transfer structure. See usart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_USART_RxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

17.4.5.4 void USART_TransferAbortSendDMA (USART_Type * *base*, usart_dma_handle_t * *handle*)

This function aborts send data using DMA.

Parameters

base	USART peripheral base address
handle	Pointer to usart_dma_handle_t structure

17.4.5.5 void USART_TransferAbortReceiveDMA (USART_Type * *base*, usart_dma_handle_t * *handle*)

This function aborts the received data using DMA.

Parameters

base	USART peripheral base address
handle	Pointer to usart_dma_handle_t structure

17.4.5.6 status_t USART_TransferGetReceiveCountDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been received.

Parameters

base	USART peripheral base address.
handle	USART 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`;

17.5 USART FreeRTOS Driver

17.5.1 Overview

Files

- file fsl_usart_freertos.h

Data Structures

- struct rtos_usart_config
  FLEX USART configuration structure. More...
- struct usart_rtos_handle_t
  FLEX USART FreeRTOS handle. More...

Driver version

- #define FSL_USART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  USART freertos driver version 2.0.1.

USART RTOS Operation

- int USART_RTOS_Init (usart_rtos_handle_t ∗handle, usart_handle_t ∗t_handle, const struct rtos_usart_config ∗cfg)
  Initializes a USART instance for operation in RTOS.
- int USART_RTOS_Deinit (usart_rtos_handle_t ∗handle)
  Deinitializes a USART instance for operation.

USART transactional Operation

- int USART_RTOS_Send (usart_rtos_handle_t ∗handle, const uint8_t ∗buffer, uint32_t length)
  Sends data in the background.
- int USART_RTOS_Receive (usart_rtos_handle_t ∗handle, uint8_t ∗buffer, uint32_t length, size_t ∗received)
  Receives data.

17.5.2 Data Structure Documentation

17.5.2.1 struct rtos_usart_config

Data Fields

- USART_Type ∗ base

USART base address.
- uint32_t srcclk
  USART source clock in Hz.
- uint32_t baudrate
  Desired communication speed.
- usart_parity_mode_t parity
  Parity setting.
- usart_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.

17.5.2.2 struct usart_rtos_handle_t

Data Fields

- USART_Type * base
  USART base address.
- usart_transfer_t txTransfer
  TX transfer structure.
- usart_transfer_t rxTransfer
  RX transfer structure.
- SemaphoreHandle_t rxSemaphore
  RX semaphore for resource sharing.
- SemaphoreHandle_t txSemaphore
  TX semaphore for resource sharing.
- EventGroupHandle_t rxEvent
  RX completion event.
- EventGroupHandle_t txEvent
  TX completion event.
- void * t_state
  Transactional state of the underlying driver.

17.5.3 Macro Definition Documentation

17.5.3.1 #define FSL_USART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

17.5.4 Function Documentation

17.5.4.1 int USART_RTOS_Init (usart_rtos_handle_t * *handle*, usart_handle_t * *t_handle*, const struct rtos_usart_config * *cfg*)

Parameters

handle	The RTOS USART handle, the pointer to allocated space for RTOS context.
t_handle	The pointer to allocated space where to store transactional layer internal state.
cfg	The pointer to the parameters required to configure the USART after initialization.

Returns

0 succeed, others fail.

17.5.4.2 int USART_RTOS_Deinit (usart_rtos_handle_t * *handle*)

This function deinitializes the USART module, sets all register values to reset value, and releases the resources.

Parameters

handle	The RTOS USART handle.

17.5.4.3 int USART_RTOS_Send (usart_rtos_handle_t * *handle,* const uint8_t * *buffer,* uint32_t *length*)

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

handle	The RTOS USART handle.
buffer	The pointer to buffer to send.
length	The number of bytes to send.

17.5.4.4 int USART_RTOS_Receive (usart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length,* size_t * *received*)

This function receives data from USART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

Parameters

handle	The RTOS USART 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 18
FMEAS: Frequency Measure Driver

18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Frequency Measure function of MCUXpresso SDK devices' SYSCON module.

It measures frequency of any on-chip or off-chip clock signal. The more precise and higher accuracy clock is selected as a reference clock. The resulting frequency is internally computed from the ratio of value of selected target and reference clock counters.

18.2 Frequency Measure Driver operation

INPUTMUX_AttachSignal() function has to be used to select reference and target clock signal sources.

FMEAS_StartMeasure() function starts the measurement cycle.

FMEAS_IsMeasureComplete() can be polled to check if the measurement cycle has finished.

FMEAS_GetFrequency() returns the frequency of the target clock. Frequency of the reference clock has to be provided as a parameter.

18.3 Typical use case

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

Files

- file fsl_fmeas.h

Macros

- #define FMEAS_INDEX 20
  The calibration duration is 2^FMEAS_INDEX times the reference clock period.

Driver version

- #define FSL_FMEAS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  Defines LPC Frequency Measure driver version 2.1.1.

FMEAS Functional Operation

- static void FMEAS_StartMeasure (FMEAS_SYSCON_Type ∗base)
  Starts a frequency measurement cycle.
- static void FMEAS_StartMeasureWithScale (FMEAS_SYSCON_Type ∗base, uint8_t scale)
  Starts a frequency measurement cycle with specific time.

- static bool FMEAS_IsMeasureComplete (FMEAS_SYSCON_Type *base)
  Indicates when a frequency measurement cycle is complete.
- uint32_t FMEAS_GetFrequency (FMEAS_SYSCON_Type *base, uint32_t refClockRate)
  Returns the computed value for a frequency measurement cycle.
- void FMEAS_GetCountWithScale (FMEAS_SYSCON_Type *base, uint8_t scale, uint32_t *refClockCount, uint32_t *targetClockCount)
  Get the clock count during the measurement time.

18.4 Macro Definition Documentation

18.4.1 #define FSL_FMEAS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

18.5 Function Documentation

18.5.1 static void FMEAS_StartMeasure (FMEAS_SYSCON_Type * *base*) [inline], [static]

Parameters

base	: SYSCON peripheral base address.

18.5.2 static void FMEAS_StartMeasureWithScale (FMEAS_SYSCON_Type * *base*, uint8_t *scale*) [inline], [static]

Parameters

base	: SYSCON peripheral base address.
scale	: measurement time is 2^scale cycle of reference clock, value is from 2 to 31.

18.5.3 static bool FMEAS_IsMeasureComplete (FMEAS_SYSCON_Type * *base*) [inline], [static]

Parameters

base	: SYSCON peripheral base address.

Returns

true if a measurement cycle is active, otherwise false.

18.5.4 uint32_t FMEAS_GetFrequency (FMEAS_SYSCON_Type * *base*, uint32_t *refClockRate*)

Parameters

base	: SYSCON peripheral base address.
refClockRate	: Reference clock rate used during the frequency measurement cycle.

Returns

Frequency in Hz.

18.5.5 void FMEAS_GetCountWithScale (FMEAS_SYSCON_Type * *base*, uint8_t *scale*, uint32_t * *refClockCount*, uint32_t * *targetClockCount*)

Parameters

base	: SYSCON peripheral base address.
scale	: measurement time is 2^scale cycle of reference clock, value is from 2 to 31.
refClockCount	: Reference clock cycle during the measurement time.
targetClock-Count	: Target clock cycle during the measurement time.

Chapter 19
GINT: Group GPIO Input Interrupt Driver

19.1 Overview

The MCUXpresso SDK provides a driver for the Group GPIO Input Interrupt (GINT).

It can configure one or more pins to generate a group interrupt when the pin conditions are met. The pins do not have to be configured as GPIO pins.

19.2 Group GPIO Input Interrupt Driver operation

GINT_SetCtrl() and GINT_ConfigPins() functions configure the pins.

GINT_EnableCallback() function enables the callback functionality. Callback function is called when the pin conditions are met.

19.3 Typical use case

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

Files

- file fsl_gint.h

Typedefs

- typedef void(∗ gint_cb_t)(void)
  GINT Callback function.

Enumerations

- enum gint_comb_t {
  kGINT_CombineOr = 0U,
  kGINT_CombineAnd = 1U }
  GINT combine inputs type.
- enum gint_trig_t {
  kGINT_TrigEdge = 0U,
  kGINT_TrigLevel = 1U }
  GINT trigger type.

Functions

- void GINT_Init (GINT_Type ∗base)
  Initialize GINT peripheral.
- void GINT_SetCtrl (GINT_Type ∗base, gint_comb_t comb, gint_trig_t trig, gint_cb_t callback)

Setup GINT peripheral control parameters.
- void GINT_GetCtrl (GINT_Type ∗base, gint_comb_t ∗comb, gint_trig_t ∗trig, gint_cb_t ∗callback)
  Get GINT peripheral control parameters.
- void GINT_ConfigPins (GINT_Type ∗base, gint_port_t port, uint32_t polarityMask, uint32_t enableMask)
  Configure GINT peripheral pins.
- void GINT_GetConfigPins (GINT_Type ∗base, gint_port_t port, uint32_t ∗polarityMask, uint32_t ∗enableMask)
  Get GINT peripheral pin configuration.
- void GINT_EnableCallback (GINT_Type ∗base)
  Enable callback.
- void GINT_DisableCallback (GINT_Type ∗base)
  Disable callback.
- static void GINT_ClrStatus (GINT_Type ∗base)
  Clear GINT status.
- static uint32_t GINT_GetStatus (GINT_Type ∗base)
  Get GINT status.
- void GINT_Deinit (GINT_Type ∗base)
  Deinitialize GINT peripheral.

Driver version

- #define FSL_GINT_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  Version 2.0.1.

19.4 Macro Definition Documentation

19.4.1 #define FSL_GINT_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

19.5 Typedef Documentation

19.5.1 typedef void(∗ gint_cb_t)(void)

19.6 Enumeration Type Documentation

19.6.1 enum gint_comb_t

Enumerator

kGINT_CombineOr A grouped interrupt is generated when any one of the enabled inputs is active.
kGINT_CombineAnd A grouped interrupt is generated when all enabled inputs are active.

19.6.2 enum gint_trig_t

Enumerator

kGINT_TrigEdge Edge triggered based on polarity.
kGINT_TrigLevel Level triggered based on polarity.

19.7 Function Documentation

19.7.1 void GINT_Init (GINT_Type * *base*)

This function initializes the GINT peripheral and enables the clock.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

19.7.2 void GINT_SetCtrl (GINT_Type ∗ *base*, gint_comb_t *comb*, gint_trig_t *trig*, gint_cb_t *callback*)

This function sets the control parameters of GINT peripheral.

Parameters

base	Base address of the GINT peripheral.
comb	Controls if the enabled inputs are logically ORed or ANDed for interrupt generation.
trig	Controls if the enabled inputs are level or edge sensitive based on polarity.
callback	This function is called when configured group interrupt is generated.

Return values

None.	

19.7.3 void GINT_GetCtrl (GINT_Type ∗ *base*, gint_comb_t ∗ *comb*, gint_trig_t ∗ *trig*, gint_cb_t ∗ *callback*)

This function returns the control parameters of GINT peripheral.

Parameters

base	Base address of the GINT peripheral.
comb	Pointer to store combine input value.
trig	Pointer to store trigger value.

callback	Pointer to store callback function.

Return values

None.	

19.7.4 void GINT_ConfigPins (GINT_Type ∗ *base*, gint_port_t *port*, uint32_t *polarityMask*, uint32_t *enableMask*)

This function enables and controls the polarity of enabled pin(s) of a given port.

Parameters

base	Base address of the GINT peripheral.
port	Port number.
polarityMask	Each bit position selects the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.
enableMask	Each bit position selects if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

Return values

None.	

19.7.5 void GINT_GetConfigPins (GINT_Type ∗ *base*, gint_port_t *port*, uint32_t ∗ *polarityMask*, uint32_t ∗ *enableMask*)

This function returns the pin configuration of a given port.

Parameters

base	Base address of the GINT peripheral.
port	Port number.
polarityMask	Pointer to store the polarity mask Each bit position indicates the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.

enableMask	Pointer to store the enable mask. Each bit position indicates if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

Return values

None.	

19.7.6 void GINT_EnableCallback (GINT_Type * *base*)

This function enables the interrupt for the selected GINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

19.7.7 void GINT_DisableCallback (GINT_Type * *base*)

This function disables the interrupt for the selected GINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

base	Base address of the peripheral.

Return values

None.	

19.7.8 static void GINT_ClrStatus (GINT_Type * *base*) `[inline]`, `[static]`

This function clears the GINT status bit.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

19.7.9 static uint32_t GINT_GetStatus (GINT_Type ∗ *base*) [inline], [static]

This function returns the GINT status.

Parameters

base	Base address of the GINT peripheral.

Return values

status	= 0 No group interrupt request. = 1 Group interrupt request active.

19.7.10 void GINT_Deinit (GINT_Type ∗ *base*)

This function disables the GINT clock.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

Chapter 20
GPIO: General Purpose I/O

20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Purpose I/O (GPIO) module of MCUXpresso SDK devices.

20.2 Function groups

20.2.1 Initialization and deinitialization

The function GPIO_PinInit() initializes the GPIO with specified configuration.

20.2.2 Pin manipulation

The function GPIO_PinWrite() set output state of selected GPIO pin. The function GPIO_PinRead() read input value of selected GPIO pin.

20.2.3 Port manipulation

The function GPIO_PortSet() sets the output level of selected GPIO pins to the logic 1. The function GPIO_PortClear() sets the output level of selected GPIO pins to the logic 0. The function GPIO_PortToggle() reverse the output level of selected GPIO pins. The function GPIO_PortRead() read input value of selected port.

20.2.4 Port masking

The function GPIO_PortMaskedSet() set port mask, only pins masked by 0 will be enabled in following functions. The function GPIO_PortMaskedWrite() sets the state of selected GPIO port, only pins masked by 0 will be affected. The function GPIO_PortMaskedRead() reads the state of selected GPIO port, only pins masked by 0 are enabled for read, pins masked by 1 are read as 0.

20.3 Typical use case

Example use of GPIO API. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/gpio

Files

- file fsl_gpio.h

Data Structures

- struct gpio_pin_config_t
  The GPIO pin configuration structure. More...

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  LPC GPIO direction definition.

Functions

- static void GPIO_PortSet (GPIO_Type ∗base, uint32_t port, 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 port, 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 port, uint32_t mask)
  Reverses current output logic of the multiple GPIO pins.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))
  LPC GPIO driver version 2.1.3.

GPIO Configuration

- void GPIO_PortInit (GPIO_Type ∗base, uint32_t port)
  Initializes the GPIO peripheral.
- void GPIO_PinInit (GPIO_Type ∗base, uint32_t port, uint32_t pin, const gpio_pin_config_t ∗config)
  Initializes a GPIO pin used by the board.

GPIO Output Operations

- static void GPIO_PinWrite (GPIO_Type ∗base, uint32_t port, uint32_t pin, uint8_t output)
  Sets the output level of the one GPIO pin to the logic 1 or 0.

GPIO Input Operations

- static uint32_t GPIO_PinRead (GPIO_Type ∗base, uint32_t port, uint32_t pin)
  Reads the current input value of the GPIO PIN.

20.4 Data Structure Documentation

20.4.1 struct gpio_pin_config_t

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

Data Fields

- gpio_pin_direction_t pinDirection
  GPIO direction, input or output.
- uint8_t outputLogic
  Set default output logic, no use in input.

20.5 Macro Definition Documentation

20.5.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))

20.6 Enumeration Type Documentation

20.6.1 enum gpio_pin_direction_t

Enumerator

kGPIO_DigitalInput Set current pin as digital input.
kGPIO_DigitalOutput Set current pin as digital output.

20.7 Function Documentation

20.7.1 void GPIO_PortInit (GPIO_Type * *base*, uint32_t *port*)

This function ungates the GPIO clock.

Parameters

base	GPIO peripheral base pointer.
port	GPIO port number.

20.7.2 void GPIO_PinInit (GPIO_Type * *base*, uint32_t *port*, uint32_t *pin*, const gpio_pin_config_t * *config*)

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or output pin configuration:

```
* // Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* //Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number
config	GPIO pin configuration pointer

20.7.3 static void GPIO_PinWrite (GPIO_Type * *base*, uint32_t *port*, uint32_t *pin*, uint8_t *output*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number
output	GPIO pin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

20.7.4 static uint32_t GPIO_PinRead (GPIO_Type * *base*, uint32_t *port*, uint32_t *pin*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number

Return values

GPIO	port input value • 0: corresponding pin input low-logic level. • 1: corresponding pin input high-logic level.

20.7.5 static void GPIO_PortSet (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

20.7.6 static void GPIO_PortClear (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

20.7.7 static void GPIO_PortToggle (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

Chapter 21
INPUTMUX: Input Multiplexing Driver

21.1 Overview

The MCUXpresso SDK provides a driver for the Input multiplexing (INPUTMUX).

It configures the inputs to the pin interrupt block, DMA trigger, and frequency measure function. Once configured, the clock is not needed for the inputmux.

21.2 Input Multiplexing Driver operation

INPUTMUX_AttachSignal function configures the specified input

21.3 Typical use case

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

Files

- file fsl_inputmux.h
- file fsl_inputmux_connections.h

Macros

- #define PINTSEL_PMUX_ID (offsetof(INPUTMUX_Type, PINTSEL))
  Periphinmux IDs.
- #define DMA_ITRIG_PMUX_ID (offsetof(INPUTMUX_Type, DMA_ITRIG_INMUX))
  0xE0U
- #define DMA_OTRIG_PMUX_ID (offsetof(INPUTMUX_Type, DMA_OTRIG_INMUX))
  0x160U
- #define FREQMEAS_PMUX_ID (offsetof(INPUTMUX_Type, FREQMEAS_REF))
  0x180U
- #define PMUX_SHIFT 20U
  20U

Enumerations

- enum inputmux_connection_t {
  kINPUTMUX_ClkInToFreqmeas = 0U + (FREQMEAS_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Xtal32MhzToFreqmeas = 1U + (FREQMEAS_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Fro1MhzToFreqmeas = 2U + (FREQMEAS_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_32KhzOscToFreqmeas = 3U + (FREQMEAS_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_MainClkToFreqmeas = 4U + (FREQMEAS_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_GpioPort0Pin4ToFreqmeas,
  kINPUTMUX_GpioPort0Pin20ToFreqmeas,
  kINPUTMUX_GpioPort0Pin16ToFreqmeas,
  kINPUTMUX_GpioPort0Pin15ToFreqmeas,
  kINPUTMUX_GpioPort0Pin0ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 0U),
  kINPUTMUX_GpioPort0Pin1ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 1U),
  kINPUTMUX_GpioPort0Pin2ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 2U),
  kINPUTMUX_GpioPort0Pin3ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 3U),
  kINPUTMUX_GpioPort0Pin4ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 4U),
  kINPUTMUX_GpioPort0Pin5ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 5U),
  kINPUTMUX_GpioPort0Pin6ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 6U),
  kINPUTMUX_GpioPort0Pin7ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 7U),
  kINPUTMUX_GpioPort0Pin8ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 8U),
  kINPUTMUX_GpioPort0Pin9ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 9U),
  kINPUTMUX_GpioPort0Pin10ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 10U),
  kINPUTMUX_GpioPort0Pin11ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 11U),
  kINPUTMUX_GpioPort0Pin12ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 12U),
  kINPUTMUX_GpioPort0Pin13ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 13U),
  kINPUTMUX_GpioPort0Pin14ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 14U),
  kINPUTMUX_GpioPort0Pin15ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 15U),
  kINPUTMUX_GpioPort0Pin16ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 16U),
  kINPUTMUX_GpioPort0Pin17ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 17U),
  kINPUTMUX_GpioPort0Pin18ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 18U),
  kINPUTMUX_GpioPort0Pin19ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 19U),
  kINPUTMUX_GpioPort0Pin20ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 20U),
  kINPUTMUX_GpioPort0Pin21ToPintsel = INPUTMUX_GpioPortPinToPintsel(0, 21U),
  kINPUTMUX_Adc0SeqaIrqToDma = 0U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Adc0SeqbIrqToDma = 1U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Ctimer0M0ToDma = 2U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Ctimer0M1ToDma = 3U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Ctimer1M0ToDma = 4U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Ctimer1M1ToDma = 5U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_PinInt0ToDma = 6U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_PinInt1ToDma = 7U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_PinInt2ToDma = 8U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_PinInt3ToDma = 9U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_AesRxToDma = 10U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_AesTxToDma = 11U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_HashRxToDma = 12U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_HashTxToDma = 13U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Otrig0ToDma = 14U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
  kINPUTMUX_Otrig1ToDma = 15U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),

MUX_SHIFT),
kINPUTMUX_DmaUsart0TxTrigoutToTriginChannels = 1U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT),
kINPUTMUX_DmaUsart1RxTrigoutToTriginChannels = 2U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT),
kINPUTMUX_DmaUsart1TxTrigoutToTriginChannels = 3U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT),
kINPUTMUX_DmaI2c0SlvaeTrigoutToTriginChannels = 4U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT),
kINPUTMUX_DmaI2c0MasterTrigoutToTriginChannels = 5U + (DMA_OTRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaI2c1SlvaeTrigoutToTriginChannels = 6U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT),
kINPUTMUX_DmaI2c1MasterTrigoutToTriginChannels = 7U + (DMA_OTRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaSpi0RxTrigoutToTriginChannels = 8U + (DMA_OTRIG_PMUX_ID << PM-UX_SHIFT),
kINPUTMUX_DmaSpi0TxTrigoutToTriginChannels = 9U + (DMA_OTRIG_PMUX_ID << PM-UX_SHIFT),
kINPUTMUX_DmaSpi1RxTrigoutToTriginChannels = 10U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT),
kINPUTMUX_DmaSpi1TxTrigoutToTriginChannels = 11U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT),
kINPUTMUX_DmaSpifi0TrigoutToTriginChannels = 12U + (DMA_OTRIG_PMUX_ID << PM-UX_SHIFT),
kINPUTMUX_DmaI2c2SlaveTrigoutToTriginChannels = 13U + (DMA_OTRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaI2c2MasterTrigoutToTriginChannels = 14U + (DMA_OTRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaDmic0Ch0TrigoutToTriginChannels = 15U + (DMA_OTRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaDmic0Ch1TrigoutToTriginChannels = 16U + (DMA_OTRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaHash0RxTrigoutToTriginChannels = 17U + (DMA_OTRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaHash0TxTrigoutToTriginChannels = 18U + (DMA_OTRIG_PMUX_ID << P-MUX_SHIFT) }

INPUTMUX connections type.

Functions

- void INPUTMUX_Init (INPUTMUX_Type ∗base)
  Initialize INPUTMUX peripheral.
- void INPUTMUX_AttachSignal (INPUTMUX_Type ∗base, uint32_t index, inputmux_connection_t connection)
  Attaches a signal.

- void INPUTMUX_Deinit (INPUTMUX_Type *base)
  Deinitialize INPUTMUX peripheral.

Driver version

- #define FSL_INPUTMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  Group interrupt driver version for SDK.

21.4 Macro Definition Documentation

21.4.1 #define PINTSEL_PMUX_ID (offsetof(INPUTMUX_Type, PINTSEL))

0xC0U

21.4.2 #define FSL_INPUTMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Version 2.0.1.

21.5 Enumeration Type Documentation

21.5.1 enum inputmux_connection_t

Enumerator

kINPUTMUX_ClkInToFreqmeas Clock Input to Frequency measure.
kINPUTMUX_Xtal32MhzToFreqmeas XTAL 32MHZ to Frequency measure.
kINPUTMUX_Fro1MhzToFreqmeas Fro 1MHz to Frequency measure.
kINPUTMUX_32KhzOscToFreqmeas 32KHz OSC to Frequency measure.
kINPUTMUX_MainClkToFreqmeas Main Clock to Frequency measure.
kINPUTMUX_GpioPort0Pin4ToFreqmeas GPIO PORT 0 Pin 4 to Frequency measure.
kINPUTMUX_GpioPort0Pin20ToFreqmeas GPIO Port 0 Pin 20 to Frequency measure.
kINPUTMUX_GpioPort0Pin16ToFreqmeas GPIO Port 0 Pin 16 to Frequency measure.
kINPUTMUX_GpioPort0Pin15ToFreqmeas GPIO Port 0 Pin 15 to Frequency measure. Pin Interrupt.
kINPUTMUX_GpioPort0Pin0ToPintsel Port 0 Pin 0 to PINT select.
kINPUTMUX_GpioPort0Pin1ToPintsel Port 0 Pin 1 to PINT select.
kINPUTMUX_GpioPort0Pin2ToPintsel Port 0 Pin 2 to PINT select.
kINPUTMUX_GpioPort0Pin3ToPintsel Port 0 Pin 3 to PINT select.
kINPUTMUX_GpioPort0Pin4ToPintsel Port 0 Pin 4 to PINT select.
kINPUTMUX_GpioPort0Pin5ToPintsel Port 0 Pin 5 to PINT select.
kINPUTMUX_GpioPort0Pin6ToPintsel Port 0 Pin 6 to PINT select.
kINPUTMUX_GpioPort0Pin7ToPintsel Port 0 Pin 7 to PINT select.
kINPUTMUX_GpioPort0Pin8ToPintsel Port 0 Pin 8 to PINT select.
kINPUTMUX_GpioPort0Pin9ToPintsel Port 0 Pin 9 to PINT select.
kINPUTMUX_GpioPort0Pin10ToPintsel Port 0 Pin 10 to PINT select.

kINPUTMUX_GpioPort0Pin11ToPintsel Port 0 Pin 11 to PINT select.
kINPUTMUX_GpioPort0Pin12ToPintsel Port 0 Pin 12 to PINT select.
kINPUTMUX_GpioPort0Pin13ToPintsel Port 0 Pin 13 to PINT select.
kINPUTMUX_GpioPort0Pin14ToPintsel Port 0 Pin 14 to PINT select.
kINPUTMUX_GpioPort0Pin15ToPintsel Port 0 Pin 15 to PINT select.
kINPUTMUX_GpioPort0Pin16ToPintsel Port 0 Pin 16 to PINT select.
kINPUTMUX_GpioPort0Pin17ToPintsel Port 0 Pin 17 to PINT select.
kINPUTMUX_GpioPort0Pin18ToPintsel Port 0 Pin 18 to PINT select.
kINPUTMUX_GpioPort0Pin19ToPintsel Port 0 Pin 19 to PINT select.
kINPUTMUX_GpioPort0Pin20ToPintsel Port 0 Pin 20 to PINT select.
kINPUTMUX_GpioPort0Pin21ToPintsel Port 0 Pin 21 to PINT select. DMA ITRIG.
kINPUTMUX_Adc0SeqaIrqToDma ADC Interrupt (Sequence A)
kINPUTMUX_Adc0SeqbIrqToDma ADC Interrupt (Sequence B)
kINPUTMUX_Ctimer0M0ToDma Timer CT32B0 Match 0 DMA request.
kINPUTMUX_Ctimer0M1ToDma Timer CT32B0 Match 1 DMA request.
kINPUTMUX_Ctimer1M0ToDma Timer CT32B1 Match 0 DMA request.
kINPUTMUX_Ctimer1M1ToDma Timer CT32B1 Match 1 DMA request.
kINPUTMUX_PinInt0ToDma Pin interrupt 0.
kINPUTMUX_PinInt1ToDma Pin interrupt 1.
kINPUTMUX_PinInt2ToDma Pin interrupt 2.
kINPUTMUX_PinInt3ToDma Pin interrupt 3.
kINPUTMUX_AesRxToDma AES RX.
kINPUTMUX_AesTxToDma AES TX.
kINPUTMUX_HashRxToDma Hash RX.
kINPUTMUX_HashTxToDma Hash TX.
kINPUTMUX_Otrig0ToDma DMA output trigger 0.
kINPUTMUX_Otrig1ToDma DMA output trigger 1.
kINPUTMUX_Otrig2ToDma DMA output trigger 2.
kINPUTMUX_Otrig3ToDma DMA output trigger 3. DMA OTRIG.
kINPUTMUX_DmaUsart0RxTrigoutToTriginChannels USART 0 RX.
kINPUTMUX_DmaUsart0TxTrigoutToTriginChannels USART 0 TX.
kINPUTMUX_DmaUsart1RxTrigoutToTriginChannels USART 1 RX.
kINPUTMUX_DmaUsart1TxTrigoutToTriginChannels USART 1 TX.
kINPUTMUX_DmaI2c0SlvaeTrigoutToTriginChannels I2C 0 Slave.
kINPUTMUX_DmaI2c0MasterTrigoutToTriginChannels I2C 0 Master.
kINPUTMUX_DmaI2c1SlvaeTrigoutToTriginChannels I2C 1 Slave.
kINPUTMUX_DmaI2c1MasterTrigoutToTriginChannels I2C 1 Master.
kINPUTMUX_DmaSpi0RxTrigoutToTriginChannels SPI 0 RX.
kINPUTMUX_DmaSpi0TxTrigoutToTriginChannels SPI 0 TX.
kINPUTMUX_DmaSpi1RxTrigoutToTriginChannels SPI 1 RX.
kINPUTMUX_DmaSpi1TxTrigoutToTriginChannels SPI 1 TX.
kINPUTMUX_DmaSpifi0TrigoutToTriginChannels SPIFI.
kINPUTMUX_DmaI2c2SlaveTrigoutToTriginChannels I2C 2 Slave.
kINPUTMUX_DmaI2c2MasterTrigoutToTriginChannels I2C 2 Master.
kINPUTMUX_DmaDmic0Ch0TrigoutToTriginChannels DMIC Channel 0.

kINPUTMUX_DmaDmic0Ch1TrigoutToTriginChannels DMIC Channel 1.
kINPUTMUX_DmaHash0RxTrigoutToTriginChannels Hash RX.
kINPUTMUX_DmaHash0TxTrigoutToTriginChannels Hash TX.

21.6 Function Documentation

21.6.1 void INPUTMUX_Init (INPUTMUX_Type ∗ *base*)

This function enables the INPUTMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.

Return values

None.	

21.6.2 void INPUTMUX_AttachSignal (INPUTMUX_Type ∗ *base,* uint32_t *index,* inputmux_connection_t *connection*)

This function gates the INPUTPMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.
index	Destination peripheral to attach the signal to.
connection	Selects connection.

Return values

None.	

21.6.3 void INPUTMUX_Deinit (INPUTMUX_Type ∗ *base*)

This function disables the INPUTMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.

Return values

None.	

Chapter 22
IOCON: I/O pin configuration

22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the I/O pin configuration (IOCON) module of MCUXpresso SDK devices.

22.2 Function groups

22.2.1 Pin mux set

The function IOCONPinMuxSet() sets a pinmux for a single pin according to the selected configuration.

22.2.2 Pin mux set

The function IOCON_SetPinMuxing() sets a pinmux for group of pins according to the selected configuration.

22.3 Typical use case

Example use of IOCON API to selection of GPIO mode.

```
int main(void)
{
    /* enable clock for IOCON */
    CLOCK_EnableClock(kCLOCK_Iocon);

    /* Set pin mux for single pin */
    IOCON_PinMuxSet(IOCON, 0, 29, IOCON_FUNC0 |
      IOCON_GPIO_MODE | IOCON_DIGITAL_EN |
      IOCON_INPFILT_OFF);

    /* Set pin mux for group of pins */
    const iocon_group_t gpio_pins[] = {
    {0, 24, (IOCON_FUNC0 | IOCON_GPIO_MODE |
      IOCON_DIGITAL_EN | IOCON_INPFILT_OFF)},
    {0, 31, (IOCON_FUNC0 | IOCON_GPIO_MODE |
      IOCON_DIGITAL_EN | IOCON_INPFILT_OFF)},
    };

    Chip_IOCON_SetPinMuxing(IOCON, gpio_pins, sizeof(gpio_pins)/sizeof(gpio_pins[0]));

}
```

Files

- file fsl_iocon.h

Data Structures

- struct iocon_group_t
  Array of IOCON pin definitions passed to IOCON_SetPinMuxing() must be in this format. More...

Macros

- #define IOCON_FUNC0 IOCON_PIO_FUNC(0)
  IOCON function and mode selection definitions.
- #define IOCON_FUNC1 IOCON_PIO_FUNC(1)
  Selects pin function 1.
- #define IOCON_FUNC2 IOCON_PIO_FUNC(2)
  Selects pin function 2.
- #define IOCON_FUNC3 IOCON_PIO_FUNC(3)
  Selects pin function 3.
- #define IOCON_FUNC4 IOCON_PIO_FUNC(4)
  Selects pin function 4.
- #define IOCON_FUNC5 IOCON_PIO_FUNC(5)
  Selects pin function 5.
- #define IOCON_FUNC6 IOCON_PIO_FUNC(6)
  Selects pin function 6.
- #define IOCON_FUNC7 IOCON_PIO_FUNC(7)
  Selects pin function 7.
- #define IOCON_MODE_PULLUP IOCON_PIO_MODE(0)
  Selects pull-up function.
- #define IOCON_MODE_REPEATER IOCON_PIO_MODE(1)
  Selects pin repeater function.
- #define IOCON_MODE_INACT IOCON_PIO_MODE(2)
  No addition pin function.
- #define IOCON_MODE_PULLDOWN IOCON_PIO_MODE(3)
  Selects pull-down function.
- #define IOCON_HYS_EN (0x1 << 5)
  Enables hysteresis ??
- #define IOCON_GPIO_MODE IOCON_PIO_SLEW0(1)
  GPIO Mode.
- #define IOCON_I2C_SLEW IOCON_PIO_SLEW0(1)
  I2C Slew Rate Control.
- #define IOCON_INV_EN IOCON_PIO_INVERT(1)
  Enables invert function on input.
- #define IOCON_ANALOG_EN IOCON_PIO_DIGIMODE(0)
  Enables analog function by setting 0 to bit 7.
- #define IOCON_DIGITAL_EN IOCON_PIO_DIGIMODE(1)
  Enables digital function by setting 1 to bit 7(default)
- #define IOCON_STDI2C_EN IOCON_PIO_FILTEROFF(1)
  I2C standard mode/fast-mode.
- #define IOCON_INPFILT_OFF IOCON_PIO_FILTEROFF(1)
  Input filter Off for GPIO pins.
- #define IOCON_INPFILT_ON IOCON_PIO_FILTEROFF(0)
  Input filter On for GPIO pins.
- #define IOCON_SLEW1_OFF IOCON_PIO_SLEW1(0)
  Driver Slew Rate Control.
- #define IOCON_SLEW1_ON IOCON_PIO_SLEW1(1)

Driver Slew Rate Control.
- #define IOCON_FASTI2C_EN (IOCON_INPFILT_ON | IOCON_SLEW1_ON)
  I2C Fast-mode Plus and high-speed slave.
- #define IOCON_OPENDRAIN_EN IOCON_PIO_OD(1)
  Enables open-drain function.
- #define IOCON_S_MODE_0CLK IOCON_PIO_SSEL(0)
  Bypass input filter.
- #define IOCON_S_MODE_1CLK IOCON_PIO_SSEL(1)
  Input pulses shorter than 1 filter clock are rejected.
- #define IOCON_S_MODE_2CLK IOCON_PIO_SSEL(2)
  Input pulses shorter than 2 filter clock2 are rejected.
- #define IOCON_S_MODE_3CLK IOCON_PIO_SSEL(3)
  Input pulses shorter than 3 filter clock2 are rejected.

Functions

- __STATIC_INLINE void IOCON_PinMuxSet (IOCON_Type ∗base, uint8_t port, uint8_t pin, uint32_t modefunc)
  Sets I/O Control pin mux.
- __STATIC_INLINE void IOCON_SetPinMuxing (IOCON_Type ∗base, const iocon_group_t ∗pinArray, uint32_t arrayLength)
  Set all I/O Control pin muxing.
- __STATIC_INLINE void IOCON_PullSet (IOCON_Type ∗base, uint8_t port, uint8_t pin, uint8_t pull_select)
  Sets I/O Control pin mux pull select.
- __STATIC_INLINE void IOCON_FuncSet (IOCON_Type ∗base, uint8_t port, uint8_t pin, uint8_t func)
  Sets I/O Control pin mux pull select.

Driver version

- #define LPC_IOCON_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  IOCON driver version 2.0.0.

22.4 Data Structure Documentation

22.4.1 struct iocon_group_t

22.5 Macro Definition Documentation

22.5.1 #define LPC_IOCON_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

22.5.2 #define IOCON_FUNC0 IOCON_PIO_FUNC(0)

Note

See the User Manual for specific modes and functions supported by the various pins.Selects pin function 0

22.6 Function Documentation

22.6.1 __STATIC_INLINE void IOCON_PinMuxSet (IOCON_Type * *base,* uint8_t *port,* uint8_t *pin,* uint32_t *modefunc*)

Parameters

base	: The base of IOCON peripheral on the chip
port	: GPIO port to mux
pin	: GPIO pin to mux
modefunc	: OR'ed values of type IOCON_∗

Returns

Nothing

22.6.2 __STATIC_INLINE void IOCON_SetPinMuxing (IOCON_Type ∗ *base,* const iocon_group_t ∗ *pinArray,* uint32_t *arrayLength*)

Parameters

base	: The base of IOCON peripheral on the chip
pinArray	: Pointer to array of pin mux selections
arrayLength	: Number of entries in pinArray

Returns

Nothing

22.6.3 __STATIC_INLINE void IOCON_PullSet (IOCON_Type ∗ *base,* uint8_t *port,* uint8_t *pin,* uint8_t *pull_select*)

Parameters

base	: The base of IOCON peripheral on the chip
port	: GPIO port to mux
pin	: GPIO pin to mux

pull_select	: OR'ed values of type IOCON_∗

Returns

Nothing

22.6.4 __STATIC_INLINE void IOCON_FuncSet (IOCON_Type ∗ *base*, uint8_t *port*, uint8_t *pin*, uint8_t *func*)

Parameters

base	: The base of IOCON peripheral on the chip
port	: GPIO port to mux
pin	: GPIO pin to mux
func	: Pinmux function

Returns

Nothing

Chapter 23
IRM: Infra-Red Modulator driver

23.1 Overview

The MCUXpresso SDK provides a Infra-Red Modulator driver for MCUXpresso SDK devices.

Files

- file fsl_cic_irb.h
- file fsl_cic_irb_private.h

Data Structures

- struct cic_irb_config_t
- struct cic_irb_instance_data_t

Enumerations

- enum cic_irb_protocols_t
- enum cic_irb_status_t
- enum cic_irb_carrier_frequency_t
- enum cic_irb_sirc_version_t
- enum cic_irb_rcmm_mode_t
- enum cic_irb_rcmm_signal_free_time_t

CIC_IRB Get Default Configuration

- cic_irb_status_t CIC_IRB_GetDefaultConfig (cic_irb_config_t ∗config)
  CIC_IRB_GetDefaultConfig.

CIC_IRB Initialization

- cic_irb_status_t CIC_IRB_Init (CIC_IRB_Type ∗base, cic_irb_config_t ∗config)
  CIC_IRB_Init.

CIC_IRB Enable

- cic_irb_status_t CIC_IRB_Enable (CIC_IRB_Type ∗base)
  CIC_IRB_Enable.

CIC_IRB Send RC5 Packet

- cic_irb_status_t CIC_IRB_SendRC5Packet (CIC_IRB_Type ∗base, bool toggle, uint8_t address, uint8_t command)
  CIC_IRB_SendRC5Packet.

CIC_IRB Send RC6 Packet

- cic_irb_status_t CIC_IRB_SendRC6Packet (CIC_IRB_Type ∗base, bool toggle, uint8_t field, uint8_t address, uint8_t command)
    CIC_IRB_SendRC6Packet.

CIC_IRB Send SIRC Packet

- cic_irb_status_t CIC_IRB_SendSIRCPacket (CIC_IRB_Type ∗base, cic_irb_sirc_version_t version, uint8_t command, uint8_t address, uint8_t extendedBits)
    CIC_IRB_SendSIRCPacket.

CIC_IRB Send RCMM Packet

- cic_irb_status_t CIC_IRB_SendRCMMPacket (CIC_IRB_Type ∗base, cic_irb_rcmm_mode_t mode, uint8_t modeBits, uint8_t address, uint8_t customerId, uint32_t data, cic_irb_rcmm_signal_free_time_t signalFreeTime)
    CIC_IRB_SendRCMMPacket.

CIC_IRB Is Busy

- cic_irb_status_t CIC_IRB_IsBusy (CIC_IRB_Type ∗base, bool ∗isBusy)
    CIC_IRB_IsBusy.

CIC_IRB Disable

- cic_irb_status_t CIC_IRB_Disable (CIC_IRB_Type ∗base)
    CIC_IRB_Disable.

CIC_IRB Deinitializations

- cic_irb_status_t CIC_IRB_DeInit (CIC_IRB_Type ∗base)
    CIC_IRB_DeInit.

CIC_IRB Get Instance Data

- cic_irb_instance_data_t ∗ CIC_IRB_GetInstanceData (CIC_IRB_Type ∗base)
    CIC_IRB_GetInstanceData.

CIC_IRB RC5 Initialise

- void CIC_IRB_RC5Initialise (CIC_IRB_Type ∗base)
    CIC_IRB_RC5Initialise.

CIC_IRB RC6 Initialise

- void CIC_IRB_RC6Initialise (CIC_IRB_Type ∗base)
    CIC_IRB_RC6Initialise.

CIC_IRB SIRC Initialise

- void CIC_IRB_SIRCInitialise (CIC_IRB_Type *base)
  CIC_IRB_SIRCInitialise.

CIC_IRB RCMM Initialise

- void CIC_IRB_RCMMInitialise (CIC_IRB_Type *base)
  CIC_IRB_RCMMInitialise.

CIC_IRB RC5 Get Instance

- uint8_t CIC_IRB_GetInstance (CIC_IRB_Type *base)
  CIC_IRB_GetInstance.

CIC_IRB Load And Send Fifo

- void CIC_IRB_LoadAndSendFifo (CIC_IRB_Type *base)
  CIC_IRB_LoadAndSendFifo.

CIC_IRB RCx Append Envelopes

- void CIC_IRB_RCxAppendEnvelopes (uint32_t bitPattern, uint8_t bitCount, cic_irb_instance_data_t *instanceData, uint8_t *previousEnvelopeLevel, int8_t *envelope, bool ManchesterEncodingIEEE802_3)
  CIC_IRB_RCxAppendEnvelopes.

23.2 Data Structure Documentation

23.2.1 struct cic_irb_config_t

CIC IRB configuration structure

This structure holds the configuration settings for the CIC IRB peripheral. To initialize this structure to reasonable defaults, call the CIC_IRB_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

23.2.2 struct cic_irb_instance_data_t

CIC IRB instance data structure

23.3 Enumeration Type Documentation

23.3.1 enum cic_irb_protocols_t

IR protocol types

23.3.2 enum cic_irb_status_t

Status code responses from API calls

23.3.3 enum cic_irb_carrier_frequency_t

Status code responses from API calls

23.3.4 enum cic_irb_sirc_version_t

Sony SIRC version types

23.3.5 enum cic_irb_rcmm_mode_t

RC-MM mode types

23.3.6 enum cic_irb_rcmm_signal_free_time_t

RC-MM signal free types

23.4 Function Documentation

23.4.1 cic_irb_status_t CIC_IRB_GetDefaultConfig (cic_irb_config_t * *config*)

This function get default configuration for IRB

Parameters

config	pointer to a configuration structure

Returns

A cic_irb_status_t status code*

23.4.2 cic_irb_status_t CIC_IRB_Init (CIC_IRB_Type * *base*, cic_irb_config_t * *config*)

Initialize the IRB peripheral. Attaches, configures and enables source and peripheral clocks, resets peripheral and initializes instance data. The peripheral is not enabled after this.

Parameters

base	CIC_IRB peripheral base address
config	pointer to a configuration structure

Returns

A cic_irb_status_t status code

23.4.3 cic_irb_status_t CIC_IRB_Enable (CIC_IRB_Type * *base*)

Enable the IRB peripheral doing protocol specific initializations. The interrupts are enabled after this call.

Parameters

base	CIC_IRB peripheral base address

Returns

A cic_irb_status_t status code

23.4.4 cic_irb_status_t CIC_IRB_SendRC5Packet (CIC_IRB_Type * *base*, bool *toggle*, uint8_t *address*, uint8_t *command*)

Send a RC-5 packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

Parameters

base	CIC_IRB peripheral base address
toggle	the state of the toggle bit to encode
address	the 5 bit address to go into the message
command	the 7 bit command to go into the message

Returns

A cic_irb_status_t status code

23.4.5 cic_irb_status_t CIC_IRB_SendRC6Packet (CIC_IRB_Type * *base*, bool *toggle*, uint8_t *field*, uint8_t *address*, uint8_t *command*)

Send a RC-5 packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

Parameters

base	CIC_IRB peripheral base address
toggle	the state of the toggle bit to encode
field	the 3 bit field to go into the message
address	the 8 bit address to go into the message
command	the 8 bit command to go into the message

Returns

A cic_irb_status_t status code

23.4.6 cic_irb_status_t CIC_IRB_SendSIRCPacket (CIC_IRB_Type ∗ *base,* cic_irb_sirc_version_t *version,* uint8_t *command,* uint8_t *address,* uint8_t *extendedBits*)

Send a SIRC packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

Parameters

base	CIC_IRB peripheral base address
version	the version of the protocol to use, varies the packet length command the 7 bit command to go into the message
address	the 5 or 8 bit address to go into the message
extendedBits	8 bits of extra data in 20 bit message

Returns

A cic_irb_status_t status code

23.4.7 cic_irb_status_t CIC_IRB_SendRCMMPacket (CIC_IRB_Type ∗ *base,* cic_irb_rcmm_mode_t *mode,* uint8_t *modeBits,* uint8_t *address,* uint8_t *customerId,* uint32_t *data,* cic_irb_rcmm_signal_free_time_t *signalFreeTime*)

Send a RCMM packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

Parameters

base	CIC_IRB peripheral base address The RC-MM message mode. See RC-MM documentation Bits The mode numerical value, varying number of bits
address	RC-MM address data field in some modes
customerId	RC-MM customer if in some modes
data	the RC-MM data field, varying length depending on mode
signalFreeTime	Silent period after packet transmission

Returns

A cic_irb_status_t status code

23.4.8 cic_irb_status_t CIC_IRB_IsBusy (CIC_IRB_Type * *base,* bool * *isBusy*)

Determine if the IRB peripheral instance is in the process of sending the previous message.

Parameters

base	CIC_IRB peripheral base address
isBusy	pointer to bool for result

Returns

A cic_irb_status_t status code

23.4.9 cic_irb_status_t CIC_IRB_Disable (CIC_IRB_Type * *base*)

Disable the IRB peripheral such that it can be enabled again without doing another init. This stops interrupts and any part sent message is abandoned.

Parameters

base	CIC_IRB peripheral base address

Returns

A cic_irb_status_t status code

23.4.10 cic_irb_status_t CIC_IRB_DeInit (CIC_IRB_Type * *base*)

De-initialises the IRB peripheral instance. The clock is stopped.

Parameters

base	CIC_IRB peripheral base address

Returns

A cic_irb_status_t status code

23.4.11 cic_irb_instance_data_t* CIC_IRB_GetInstanceData (CIC_IRB_Type * *base*)

Get a pointer to the data structure containing context information for a peripheral instance

Parameters

base	CIC_IRB peripheral base address.

Returns

Pointer to the instance data structure

NOTES: Returns NULL if an instance not found for base

23.4.12 void CIC_IRB_RC5Initialise (CIC_IRB_Type * *base*)

Initialise the peripheral ready to send RC5 messages.

Parameters

base	CIC_IRB peripheral base address.

23.4.13 void CIC_IRB_RC6Initialise (CIC_IRB_Type * *base*)

Initialise the peripheral ready to send RC6 messages.

Parameters

base	CIC_IRB peripheral base address.

23.4.14 void CIC_IRB_SIRCInitialise (CIC_IRB_Type * *base*)

Initialise the peripheral ready to send SIRC messages.

Parameters

base	CIC_IRB peripheral base address.

23.4.15 void CIC_IRB_RCMMInitialise (CIC_IRB_Type * *base*)

Initialise the peripheral ready to send RCMM messages.

Parameters

base	CIC_IRB peripheral base address.

23.4.16 uint8_t CIC_IRB_GetInstance (CIC_IRB_Type * *base*)

Get the CIC_IRB instance from peripheral base address.

Parameters

base	CIC_IRB peripheral base address.

Returns

CIC_IRB instance number.

NOTES: Returns FSL_FEATURE_SOC_CIC_IRB_COUNT if instance for base not found.

23.4.17 void CIC_IRB_LoadAndSendFifo (CIC_IRB_Type * *base*)

Load the FIFO with the first part of a series of envelopes to send. If a message's complete set of envelopes fits within the FIFO size then the whole message will be sent. If not then the maximum number of envelopes is loaded into the FIFO that is possible. Subsequent FIFO loads for the remaining envelopes are loaded into the FIFO in the interrupt handler which is triggered when the FIFO becomes empty.

Parameters

base	CIC_IRB peripheral base address.

23.4.18 void CIC_IRB_RCxAppendEnvelopes (uint32_t *bitPattern,* uint8_t *bitCount,* cic_irb_instance_data_t ∗ *instanceData,* uint8_t ∗ *previousEnvelopeLevel,* int8_t ∗ *envelope,* bool *ManchesterEncodingIEEE802_3*)

Take a bit pattern that comprises part of a RCx message and encode this in envelopes using the appropriate type of Manchester encoding. The envelopes can be appended to an array of previously encoded envelopes or can be a complete message.

Parameters

bitPattern	the binary data to encode into the message
bitCount	the number of bits in bitPattern
instanceData	pointer to the peripheral instance data structure
previousEnvelopeLevel	the level of the last envelope of the preceding part of the encoded message
envelope	pointer to array to add this bit pattern's encoded envelopes
ManchesterEncodingIEEE802_3	type of Manchester encoding, true for IEEE 802.3, false for Thomas

Chapter 24
PINT: Pin Interrupt and Pattern Match Driver

24.1 Overview

The MCUXpresso SDK provides a driver for the Pin Interrupt and Pattern match (PINT).

It can configure one or more pins to generate a pin interrupt when the pin or pattern match conditions are met. The pins do not have to be configured as gpio pins however they must be connected to PINT via INPUTMUX. Only the pin interrupt or pattern match function can be active for interrupt generation. If the pin interrupt function is enabled then the pattern match function can be used for wakeup via RXEV.

24.2 Pin Interrupt and Pattern match Driver operation

PINT_PinInterruptConfig() function configures the pins for pin interrupt.

PINT_PatternMatchConfig() function configures the pins for pattern match.

24.2.1 Pin Interrupt use case

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

24.2.2 Pattern match use case

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

Files

- file fsl_pint.h

Typedefs

- typedef void(∗ pint_cb_t)(pint_pin_int_t pintr, uint32_t pmatch_status)
  PINT Callback function.

Enumerations

- enum pint_pin_enable_t {
  kPINT_PinIntEnableNone = 0U,
  kPINT_PinIntEnableRiseEdge = PINT_PIN_RISE_EDGE,
  kPINT_PinIntEnableFallEdge = PINT_PIN_FALL_EDGE,
  kPINT_PinIntEnableBothEdges = PINT_PIN_BOTH_EDGE,
  kPINT_PinIntEnableLowLevel = PINT_PIN_LOW_LEVEL,
  kPINT_PinIntEnableHighLevel = PINT_PIN_HIGH_LEVEL }

PINT Pin Interrupt enable type.

- enum pint_pin_int_t {
  kPINT_PinInt0 = 0U,
  kPINT_PinInt1 = 1U,
  kPINT_PinInt2 = 2U,
  kPINT_PinInt3 = 3U }

  PINT Pin Interrupt type.
- enum pint_pmatch_input_src_t {
  kPINT_PatternMatchInp0Src = 0U,
  kPINT_PatternMatchInp1Src = 1U,
  kPINT_PatternMatchInp2Src = 2U,
  kPINT_PatternMatchInp3Src = 3U,
  kPINT_PatternMatchInp4Src = 4U,
  kPINT_PatternMatchInp5Src = 5U,
  kPINT_PatternMatchInp6Src = 6U,
  kPINT_PatternMatchInp7Src = 7U }

  PINT Pattern Match bit slice input source type.
- enum pint_pmatch_bslice_t {
  kPINT_PatternMatchBSlice0 = 0U,
  kPINT_PatternMatchBSlice1 = 1U,
  kPINT_PatternMatchBSlice2 = 2U,
  kPINT_PatternMatchBSlice3 = 3U }

  PINT Pattern Match bit slice type.
- enum pint_pmatch_bslice_cfg_t {
  kPINT_PatternMatchAlways = 0U,
  kPINT_PatternMatchStickyRise = 1U,
  kPINT_PatternMatchStickyFall = 2U,
  kPINT_PatternMatchStickyBothEdges = 3U,
  kPINT_PatternMatchHigh = 4U,
  kPINT_PatternMatchLow = 5U,
  kPINT_PatternMatchNever = 6U,
  kPINT_PatternMatchBothEdges = 7U }

  PINT Pattern Match configuration type.

Functions

- void PINT_Init (PINT_Type ∗base)

  Initialize PINT peripheral.
- void PINT_PinInterruptConfig (PINT_Type ∗base, pint_pin_int_t intr, pint_pin_enable_t enable, pint_cb_t callback)

  Configure PINT peripheral pin interrupt.
- void PINT_PinInterruptGetConfig (PINT_Type ∗base, pint_pin_int_t pintr, pint_pin_enable_t ∗enable, pint_cb_t ∗callback)

  Get PINT peripheral pin interrupt configuration.
- void PINT_PinInterruptClrStatus (PINT_Type ∗base, pint_pin_int_t pintr)

  Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.
- static uint32_t PINT_PinInterruptGetStatus (PINT_Type ∗base, pint_pin_int_t pintr)

Get Selected pin interrupt status.
- void PINT_PinInterruptClrStatusAll (PINT_Type *base)
  Clear all pin interrupts status only when pins were triggered by edge-sensitive.
- static uint32_t PINT_PinInterruptGetStatusAll (PINT_Type *base)
  Get all pin interrupts status.
- static void PINT_PinInterruptClrFallFlag (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt fall flag.
- static uint32_t PINT_PinInterruptGetFallFlag (PINT_Type *base, pint_pin_int_t pintr)
  Get selected pin interrupt fall flag.
- static void PINT_PinInterruptClrFallFlagAll (PINT_Type *base)
  Clear all pin interrupt fall flags.
- static uint32_t PINT_PinInterruptGetFallFlagAll (PINT_Type *base)
  Get all pin interrupt fall flags.
- static void PINT_PinInterruptClrRiseFlag (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt rise flag.
- static uint32_t PINT_PinInterruptGetRiseFlag (PINT_Type *base, pint_pin_int_t pintr)
  Get selected pin interrupt rise flag.
- static void PINT_PinInterruptClrRiseFlagAll (PINT_Type *base)
  Clear all pin interrupt rise flags.
- static uint32_t PINT_PinInterruptGetRiseFlagAll (PINT_Type *base)
  Get all pin interrupt rise flags.
- void PINT_PatternMatchConfig (PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
  Configure PINT pattern match.
- void PINT_PatternMatchGetConfig (PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
  Get PINT pattern match configuration.
- static uint32_t PINT_PatternMatchGetStatus (PINT_Type *base, pint_pmatch_bslice_t bslice)
  Get pattern match bit slice status.
- static uint32_t PINT_PatternMatchGetStatusAll (PINT_Type *base)
  Get status of all pattern match bit slices.
- uint32_t PINT_PatternMatchResetDetectLogic (PINT_Type *base)
  Reset pattern match detection logic.
- static void PINT_PatternMatchEnable (PINT_Type *base)
  Enable pattern match function.
- static void PINT_PatternMatchDisable (PINT_Type *base)
  Disable pattern match function.
- static void PINT_PatternMatchEnableRXEV (PINT_Type *base)
  Enable RXEV output.
- static void PINT_PatternMatchDisableRXEV (PINT_Type *base)
  Disable RXEV output.
- void PINT_EnableCallback (PINT_Type *base)
  Enable callback.
- void PINT_DisableCallback (PINT_Type *base)
  Disable callback.
- void PINT_Deinit (PINT_Type *base)
  Deinitialize PINT peripheral.
- void PINT_EnableCallbackByIndex (PINT_Type *base, pint_pin_int_t pintIdx)
  enable callback by pin index.
- void PINT_DisableCallbackByIndex (PINT_Type *base, pint_pin_int_t pintIdx)
  disable callback by pin index.

Driver version

- #define FSL_PINT_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  Version 2.1.3.

24.3 Typedef Documentation

24.3.1 typedef void(∗ pint_cb_t)(pint_pin_int_t pintr, uint32_t pmatch_status)

24.4 Enumeration Type Documentation

24.4.1 enum pint_pin_enable_t

Enumerator

kPINT_PinIntEnableNone Do not generate Pin Interrupt.
kPINT_PinIntEnableRiseEdge Generate Pin Interrupt on rising edge.
kPINT_PinIntEnableFallEdge Generate Pin Interrupt on falling edge.
kPINT_PinIntEnableBothEdges Generate Pin Interrupt on both edges.
kPINT_PinIntEnableLowLevel Generate Pin Interrupt on low level.
kPINT_PinIntEnableHighLevel Generate Pin Interrupt on high level.

24.4.2 enum pint_pin_int_t

Enumerator

kPINT_PinInt0 Pin Interrupt 0.
kPINT_PinInt1 Pin Interrupt 1.
kPINT_PinInt2 Pin Interrupt 2.
kPINT_PinInt3 Pin Interrupt 3.

24.4.3 enum pint_pmatch_input_src_t

Enumerator

kPINT_PatternMatchInp0Src Input source 0.
kPINT_PatternMatchInp1Src Input source 1.
kPINT_PatternMatchInp2Src Input source 2.
kPINT_PatternMatchInp3Src Input source 3.
kPINT_PatternMatchInp4Src Input source 4.
kPINT_PatternMatchInp5Src Input source 5.
kPINT_PatternMatchInp6Src Input source 6.
kPINT_PatternMatchInp7Src Input source 7.

24.4.4 enum pint_pmatch_bslice_t

Enumerator

kPINT_PatternMatchBSlice0 Bit slice 0.
kPINT_PatternMatchBSlice1 Bit slice 1.
kPINT_PatternMatchBSlice2 Bit slice 2.
kPINT_PatternMatchBSlice3 Bit slice 3.

24.4.5 enum pint_pmatch_bslice_cfg_t

Enumerator

kPINT_PatternMatchAlways Always Contributes to product term match.
kPINT_PatternMatchStickyRise Sticky Rising edge.
kPINT_PatternMatchStickyFall Sticky Falling edge.
kPINT_PatternMatchStickyBothEdges Sticky Rising or Falling edge.
kPINT_PatternMatchHigh High level.
kPINT_PatternMatchLow Low level.
kPINT_PatternMatchNever Never contributes to product term match.
kPINT_PatternMatchBothEdges Either rising or falling edge.

24.5 Function Documentation

24.5.1 void PINT_Init (PINT_Type * *base*)

This function initializes the PINT peripheral and enables the clock.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.2 void PINT_PinInterruptConfig (PINT_Type * *base,* pint_pin_int_t *intr,* pint_pin_enable_t *enable,* pint_cb_t *callback*)

This function configures a given pin interrupt.

Parameters

base	Base address of the PINT peripheral.
intr	Pin interrupt.
enable	Selects detection logic.
callback	Callback.

Return values

None.	

24.5.3 void PINT_PinInterruptGetConfig (PINT_Type * *base*, pint_pin_int_t *pintr*, pint_pin_enable_t * *enable*, pint_cb_t * *callback*)

This function returns the configuration of a given pin interrupt.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.
enable	Pointer to store the detection logic.
callback	Callback.

Return values

None.	

24.5.4 void PINT_PinInterruptClrStatus (PINT_Type * *base*, pint_pin_int_t *pintr*)

This function clears the selected pin interrupt status.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

24.5.5 static uint32_t PINT_PinInterruptGetStatus (PINT_Type * *base,* pint_pin_int_t *pintr*) [inline], [static]

This function returns the selected pin interrupt status.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

status	= 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

24.5.6 void PINT_PinInterruptClrStatusAll (PINT_Type * *base*)

This function clears the status of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.7 static uint32_t PINT_PinInterruptGetStatusAll (PINT_Type * *base*) [inline], [static]

This function returns the status of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

status	Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active.

24.5.8 static void PINT_PinInterruptClrFallFlag (PINT_Type * *base*, pint_pin_int_t *pintr*) [inline], [static]

This function clears the selected pin interrupt fall flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

24.5.9 static uint32_t PINT_PinInterruptGetFallFlag (PINT_Type * *base*, pint_pin_int_t *pintr*) [inline], [static]

This function returns the selected pin interrupt fall flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

flag	= 0 Falling edge has not been detected. = 1 Falling edge has been detected.

24.5.10 static void PINT_PinInterruptClrFallFlagAll (PINT_Type * *base*) [inline], [static]

This function clears the fall flag for all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.11 static uint32_t PINT_PinInterruptGetFallFlagAll (PINT_Type * *base*) [inline], [static]

This function returns the fall flag of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

flags	Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

24.5.12 static void PINT_PinInterruptClrRiseFlag (PINT_Type * *base*, pint_pin_int_t *pintr*) [inline], [static]

This function clears the selected pin interrupt rise flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

24.5.13 static uint32_t PINT_PinInterruptGetRiseFlag (PINT_Type * base, pint_pin_int_t pintr) [inline], [static]

This function returns the selected pin interrupt rise flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

flag	= 0 Rising edge has not been detected. = 1 Rising edge has been detected.

24.5.14 static void PINT_PinInterruptClrRiseFlagAll (PINT_Type ∗ *base*) [inline], [static]

This function clears the rise flag for all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.15 static uint32_t PINT_PinInterruptGetRiseFlagAll (PINT_Type ∗ *base*) [inline], [static]

This function returns the rise flag of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

flags	Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

24.5.16 void PINT_PatternMatchConfig (PINT_Type * *base*, pint_pmatch_bslice_t *bslice*, pint_pmatch_cfg_t * *cfg*)

This function configures a given pattern match bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.
cfg	Pointer to bit slice configuration.

Return values

None.	

24.5.17 void PINT_PatternMatchGetConfig (PINT_Type ∗ *base*, pint_pmatch_bslice_t *bslice*, pint_pmatch_cfg_t ∗ *cfg*)

This function returns the configuration of a given pattern match bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.
cfg	Pointer to bit slice configuration.

Return values

None.	

24.5.18 static uint32_t PINT_PatternMatchGetStatus (PINT_Type ∗ *base*, pint_pmatch_bslice_t *bslice*) [inline], [static]

This function returns the status of selected bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.

Return values

status	= 0 Match has not been detected. = 1 Match has been detected.

24.5.19 static uint32_t PINT_PatternMatchGetStatusAll (PINT_Type ∗ *base*) [inline], [static]

This function returns the status of all bit slices.

Parameters

base	Base address of the PINT peripheral.

Return values

status	Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected.

24.5.20 uint32_t PINT_PatternMatchResetDetectLogic (PINT_Type ∗ *base*)

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

base	Base address of the PINT peripheral.

Return values

pmstatus	Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.

24.5.21 static void PINT_PatternMatchEnable (PINT_Type ∗ *base*) [inline], [static]

This function enables the pattern match function.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.22 static void PINT_PatternMatchDisable (PINT_Type ∗ *base*) [inline], [static]

This function disables the pattern match function.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.23 static void PINT_PatternMatchEnableRXEV (PINT_Type ∗ *base*) [inline], [static]

This function enables the pattern match RXEV output.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.24 static void PINT_PatternMatchDisableRXEV (PINT_Type ∗ *base*) [inline], [static]

This function disables the pattern match RXEV output.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.25 void PINT_EnableCallback (PINT_Type ∗ *base*)

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.26 void PINT_DisableCallback (PINT_Type ∗ *base*)

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

base	Base address of the peripheral.

Return values

None.	

24.5.27 void PINT_Deinit (PINT_Type ∗ *base*)

This function disables the PINT clock.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

24.5.28 void PINT_EnableCallbackByIndex (PINT_Type * *base*, pint_pin_int_t *pintIdx*)

This function enables callback by pin index instead of enabling all pins.

Parameters

base	Base address of the peripheral.
pinIdx	pin index.

Return values

None.	

24.5.29 void PINT_DisableCallbackByIndex (PINT_Type * *base*, pint_pin_int_t *pintIdx*)

This function disables callback by pin index instead of disabling all pins.

Parameters

base	Base address of the peripheral.
pinIdx	pin index.

Return values

None.	

Chapter 25
Power: Power driver

25.1 Overview

The MCUXpresso SDK provides a power driver for the MCUXpresso SDK devices.

25.2 Function description

Power driver and library provides these functions:

- Functions to enable and disable power to different peripherals
- Functions to obtain power down config structure with default parameters
- Functions to determine cause of reset
- Functions to get power Library API to return the library version.

25.2.1 Power enable and disable

Power driver provides two API's POWER_EnablePD() and POWER_DisablePD() to enable or disable the PDRUNCFG bits in SYSCON The PDRUNCFG has an inverted logic i.e. the peripheral is powered on when the bit is cleared and powered off when bit is set. So the API POWER_DisablePD() is used to power on a peripheral and POWER_EnablePD() is used to power off a peripheral. The API's take a parameter of type pd_bit_t which organizes the PDRUNCFG bits. The driver also provides two separate API's to power down and power up Flash, POWER_PowerDownFlash() and POWER_PowerUpFlash()

Files

- file fsl_power.h

Data Structures

- struct pm_bod_cfg_t
  BOD config. More...
- struct pm_power_config_t
  Power config. More...

Macros

- #define POWER_BOD_ENABLE (1 << 0)
  BODVBAT configuration flag.
- #define POWER_BOD_HIGH (1 << 3)
  ES2 BOD VBAT only.
- #define POWER_BOD_LVL_1_75V 9
  BOD trigger level setting.

- #define POWER_BOD_LVL_1_8V 10
  BOD trigger level 1.8V.
- #define POWER_BOD_LVL_1_9V 11
  BOD trigger level 1.9V.
- #define POWER_BOD_LVL_2_0V 12
  BOD trigger level 2.0V.
- #define POWER_BOD_LVL_2_1V 13
  BOD trigger level 2.1V.
- #define POWER_BOD_LVL_2_2V 14
  BOD trigger level 2.2V.
- #define POWER_BOD_LVL_2_3V 15
  BOD trigger level 2.3V.
- #define POWER_BOD_LVL_2_4V 16
  BOD trigger level 2.4V.
- #define POWER_BOD_LVL_2_5V 17
  BOD trigger level 2.5V.
- #define POWER_BOD_LVL_2_6V 18
  BOD trigger level 2.6V.
- #define POWER_BOD_LVL_2_7V 19
  BOD trigger level 2.7V.
- #define POWER_BOD_LVL_2_8V 20
  BOD trigger level 2.8V.
- #define POWER_BOD_LVL_2_9V 21
  BOD trigger level 2.9V.
- #define POWER_BOD_LVL_3_0V 22
  BOD trigger level 3.0V.
- #define POWER_BOD_LVL_3_1V 23
  BOD trigger level 3.1V.
- #define POWER_BOD_LVL_3_2V 24
  BOD trigger level 3.2V.
- #define POWER_BOD_LVL_3_3V 25
  BOD trigger level 3.3V.
- #define POWER_BOD_HYST_25MV 0
  BOD Hysteresis control setting.
- #define POWER_BOD_HYST_50MV 1
  BOD Hysteresis control 50mV.
- #define POWER_BOD_HYST_75MV 2
  BOD Hysteresis control 75mV.
- #define POWER_BOD_HYST_100MV 3
  BOD Hysteresis control 100mV, default at Reset.
- #define PM_CFG_SRAM_BANK_BIT_BASE 0
  SRAM banks definition list for retention in power down modes !
- #define PM_CFG_SRAM_BANK0_RET (1<<0)
  On ES1, this bank shall be kept in retention for Warmstart from power down.
- #define PM_CFG_SRAM_BANK1_RET (1<<1)
  Bank 1 shall be kept in retention.
- #define PM_CFG_SRAM_BANK2_RET (1<<2)
  Bank 2 shall be kept in retention.
- #define PM_CFG_SRAM_BANK3_RET (1<<3)
  Bank 3 shall be kept in retention.
- #define PM_CFG_SRAM_BANK4_RET (1<<4)

Bank 4 shall be kept in retention.
- #define PM_CFG_SRAM_BANK5_RET (1<<5)
  Bank 5 shall be kept in retention.
- #define PM_CFG_SRAM_BANK6_RET (1<<6)
  Bank 6 shall be kept in retention.
- #define PM_CFG_SRAM_BANK7_RET (1<<7)
  On ES2, this bank shall be kept in retention for Warmstart.
- #define PM_CFG_SRAM_BANK8_RET (1<<8)
  Bank 8 shall be kept in retention.
- #define PM_CFG_SRAM_BANK9_RET (1<<9)
  Bank 9 shall be kept in retention.
- #define PM_CFG_SRAM_BANK10_RET (1<<10)
  Bank 10 shall be kept in retention.
- #define PM_CFG_SRAM_BANK11_RET (1<<11)
  Bank 11 shall be kept in retention.
- #define PM_CFG_SRAM_ALL_RETENTION 0xFFF
  All banks shall be kept in retention.
- #define PM_CFG_KEEP_AO_VOLTAGE (1<<15)
  keep the same voltage on the Always-on power domain - typical used with FRO32K to avoid timebase drift
- #define POWER_WAKEUPSRC_SYSTEM LOWPOWER_WAKEUPSRCINT0_SYSTEM_IRQ
  BOD, Watchdog Timer, Flash controller, [DEEP SLEEP] BODVBAT [POWER_DOWN].
- #define POWER_WAKEUPSRC_DMA LOWPOWER_WAKEUPSRCINT0_DMA_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_GINT LOWPOWER_WAKEUPSRCINT0_GINT_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_IRBLASTER LOWPOWER_WAKEUPSRCINT0_IRBLASTER_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PINT0 LOWPOWER_WAKEUPSRCINT0_PINT0_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PINT1 LOWPOWER_WAKEUPSRCINT0_PINT1_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PINT2 LOWPOWER_WAKEUPSRCINT0_PINT2_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PINT3 LOWPOWER_WAKEUPSRCINT0_PINT3_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_SPIFI LOWPOWER_WAKEUPSRCINT0_SPIFI_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_TIMER0 LOWPOWER_WAKEUPSRCINT0_TIMER0_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_TIMER1 LOWPOWER_WAKEUPSRCINT0_TIMER1_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_USART0 LOWPOWER_WAKEUPSRCINT0_USART0_IRQ
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_USART1 LOWPOWER_WAKEUPSRCINT0_USART1_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_I2C0 LOWPOWER_WAKEUPSRCINT0_I2C0_IRQ
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_I2C1 LOWPOWER_WAKEUPSRCINT0_I2C1_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_SPI0 LOWPOWER_WAKEUPSRCINT0_SPI0_IRQ

[DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_SPI1 LOWPOWER_WAKEUPSRCINT0_SPI1_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM0 LOWPOWER_WAKEUPSRCINT0_PWM0_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM1 LOWPOWER_WAKEUPSRCINT0_PWM1_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM2 LOWPOWER_WAKEUPSRCINT0_PWM2_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM3 LOWPOWER_WAKEUPSRCINT0_PWM3_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM4 LOWPOWER_WAKEUPSRCINT0_PWM4_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM5 LOWPOWER_WAKEUPSRCINT0_PWM5_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM6 LOWPOWER_WAKEUPSRCINT0_PWM6_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM7 LOWPOWER_WAKEUPSRCINT0_PWM7_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM8 LOWPOWER_WAKEUPSRCINT0_PWM8_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM9 LOWPOWER_WAKEUPSRCINT0_PWM9_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_PWM10 LOWPOWER_WAKEUPSRCINT0_PWM10_IR
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_I2C2 LOWPOWER_WAKEUPSRCINT0_I2C2_IRQ
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_RTC LOWPOWER_WAKEUPSRCINT0_RTC_IRQ
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_NFCTAG LOWPOWER_WAKEUPSRCINT0_NFCTAG_IRQ
  [DEEP SLEEP, POWER DOWN (ES2 Only), DEEP DOWN (ES2 only)]
- #define POWER_WAKEUPSRC_MAILBOX LOWPOWER_WAKEUPSRCINT0_MAILBOX_IRQ
  Mailbox, Wake-up from DEEP SLEEP and POWER DOWN low power mode [DEEP SLEEP, POWER DOWN].
- #define POWER_WAKEUPSRC_ADC_SEQA ((uint64_t)LOWPOWER_WAKEUPSRCINT1_ADC_SEQA_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_ADC_SEQB ((uint64_t)LOWPOWER_WAKEUPSRCINT1_ADC_SEQB_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_ADC_THCMP_OVR ((uint64_t)LOWPOWER_WAKEUPSRCINT1_ADC_THCMP_OVR_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_DMIC ((uint64_t)LOWPOWER_WAKEUPSRCINT1_DMIC_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_HWVAD ((uint64_t)LOWPOWER_WAKEUPSRCINT1_HWVAD_IRQ << 32)
  [DEEP SLEEP]

- #define POWER_WAKEUPSRC_BLE_DP ((uint64_t)LOWPOWER_WAKEUPSRCINT1_BLE_DP_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_BLE_DP0 ((uint64_t)LOWPOWER_WAKEUPSRCINT1_BLE_DP0_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_BLE_DP1 ((uint64_t)LOWPOWER_WAKEUPSRCINT1_BLE_DP1_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_BLE_DP2 ((uint64_t)LOWPOWER_WAKEUPSRCINT1_BLE_DP2_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_BLE_LL_ALL ((uint64_t)LOWPOWER_WAKEUPSRCINT1_BLE_LL_ALL_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_ZIGBEE_MAC ((uint64_t)LOWPOWER_WAKEUPSRCINT1_ZIGBEE_MAC_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_ZIGBEE_MODEM ((uint64_t)LOWPOWER_WAKEUPSRCINT1_ZIGBEE_MODEM_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_RFP_TMU ((uint64_t)LOWPOWER_WAKEUPSRCINT1_RFP_TMU_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_RFP_AGC ((uint64_t)LOWPOWER_WAKEUPSRCINT1_RFP_AGC_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_ISO7816 ((uint64_t)LOWPOWER_WAKEUPSRCINT1_ISO7816_IRQ << 32)
  [DEEP SLEEP]
- #define POWER_WAKEUPSRC_ANA_COMP ((uint64_t)LOWPOWER_WAKEUPSRCINT1_ANA_COMP_IRQ << 32)
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_WAKE_UP_TIMER0 ((uint64_t)LOWPOWER_WAKEUPSRCINT1_WAKE_UP_TIMER0_IRQ << 32)
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_WAKE_UP_TIMER1 ((uint64_t)LOWPOWER_WAKEUPSRCINT1_WAKE_UP_TIMER1_IRQ << 32)
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_BLE_WAKE_TIMER ((uint64_t)LOWPOWER_WAKEUPSRCINT1_BLE_WAKE_TIMER_IRQ << 32)
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_BLE_OSC_EN ((uint64_t)LOWPOWER_WAKEUPSRCINT1_BLE_OSC_EN_IRQ << 32)
  [DEEP SLEEP, POWER DOWN]
- #define POWER_WAKEUPSRC_IO ((uint64_t)LOWPOWER_WAKEUPSRCINT1_IO_IRQ << 32)
  [POWER DOWN, DEEP DOWN]

Enumerations

- enum pd_bit_t {
  kPDRUNCFG_PD_LDO_ADC_EN = 22,
  kPDRUNCFG_PD_BOD_MEM_EN = 23,
  kPDRUNCFG_PD_BOD_CORE_EN = 24,
  kPDRUNCFG_PD_FRO32K_EN = 25,
  kPDRUNCFG_PD_XTAL32K_EN = 26,
  kPDRUNCFG_PD_BOD_ANA_COMP_EN = 27 }
  PDRUNCFG bits offset.
- enum pm_power_mode_t {
  PM_POWER_DOWN,
  PM_DEEP_DOWN }
  Power modes.
- enum reset_cause_t { ,
  RESET_POR = (1 << 0),
  RESET_EXT_PIN = (1 << 1),
  RESET_BOR = (1 << 2),
  RESET_SYS_REQ = (1 << 3),
  RESET_WDT = (1 << 4),
  RESET_WAKE_DEEP_PD = (1 << 5),
  RESET_WAKE_PD = (1 << 6),
  RESET_SW_REQ = (1 << 7) }
  Reset Cause definition.
- enum pm_ldo_volt_t {
  PM_LDO_VOLT_1_1V_DEFAULT,
  PM_LDO_VOLT_1_0V }
  LDO voltage setting.

Power Configuration

- void POWER_Init (void)
  Initialize the sdk power drivers.
- void POWER_SetTrimDefaultActiveVoltage (void)
  Optimize the LDO voltage for power saving Initialize the power domains.
- void POWER_BodSetUp (void)
  BODMEM and BODCORE setup.
- void POWER_BodActivate (void)
  enable SW reset for the BODCORE
- static void POWER_EnablePD (pd_bit_t en)
  API to enable PDRUNCFG bit in the Syscon.
- static void POWER_DisablePD (pd_bit_t en)
  API to disable PDRUNCFG bit in the Syscon.
- static uint32_t POWER_GetIoWakeStatus (void)
  Get IO and Ntag Field detect Wake-up sources from Power Down and Deep Power Down modes.
- static void POWER_EnterSleep (void)
  Power API to enter sleep mode (Doze mode)
- bool POWER_EnterPowerMode (pm_power_mode_t pm_power_mode, pm_power_config_t *pm-_power_config)

Power Library API to enter different power mode.
- reset_cause_t POWER_GetResetCause (void)
  determine cause of reset
- void POWER_ClearResetCause (void)
  Clear cause of reset.
- uint32_t POWER_GetLibVersion (void)
  Power Library API to return the library version.
- void POWER_BodVbatGetDefaultConfig (pm_bod_cfg_t ∗bod_cfg_p)
  Get default Vbat BOD config parameters, level @1.75V, Hysteresis @ 100mV.
- bool POWER_BodVbatConfig (pm_bod_cfg_t ∗bod_cfg_p)
  Configure the VBAT BOD.
- void POWER_ApplyLdoActiveVoltage (pm_ldo_volt_t ldoVolt)
  Configure the LDO voltage.
- #define POWER_ENTER_SLEEP() __DSB(); __WFI(); __ISB();
  Power API to enter sleep mode (Doze mode)

25.3 Data Structure Documentation

25.3.1 struct pm_bod_cfg_t

Data Fields

- uint8_t bod_level
  BOD trigger level.
- uint8_t bod_hyst
  BOD Hysteresis control.
- uint8_t bod_cfg
  BOD config setting.

25.3.2 struct pm_power_config_t

Data Fields

- pm_wake_source_t pm_wakeup_src
  Wakeup source select.
- uint32_t pm_wakeup_io
  Wakeup IO.
- uint32_t pm_config
  Power mode config.

25.4 Macro Definition Documentation

25.4.1 #define POWER_BOD_LVL_1_75V 9

Default at Reset , 1.7V on ES1

25.4.2 #define POWER_BOD_HYST_25MV 0

BOD Hysteresis control 25mV

25.4.3 #define POWER_ENTER_SLEEP( ) __DSB(); __WFI(); __ISB();

Note

: The static inline function has not the expecetd effect in -O0. If order to force inline this macro is added

Returns

none

25.5 Enumeration Type Documentation

25.5.1 enum pd_bit_t

Enumerator

kPDRUNCFG_PD_LDO_ADC_EN Offset is 22, LDO ADC enabled.
kPDRUNCFG_PD_BOD_MEM_EN Offset is 23, BOD MEM enabled.
kPDRUNCFG_PD_BOD_CORE_EN Offset is 24, BOD CORE enabled.
kPDRUNCFG_PD_FRO32K_EN Offset is 25, FRO32K enabled.
kPDRUNCFG_PD_XTAL32K_EN Offset is 26, XTAL32K enabled.
kPDRUNCFG_PD_BOD_ANA_COMP_EN Offset is 27, Analog Comparator enabled.

25.5.2 enum pm_power_mode_t

Enumerator

PM_POWER_DOWN Power down mode.
PM_DEEP_DOWN Deep power down mode.

25.5.3 enum reset_cause_t

Enumerator

RESET_POR The last chip reset was caused by a Power On Reset.
RESET_EXT_PIN The last chip reset was caused by a Pad Reset.
RESET_BOR The last chip reset was caused by a Brown Out Detector.

RESET_SYS_REQ The last chip reset was caused by a System Reset requested by the ARM CPU.
RESET_WDT The last chip reset was caused by the Watchdog Timer.
RESET_WAKE_DEEP_PD The last chip reset was caused by a Wake-up I/O (GPIO or internal NTAG FD INT).
RESET_WAKE_PD The last CPU reset was caused by a Wake-up from Power down (many sources possible: timer, IO, ...).
RESET_SW_REQ The last chip reset was caused by a Software. ES2 Only

25.5.4 enum pm_ldo_volt_t

Enumerator

PM_LDO_VOLT_1_1V_DEFAULT LDO voltage 1.1V.
PM_LDO_VOLT_1_0V not safe at system start/wakeup and CPU clock switch to higher frequency

25.6 Function Documentation

25.6.1 void POWER_Init (void)

Optimize the LDO voltage for power saving Initialize the power domains

Returns

none

25.6.2 void POWER_SetTrimDefaultActiveVoltage (void)

Returns

none

25.6.3 void POWER_BodSetUp (void)

Enable the BOD core and BOD mem Disable the analog comnparator clock

Returns

none

25.6.4 void POWER_BodActivate (void)

Returns

none

25.6.5 static void POWER_EnablePD (pd_bit_t *en*) [inline], [static]

Note that enabling the bit powers down the peripheral

Parameters

en	peripheral for which to enable the PDRUNCFG bit

Returns

none

25.6.6 static void POWER_DisablePD (pd_bit_t *en*) [inline], [static]

Note that disabling the bit powers up the peripheral

Parameters

en	peripheral for which to disable the PDRUNCFG bit

Returns

none

25.6.7 static uint32_t POWER_GetIoWakeStatus (void) [inline], [static]

Allow to identify the wake-up source when waking up from Power-Down modes or Deep Power Down modes. Status is reset by POR, RSTN, WDT. bit in range from 0 to 21 are for DIO0 to DIO21 bit 22 is NTAG field detect wakeup source

Returns

IO and Field detect Wake-up source

25.6.8 static void POWER_EnterSleep (void) [inline], [static]

Note

: If the user desires to program a wakeup timer before going to sleep, it needs to use either the fsl_wtimer.h API or use the POWER_SetLowPower() API instead see POWER_ENTER_SLEEP

Returns

none

25.6.9 bool POWER_EnterPowerMode (pm_power_mode_t *pm_power_mode*, pm_power_config_t * *pm_power_config*)

If requested mode is PM_POWER_DOWN, the API will perform the clamping of the DIOs if the PIO register has the bit IO_CLAMPING set: SYSCON->RETENTIONCTRL.IOCLAMP will be set

Parameters

pm_power_mode	Power modes

See Also

pm_power_mode_t

Parameters

pm_power_config	Power config

See Also

pm_power_config_t

Returns

false if chip could not go to sleep. Configuration structure is incorrect

25.6.10 reset_cause_t POWER_GetResetCause (void)

Returns

reset_cause

25.6.11 uint32_t POWER_GetLibVersion (void)

Parameters

none	

Returns

version number of the power library

25.6.12 void POWER_BodVbatGetDefaultConfig (pm_bod_cfg_t * *bod_cfg_p*)

Parameters

bod_cfg_p	BOD config

See Also

pm_bod_cfg_t

Returns

none

25.6.13 bool POWER_BodVbatConfig (pm_bod_cfg_t * *bod_cfg_p*)

Parameters

bod_cfg_p	BOD config

See Also

pm_bod_cfg_t

Returns

false if configuration parameters are incorrect

25.6.14 void POWER_ApplyLdoActiveVoltage (pm_ldo_volt_t *ldoVolt*)

Parameters

ldoVolt	LDO voltage setting

See Also

pm_ldo_volt_t

Returns

none

Chapter 26
PWM: Pulse Width Modulator

26.1 Overview

The SDK provides a driver for the Pulse Width Modulator (PWM).

The function PWM_Init() initializes the PWM module with specified configurations, the function PWM_-GetDefaultConfig() could help to get the default configurations. The initialization function configures the module to use the specified clock for PWM operation.

The function PWM_SetupPwm() sets up the PWM channel for PWM output. The function can set up PWM signal property the channel. The PWM has 10 channels: 0 to 9. Each channel has its own period, compare match value, and polarity specified. The settings are applied to the specified channel requesting PWM output. The period and compare match are 16-bit values. At the compare match value, within the period, the PWM output toggles. The period value is loaded to downcounter, which decrements to 0. Once it reaches 0, it reloads the count and starts the signal out again, until the PWM channel is stopped. The function also sets up the channel output level after the channel is disabled. The 11th channel (ChannelAll) is a special channel which outputs the same output signals on other 10 channels (0 to 9) when it is set up and enabled.

The function PWM_ReadPeriodValue() reads the current period (downcounter value) for the PWM channel. The function PWM_ReadCompareValue() reads the compare match value for the PWM channel.

The function PWM_StartTimer() can be used to start the PWM channel. The function PWM_StopTimer() can be used to stop the PWM channel.

Provide functions to get and clear the PWM status.

Provide functions to enable/disable PWM interrupts and get current enabled interrupts.

26.2 Typical use case

26.2.1 PWM output

Configures PWM channel to output PWM signal.

```
int main(void)
{
    /* Structure of initialize PWM */
    pwm_config_t pwmConfig;
    pwm_setup_t pwmChan0;
    uint32_t pwmClockFrq;
    uint32_t pwmChan0Clk;

    /* Board pin, clock, debug console initialization */
    BOARD_InitHardware();

    PRINTF("PWM driver example\n");
```

```
    pwmClockFrq = CLOCK_GetFreq(kCLOCK_Pwm);

    PWM_GetDefaultConfig(&pwmConfig);

    /* Use 32MHz clock */
    pwmConfig.clk_sel = kPWM_Osc32Mclk;

    /* Initialize PWM */
    if (PWM_Init(BOARD_PWM_BASEADDR, &pwmConfig) != kStatus_Success)
    {
        PRINTF("PWM initialization failed\n");
        return 1;
    }

    /* Set up PWM channel 0 to generate PWM pulse of 100 us with 50% duty cycle */
    pwmChan0.pol_ctrl = kPWM_SetHighOnMatchLowOnPeriod;
    pwmChan0.dis_out_level = kPWM_SetLow;
    pwmChan0.prescaler_val = 0;
    pwmChan0Clk = pwmClockFrq / (1 + pwmChan0.prescaler_val);
    pwmChan0.period_val = USEC_TO_COUNT(100, pwmChan0Clk);
    pwmChan0.comp_val = pwmChan0.period_val / 2;
    if(PWM_SetupPwm (BOARD_PWM_BASEADDR, kPWM_Pwm0, &pwmChan0) !=  kStatus_Success)
    {
        PRINTF("PWM chan0 setup failed\n");
        return 1;
    }

    /* Start the PWM generation channel 0 */
    PWM_StartTimer(BOARD_PWM_BASEADDR, kPWM_Pwm0);

    while (1U)
    {
        ;
    }
}
```

Files

- file fsl_pwm.h

Data Structures

- struct pwm_config_t
  PWM configuration structure. More...
- struct pwm_setup_t
  PWM channel setup structure. More...

Enumerations

- enum pwm_channels_t {
  kPWM_Pwm0 = 0x0,
  kPWM_Pwm1,
  kPWM_Pwm2,
  kPWM_Pwm3,
  kPWM_Pwm4,
  kPWM_Pwm5,
  kPWM_Pwm6,
  kPWM_Pwm7,
  kPWM_Pwm8,
  kPWM_Pwm9,
  kPWM_PwmAll }
  PWM channel selection values.
- enum pwm_polarity_control_t {
  kPWM_SetHighOnMatchLowOnPeriod = 0x0,
  kPWM_SetLowOnMatchHighOnPeriod }
  PWM channel polarity control values.
- enum pwm_dis_output_level_t {
  kPWM_SetLow = 0x0,
  kPWM_SetHigh }
  PWM channel disable output level values.
- enum pwm_interrupt_enable_t {
  kPWM_InterruptDisabled = 0x0,
  kPWM_InterruptEnabled }
  PWM channel interrupt enable flags.
- enum pwm_interrupt_status_t {
  kPWM_NoInterrupt = 0x0,
  kPWM_InterruptPendig }
  PWM channel interrupt status flags.

Driver version

- #define FSL_PWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  PWM driver version 2.0.0.

Initialization and deinitialization

- void PWM_GetDefaultConfig (pwm_config_t ∗userConfig)
  Fill in the PWM config struct with the default settings.
- status_t PWM_Init (PWM_Type ∗base, const pwm_config_t ∗userConfig)
  Initializes the PWM module.
- void PWM_Deinit (PWM_Type ∗base)
  Gate the PWM module clock.

PWM module output

- status_t PWM_SetupPwm (PWM_Type *base, pwm_channels_t pwm_chan, pwm_setup_t *pwmSetup)
  Sets up the PWM channel.

PWM Interrupts Interface

- static void PWM_EnableInterrupts (PWM_Type *base, pwm_channels_t pwm_chan)
  Enable PWM channel interrupt.
- static void PWM_DisableInterrupts (PWM_Type *base, pwm_channels_t pwm_chan)
  Disable PWM channel interrupt.
- static uint32_t PWM_GetEnabledInterrupts (PWM_Type *base, pwm_channels_t pwm_chan)
  Gets the enabled PWM interrupts.

Status Interface

- uint32_t PWM_GetStatusFlags (PWM_Type *base, pwm_channels_t pwm_chan)
  Gets the PWM status flags.
- void PWM_ClearStatusFlags (PWM_Type *base, pwm_channels_t pwm_chan)
  Clears the PWM status flags.

Timer Start and Stop

- static void PWM_StartTimer (PWM_Type *base, pwm_channels_t pwm_chan)
  Start PWM channel.
- static void PWM_StopTimer (PWM_Type *base, pwm_channels_t pwm_chan)
  Stop PWM channel.
- uint16_t PWM_ReadPeriodValue (PWM_Type *base, pwm_channels_t pwm_chan)
  Read current period value for PWM channel.
- uint16_t PWM_ReadCompareValue (PWM_Type *base, pwm_channels_t pwm_chan)
  Read compare match value for PWM channel.

26.3 Data Structure Documentation

26.3.1 struct pwm_config_t

Data Fields

- pwm_clock_source_t clk_sel
  PWM clock select value.

26.3.2 struct pwm_setup_t

Data Fields

- pwm_polarity_control_t pol_ctrl
  Channel polarity control.

- pwm_dis_output_level_t dis_out_level
  Channel disable output level.
- uint16_t prescaler_val
  Channel Prescaler value.
- uint16_t period_val
  Channel PWM period value.
- uint16_t comp_val
  Channel compare match value.

26.3.2.0.0.27 Field Documentation

26.3.2.0.0.27.1 uint16_t pwm_setup_t::prescaler_val

- 10 bit value

26.4 Macro Definition Documentation

26.4.1 #define FSL_PWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

26.5 Enumeration Type Documentation

26.5.1 enum pwm_channels_t

Enumerator

kPWM_Pwm0 Channel 0.
kPWM_Pwm1 Channel 1.
kPWM_Pwm2 Channel 2.
kPWM_Pwm3 Channel 3.
kPWM_Pwm4 Channel 4.
kPWM_Pwm5 Channel 5.
kPWM_Pwm6 Channel 6.
kPWM_Pwm7 Channel 7.
kPWM_Pwm8 Channel 8.
kPWM_Pwm9 Channel 9.
kPWM_PwmAll Channel 10 - All the channels will output same output programmed in this channel.

26.5.2 enum pwm_polarity_control_t

Enumerator

kPWM_SetHighOnMatchLowOnPeriod Set high on compare match, set low at end of PWM period.

kPWM_SetLowOnMatchHighOnPeriod Set low on compare match, set high at end of PWM period.

26.5.3 enum pwm_dis_output_level_t

Enumerator

kPWM_SetLow Set to Low level.
kPWM_SetHigh Set to High level.

26.5.4 enum pwm_interrupt_enable_t

Enumerator

kPWM_InterruptDisabled PWM channel interrupt disabled.
kPWM_InterruptEnabled PWM channel interrupt enabled.

26.5.5 enum pwm_interrupt_status_t

Enumerator

kPWM_NoInterrupt PWM channel interrupt not occurred.
kPWM_InterruptPendig PWM channel interrupt pending.

26.6 Function Documentation

26.6.1 void PWM_GetDefaultConfig (pwm_config_t * *userConfig*)

The default values are:

```
*   userConfig->clk_sel = kPWM_Osc32Mclk;
*
```

Parameters

userConfig	Pointer to user's PWM config structure.

26.6.2 status_t PWM_Init (PWM_Type * *base,* const pwm_config_t * *userConfig*)

Call this API to ungate the PWM clock and configure the PWM HW.

Note

This API should be called at the beginning of the application to use the PWM driver, or any operation to the PWM module could cause hard fault because PWM module clock is not enabled. The configuration structure can be filled by user from scratch, or be set with default values by PWM_-GetDefaultConfig(). After calling this API, the application can configure PWM channels to generate PWM outputs. Example:

```
* pwm_config_t userConfig = {
* .clk_sel = kPWM_Fro48Mclk,
* };
* PWM_Init(PWM, &userConfig);
*
```

Parameters

base	PWM base address
userConfig	pointer to user configuration structure

Returns

kStatus_Success - Success
kStatus_InvalidArgument - Invalid input parameter

26.6.3 void PWM_Deinit (PWM_Type ∗ *base*)

Parameters

base	PWM base address

26.6.4 status_t PWM_SetupPwm (PWM_Type ∗ *base,* pwm_channels_t *pwm_chan,* pwm_setup_t ∗ *pwmSetup*)

The function initializes the PWM channel according to the parameters passed in by the user. The function sets up the PWM compare match register & period registers.

Parameters

base	PWM base address
pwm_chan	PWM channel select value
pwmSetup	Pointer to PWM user setup structure

Returns

kStatus_Success - Success
kStatus_InvalidArgument - Invalid input parameter

26.6.5 static void PWM_EnableInterrupts (PWM_Type * *base,* pwm_channels_t *pwm_chan*) [inline], [static]

This function enables the interrupt for the specified PWM channel.

Parameters

base	PWM base address
pwm_chan	PWM channel select value

26.6.6 static void PWM_DisableInterrupts (PWM_Type * *base,* pwm_channels_t *pwm_chan*) [inline], [static]

This function disables the interrupt for the specified PWM channel.

Parameters

base	PWM base address
pwm_chan	PWM channel select value

26.6.7 static uint32_t PWM_GetEnabledInterrupts (PWM_Type * *base,* pwm_channels_t *pwm_chan*) [inline], [static]

Parameters

base	PWM base address
pwm_chan	PWM channel select value

Returns

PWM interrupt enabled status. This is the one of the values specified in enumeration pwm_interrupt_enable_t

26.6.8 uint32_t PWM_GetStatusFlags (PWM_Type * *base*, pwm_channels_t *pwm_chan*)

Parameters

base	PWM base address
pwm_chan	PWM channel select value

Returns

The status flags. This is the one of the value of members of the enumeration pwm_interrupt_status_t

26.6.9 void PWM_ClearStatusFlags (PWM_Type * *base*, pwm_channels_t *pwm_chan*)

Parameters

base	PWM base address
pwm_chan	PWM channel select value

26.6.10 static void PWM_StartTimer (PWM_Type * *base*, pwm_channels_t *pwm_chan*) [inline], [static]

The API will start PWM channel output on the pin. Before calling this API, make sure that the PWM channel is set up using PWM_SetupPwm() API.

Parameters

base	PWM base address
pwm_chan	PWM channel select value

26.6.11 static void PWM_StopTimer (PWM_Type * *base*, pwm_channels_t *pwm_chan*) [inline], [static]

The API will stop PWM channel output on the pin.

Parameters

base	PWM base address
pwm_chan	PWM channel select value

26.6.12 uint16_t PWM_ReadPeriodValue (PWM_Type * *base*, pwm_channels_t *pwm_chan*)

The API will read the current period value set for the PWM channel.

Parameters

base	PWM base address
pwm_chan	PWM channel select value

Returns

16-bit period value

26.6.13 uint16_t PWM_ReadCompareValue (PWM_Type * *base*, pwm_channels_t *pwm_chan*)

The API will read the compare match value set for the PWM channel.

Parameters

base	PWM base address
pwm_chan	PWM channel select value

Returns

16-bit period value

Chapter 27
RNG: Random Number generator

27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number generator module of MCU-Xpresso SDK devices.

Files

- file fsl_rng.h

Enumerations

- enum trng_mode_t {
  trng_UpdateOnce = 0x1,
  trng_FreeRunning = 0x2 }

Functions

- status_t TRNG_GetDefaultConfig (trng_config_t ∗userConfig)
  Gets Default config of TRNG.
- status_t TRNG_Init (RNG_Type ∗base, const trng_config_t ∗userConfig)
  Initializes the TRNG.
- void TRNG_Deinit (RNG_Type ∗base)
  Shuts down the TRNG.
- status_t TRNG_GetRandomData (RNG_Type ∗base, void ∗data, size_t data_size)
  Gets random data.

27.2 Enumeration Type Documentation

27.2.1 enum trng_mode_t

RNG return status types RNG operating modes

Enumerator

trng_UpdateOnce TRNG update once & disable.
trng_FreeRunning TRNG updates continuously.

27.3 Function Documentation

27.3.1 status_t TRNG_GetDefaultConfig (trng_config_t ∗ *userConfig*)

This function initializes the TRNG configuration structure.

Parameters

userConfig	Pointer to TRNG configuration structure

27.3.2 status_t TRNG_Init (RNG_Type ∗ *base,* const trng_config_t ∗ *userConfig*)

This function initializes the TRNG.

Parameters

base	TRNG base address
userConfig	The configuration of TRNG

Returns

kStatus_Success - Success kStatus_InvalidArgument - Invalid parameter

27.3.3 void TRNG_Deinit (RNG_Type ∗ *base*)

This function shuts down the TRNG.

Parameters

base	TRNG base address

27.3.4 status_t TRNG_GetRandomData (RNG_Type ∗ *base,* void ∗ *data,* size_t *data_size*)

This function gets random data from the TRNG.

Parameters

base	TRNG base address
data	pointer to user buffer to be filled by random data

data_size	size of data in bytes

Returns

TRNG status

Chapter 28
RTC: Real Time Clock

28.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC).

28.2 Function groups

The RTC driver supports operating the module as a time counter.

28.2.1 Initialization and deinitialization

The function RTC_Init() initializes the RTC with specified configurations. The function RTC_GetDefault-Config() gets the default configurations.

The function RTC_Deinit() disables the RTC timer and disables the module clock.

28.2.2 Set & Get Datetime

The function RTC_SetDatetime() sets the timer period in seconds. User passes in the details in date & time format by using the below data structure.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc The function RTC_GetDatetime() reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

28.2.3 Set & Get Alarm

The function RTC_SetAlarm() sets the alarm time period in seconds. User passes in the details in date & time format by using the datetime data structure.

The function RTC_GetAlarm() reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

28.2.4 Start & Stop timer

The function RTC_StartTimer() starts the RTC time counter.

The function RTC_StopTimer() stops the RTC time counter.

28.2.5 Status

Provides functions to get and clear the RTC status.

28.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

28.2.7 High resolution timer

Provides functions to enable high resolution timer and set and get the wake time.

28.3 Typical use case

28.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc

Files

- file fsl_rtc.h

Data Structures

- struct rtc_datetime_t
  Structure is used to hold the date and time. More...

Enumerations

- enum rtc_interrupt_enable_t {
  kRTC_AlarmInterruptEnable = RTC_CTRL_ALARMDPD_EN_MASK,
  kRTC_WakeupInterruptEnable = RTC_CTRL_WAKEDPD_EN_MASK }
  List of RTC interrupts.
- enum rtc_status_flags_t {
  kRTC_AlarmFlag = RTC_CTRL_ALARM1HZ_MASK,
  kRTC_WakeupFlag = RTC_CTRL_WAKE1KHZ_MASK }
  List of RTC flags.

Functions

- static void RTC_SetWakeupCount (RTC_Type ∗base, uint16_t wakeupValue)
  Enable the RTC high resolution timer and set the wake-up time.
- static uint16_t RTC_GetWakeupCount (RTC_Type ∗base)
  Read actual RTC counter value.

- static void RTC_Reset (RTC_Type *base)
  Performs a software reset on the RTC module.

Driver version

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

Initialization and deinitialization

- void RTC_Init (RTC_Type *base)
  Ungates the RTC clock and enables the RTC oscillator.
- static void RTC_Deinit (RTC_Type *base)
  Stop the timer and gate the RTC clock.

Current Time & Alarm

- status_t RTC_SetDatetime (RTC_Type *base, const rtc_datetime_t *datetime)
  Sets the RTC date and time according to the given time structure.
- void RTC_GetDatetime (RTC_Type *base, rtc_datetime_t *datetime)
  Gets the RTC time and stores it in the given time structure.
- status_t RTC_SetAlarm (RTC_Type *base, const rtc_datetime_t *alarmTime)
  Sets the RTC alarm time.
- void RTC_GetAlarm (RTC_Type *base, rtc_datetime_t *datetime)
  Returns the RTC alarm time.

Interrupt Interface

- static void RTC_EnableInterrupts (RTC_Type *base, uint32_t mask)
  Enables the selected RTC interrupts.
- static void RTC_DisableInterrupts (RTC_Type *base, uint32_t mask)
  Disables the selected RTC interrupts.
- static uint32_t RTC_GetEnabledInterrupts (RTC_Type *base)
  Gets the enabled RTC interrupts.

Status Interface

- static uint32_t RTC_GetStatusFlags (RTC_Type *base)
  Gets the RTC status flags.
- static void RTC_ClearStatusFlags (RTC_Type *base, uint32_t mask)
  Clears the RTC status flags.

Timer Start and Stop

- static void RTC_StartTimer (RTC_Type *base)
  Starts the RTC time counter.
- static void RTC_StopTimer (RTC_Type *base)
  Stops the RTC time counter.

28.4 Data Structure Documentation

28.4.1 struct 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.

28.4.1.0.0.28 Field Documentation

28.4.1.0.0.28.1 uint16_t rtc_datetime_t::year

28.4.1.0.0.28.2 uint8_t rtc_datetime_t::month

28.4.1.0.0.28.3 uint8_t rtc_datetime_t::day

28.4.1.0.0.28.4 uint8_t rtc_datetime_t::hour

28.4.1.0.0.28.5 uint8_t rtc_datetime_t::minute

28.4.1.0.0.28.6 uint8_t rtc_datetime_t::second

28.5 Enumeration Type Documentation

28.5.1 enum rtc_interrupt_enable_t

Enumerator

kRTC_AlarmInterruptEnable Alarm interrupt.
kRTC_WakeupInterruptEnable Wake-up interrupt.

28.5.2 enum rtc_status_flags_t

Enumerator

kRTC_AlarmFlag Alarm flag.
kRTC_WakeupFlag 1kHz wake-up timer flag

28.6 Function Documentation

28.6.1 void RTC_Init (RTC_Type * *base*)

Note

This API should be called at the beginning of the application using the RTC driver.

Parameters

base	RTC peripheral base address

28.6.2 static void RTC_Deinit (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

28.6.3 status_t RTC_SetDatetime (RTC_Type * *base*, const rtc_datetime_t * *datetime*)

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

base	RTC peripheral base address
datetime	Pointer to structure where the date and time details to set are stored

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

28.6.4 void RTC_GetDatetime (RTC_Type * *base*, rtc_datetime_t * *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to structure where the date and time details are stored.

28.6.5 status_t RTC_SetAlarm (RTC_Type * *base,* const rtc_datetime_t * *alarmTime*)

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

base	RTC peripheral base address
alarmTime	Pointer to structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

28.6.6 void RTC_GetAlarm (RTC_Type * *base,* rtc_datetime_t * *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to structure where the alarm date and time details are stored.

28.6.7 static void RTC_SetWakeupCount (RTC_Type * *base,* uint16_t *wakeupValue*) [inline], [static]

Parameters

base	RTC peripheral base address
wakeupValue	The value to be loaded into the RTC WAKE register

28.6.8 static uint16_t RTC_GetWakeupCount (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

28.6.9 static void RTC_EnableInterrupts (RTC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	RTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration rtc_-interrupt_enable_t

28.6.10 static void RTC_DisableInterrupts (RTC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	RTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration rtc_-interrupt_enable_t

28.6.11 static uint32_t RTC_GetEnabledInterrupts (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration rtc_interrupt_enable_t

28.6.12 static uint32_t RTC_GetStatusFlags (RTC_Type ∗ *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

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

28.6.13 static void RTC_ClearStatusFlags (RTC_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	RTC peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration rtc_status_flags_t

28.6.14 static void RTC_StartTimer (RTC_Type ∗ *base*) [inline], [static]

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

base	RTC peripheral base address

28.6.15 static void RTC_StopTimer (RTC_Type * *base*) [inline], [static]

RTC's seconds register can be written to only when the timer is stopped.

Parameters

base	RTC peripheral base address

28.6.16 static void RTC_Reset (RTC_Type * *base*) [inline], [static]

This resets all RTC registers to their reset value. The bit is cleared by software explicitly clearing it.

Parameters

base	RTC peripheral base address

Chapter 29
SPIFI: SPIFI flash interface driver

29.1 Overview

Modules

- SPIFI DMA Driver
- SPIFI Driver

Data Structures

- struct spifi_command_t
  SPIFI command structure. More...
- struct spifi_config_t
  SPIFI region configuration structure. More...
- struct spifi_transfer_t
  Transfer structure for SPIFI. More...
- struct spifi_dma_handle_t
  SPIFI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ spifi_dma_callback_t)(SPIFI_Type ∗base, spifi_dma_handle_t ∗handle, status_t status, void ∗userData)
  SPIFI DMA transfer callback function for finish and error.

Enumerations

- enum _status_t {
  kStatus_SPIFI_Idle = MAKE_STATUS(kStatusGroup_SPIFI, 0),
  kStatus_SPIFI_Busy = MAKE_STATUS(kStatusGroup_SPIFI, 1),
  kStatus_SPIFI_Error = MAKE_STATUS(kStatusGroup_SPIFI, 2) }
  Status structure of SPIFI.
- enum spifi_interrupt_enable_t { kSPIFI_CommandFinishInterruptEnable = SPIFI_CTRL_INTEN_MASK }
  SPIFI interrupt source.
- enum spifi_spi_mode_t {
  kSPIFI_SPISckLow = 0x0U,
  kSPIFI_SPISckHigh = 0x1U }
  SPIFI SPI mode select.
- enum spifi_dual_mode_t {
  kSPIFI_QuadMode = 0x0U,
  kSPIFI_DualMode = 0x1U }
  SPIFI dual mode select.

- enum spifi_data_direction_t {
  kSPIFI_DataInput = 0x0U,
  kSPIFI_DataOutput = 0x1U }
  SPIFI data direction.
- enum spifi_command_format_t {
  kSPIFI_CommandAllSerial = 0x0,
  kSPIFI_CommandDataQuad = 0x1U,
  kSPIFI_CommandOpcodeSerial = 0x2U,
  kSPIFI_CommandAllQuad = 0x3U }
  SPIFI command opcode format.
- enum spifi_command_type_t {
  kSPIFI_CommandOpcodeOnly = 0x1U,
  kSPIFI_CommandOpcodeAddrOneByte = 0x2U,
  kSPIFI_CommandOpcodeAddrTwoBytes = 0x3U,
  kSPIFI_CommandOpcodeAddrThreeBytes = 0x4U,
  kSPIFI_CommandOpcodeAddrFourBytes = 0x5U,
  kSPIFI_CommandNoOpcodeAddrThreeBytes = 0x6U,
  kSPIFI_CommandNoOpcodeAddrFourBytes = 0x7U }
  SPIFI command type.
- enum _spifi_status_flags {
  kSPIFI_MemoryCommandWriteFinished = SPIFI_STAT_MCINIT_MASK,
  kSPIFI_CommandWriteFinished = SPIFI_STAT_CMD_MASK,
  kSPIFI_InterruptRequest = SPIFI_STAT_INTRQ_MASK }
  SPIFI status flags.

Functions

- static void SPIFI_EnableDMA (SPIFI_Type ∗base, bool enable)
  Enable or disable DMA request for SPIFI.
- static uint32_t SPIFI_GetDataRegisterAddress (SPIFI_Type ∗base)
  Gets the SPIFI data register address.
- static void SPIFI_WriteData (SPIFI_Type ∗base, uint32_t data)
  Write a word data in address of SPIFI.
- void SPIFI_WriteBuffer (SPIFI_Type ∗base, uint8_t ∗buf, size_t size_to_write)
  Write a buffer worth of data to SPIFI .
- static void SPIFI_WriteDataByte (SPIFI_Type ∗base, uint8_t data)
  Write a byte data in address of SPIFI.
- void SPIFI_WriteDataHalfword (SPIFI_Type ∗base, uint16_t data)
  Write a halfword data in address of SPIFI.
- static uint32_t SPIFI_ReadData (SPIFI_Type ∗base)
  Read data from serial flash.
- static uint8_t SPIFI_ReadDataByte (SPIFI_Type ∗base)
  Read a byte data from serial flash.
- uint16_t SPIFI_ReadDataHalfword (SPIFI_Type ∗base)
  Read a halfword data from serial flash.

Driver version

- #define FSL_SPIFI_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

SPIFI driver version 2.0.2.

Initialization and deinitialization

- uint32_t SPIFI_GetInstance (SPIFI_Type *base)
  Get the SPIFI instance from peripheral base address.
- void SPIFI_Init (SPIFI_Type *base, const spifi_config_t *config)
  Initializes the SPIFI with the user configuration structure.
- void SPIFI_GetDefaultConfig (spifi_config_t *config)
  Get SPIFI default configure settings.
- void SPIFI_Deinit (SPIFI_Type *base)
  Deinitializes the SPIFI regions.

Basic Control Operations

- void SPIFI_SetCommand (SPIFI_Type *base, const spifi_command_t *cmd)
  Set SPIFI flash command.
- static void SPIFI_SetCommandAddress (SPIFI_Type *base, uint32_t addr)
  Set SPIFI command address.
- static void SPIFI_SetIntermediateData (SPIFI_Type *base, uint32_t val)
  Set SPIFI intermediate data.
- static void SPIFI_SetCacheLimit (SPIFI_Type *base, uint32_t val)
  Set SPIFI Cache limit value.
- static void SPIFI_ResetCommand (SPIFI_Type *base)
  Reset the command field of SPIFI.
- void SPIFI_SetMemoryCommand (SPIFI_Type *base, const spifi_command_t *cmd)
  Set SPIFI flash AHB read command.
- static void SPIFI_EnableInterrupt (SPIFI_Type *base, uint32_t mask)
  Enable SPIFI interrupt.
- static void SPIFI_DisableInterrupt (SPIFI_Type *base, uint32_t mask)
  Disable SPIFI interrupt.

Status

- static uint32_t SPIFI_GetStatusFlag (SPIFI_Type *base)
  Get the status of all interrupt flags for SPIFI.

Driver version

- #define FSL_SPIFI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  SPIFI DMA driver version 2.0.2.

DMA Transactional

- void SPIFI_TransferTxCreateHandleDMA (SPIFI_Type *base, spifi_dma_handle_t *handle, spifi_dma_callback_t callback, void *userData, dma_handle_t *dmaHandle)
  Initializes the SPIFI handle for send which is used in transactional functions and set the callback.
- void SPIFI_TransferRxCreateHandleDMA (SPIFI_Type *base, spifi_dma_handle_t *handle, spifi_dma_callback_t callback, void *userData, dma_handle_t *dmaHandle)
  Initializes the SPIFI handle for receive which is used in transactional functions and set the callback.

- status_t SPIFI_TransferSendDMA (SPIFI_Type *base, spifi_dma_handle_t *handle, spifi_transfer_t *xfer)
  Transfers SPIFI data using an DMA non-blocking method.
- status_t SPIFI_TransferReceiveDMA (SPIFI_Type *base, spifi_dma_handle_t *handle, spifi_transfer_t *xfer)
  Receives data using an DMA non-blocking method.
- void SPIFI_TransferAbortSendDMA (SPIFI_Type *base, spifi_dma_handle_t *handle)
  Aborts the sent data using DMA.
- void SPIFI_TransferAbortReceiveDMA (SPIFI_Type *base, spifi_dma_handle_t *handle)
  Aborts the receive data using DMA.
- status_t SPIFI_TransferGetSendCountDMA (SPIFI_Type *base, spifi_dma_handle_t *handle, size_t *count)
  Gets the transferred counts of send.
- status_t SPIFI_TransferGetReceiveCountDMA (SPIFI_Type *base, spifi_dma_handle_t *handle, size_t *count)
  Gets the status of the receive transfer.

29.2 Data Structure Documentation

29.2.1 struct spifi_command_t

Data Fields

- uint16_t dataLen
  How many data bytes are needed in this command.
- bool isPollMode
  For command need to read data from serial flash.
- spifi_data_direction_t direction
  Data direction of this command.
- uint8_t intermediateBytes
  How many intermediate bytes needed.
- spifi_command_format_t format
  Command format.
- spifi_command_type_t type
  Command type.
- uint8_t opcode
  Command opcode value.

29.2.1.0.0.29 Field Documentation

29.2.1.0.0.29.1 uint16_t spifi_command_t::dataLen

29.2.1.0.0.29.2 spifi_data_direction_t spifi_command_t::direction

29.2.2 struct spifi_config_t

Data Fields

- uint16_t timeout

SPI transfer timeout, the unit is SCK cycles.
- uint8_t csHighTime
  CS high time cycles.
- bool disablePrefetch
  True means SPIFI will not attempt a speculative prefetch.
- bool disableCachePrefech
  Disable prefetch of cache line.
- bool isFeedbackClock
  Is data sample uses feedback clock.
- spifi_spi_mode_t spiMode
  SPIFI spi mode select.
- bool isReadFullClockCycle
  If enable read full clock cycle.
- spifi_dual_mode_t dualMode
  SPIFI dual mode, dual or quad.

29.2.2.0.0.30 Field Documentation

29.2.2.0.0.30.1 bool spifi_config_t::disablePrefetch

29.2.2.0.0.30.2 bool spifi_config_t::isFeedbackClock

29.2.2.0.0.30.3 bool spifi_config_t::isReadFullClockCycle

29.2.2.0.0.30.4 spifi_dual_mode_t spifi_config_t::dualMode

29.2.3 struct spifi_transfer_t

Data Fields

- uint8_t ∗ data
  Pointer to data to transmit.
- size_t dataSize
  Bytes to be transmit.

29.2.4 struct _spifi_dma_handle

Data Fields

- dma_handle_t ∗ dmaHandle
  DMA handler for SPIFI send.
- size_t transferSize
  Bytes need to transfer.
- uint32_t state
  Internal state for SPIFI DMA transfer.
- spifi_dma_callback_t callback
  Callback for users while transfer finish or error occurred.
- void ∗ userData

User callback parameter.

29.2.4.0.0.31 Field Documentation

29.2.4.0.0.31.1 size_t spifi_dma_handle_t::transferSize

29.3 Macro Definition Documentation

29.3.1 #define FSL_SPIFI_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

29.3.2 #define FSL_SPIFI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

29.4 Enumeration Type Documentation

29.4.1 enum _status_t

Enumerator

kStatus_SPIFI_Idle SPIFI is in idle state.
kStatus_SPIFI_Busy SPIFI is busy.
kStatus_SPIFI_Error Error occurred during SPIFI transfer.

29.4.2 enum spifi_interrupt_enable_t

Enumerator

kSPIFI_CommandFinishInterruptEnable Interrupt while command finished.

29.4.3 enum spifi_spi_mode_t

Enumerator

kSPIFI_SPISckLow SCK low after last bit of command, keeps low while CS high.
kSPIFI_SPISckHigh SCK high after last bit of command and while CS high.

29.4.4 enum spifi_dual_mode_t

Enumerator

kSPIFI_QuadMode SPIFI uses IO3:0.
kSPIFI_DualMode SPIFI uses IO1:0.

29.4.5 enum spifi_data_direction_t

Enumerator

kSPIFI_DataInput Data input from serial flash.
kSPIFI_DataOutput Data output to serial flash.

29.4.6 enum spifi_command_format_t

Enumerator

kSPIFI_CommandAllSerial All fields of command are serial.
kSPIFI_CommandDataQuad Only data field is dual/quad, others are serial.
kSPIFI_CommandOpcodeSerial Only opcode field is serial, others are quad/dual.
kSPIFI_CommandAllQuad All fields of command are dual/quad mode.

29.4.7 enum spifi_command_type_t

Enumerator

kSPIFI_CommandOpcodeOnly Command only have opcode, no address field.
kSPIFI_CommandOpcodeAddrOneByte Command have opcode and also one byte address field.
kSPIFI_CommandOpcodeAddrTwoBytes Command have opcode and also two bytes address field.
kSPIFI_CommandOpcodeAddrThreeBytes Command have opcode and also three bytes address field.
kSPIFI_CommandOpcodeAddrFourBytes Command have opcode and also four bytes address field.
kSPIFI_CommandNoOpcodeAddrThreeBytes Command have no opcode and three bytes address field.
kSPIFI_CommandNoOpcodeAddrFourBytes Command have no opcode and four bytes address field.

29.4.8 enum _spifi_status_flags

Enumerator

kSPIFI_MemoryCommandWriteFinished Memory command write finished.
kSPIFI_CommandWriteFinished Command write finished.
kSPIFI_InterruptRequest CMD flag from 1 to 0, means command execute finished.

29.5 Function Documentation

29.5.1 uint32_t SPIFI_GetInstance (SPIFI_Type * base)

Parameters

base	SPIFI peripheral base address.

Returns

SPIFI instance.

29.5.2 void SPIFI_Init (SPIFI_Type * *base,* const spifi_config_t * *config*)

This function configures the SPIFI module with the user-defined configuration.

Parameters

base	SPIFI peripheral base address.
config	The pointer to the configuration structure.

29.5.3 void SPIFI_GetDefaultConfig (spifi_config_t * *config*)

Parameters

config	SPIFI config structure pointer.

29.5.4 void SPIFI_Deinit (SPIFI_Type * *base*)

Parameters

base	SPIFI peripheral base address.

29.5.5 void SPIFI_SetCommand (SPIFI_Type * *base,* const spifi_command_t * *cmd*)

Parameters

base	SPIFI peripheral base address.
cmd	SPIFI command structure pointer.

29.5.6 static void SPIFI_SetCommandAddress (SPIFI_Type * *base*, uint32_t *addr*) [inline], [static]

Parameters

base	SPIFI peripheral base address.
addr	Address value for the command.

29.5.7 static void SPIFI_SetIntermediateData (SPIFI_Type * *base*, uint32_t *val*) [inline], [static]

Before writing a command wihch needs specific intermediate value, users shall call this function to write it. The main use of this function for current serial flash is to select no-opcode mode and cancelling this mode. As dummy cycle do not care about the value, no need to call this function.

Parameters

base	SPIFI peripheral base address.
val	Intermediate data.

29.5.8 static void SPIFI_SetCacheLimit (SPIFI_Type * *base*, uint32_t *val*) [inline], [static]

SPIFI includes caching of prevously-accessed data to improve performance. Software can write an address to this function, to prevent such caching at and above the address.

Parameters

base	SPIFI peripheral base address.

val	Zero-based upper limit of cacheable memory.

29.5.9 static void SPIFI_ResetCommand (SPIFI_Type ∗ *base*) [inline], [static]

This function is used to abort the current command or memory mode.

Parameters

base	SPIFI peripheral base address.

29.5.10 void SPIFI_SetMemoryCommand (SPIFI_Type ∗ *base,* const spifi_command_t ∗ *cmd*)

Call this function means SPIFI enters to memory mode, while users need to use command, a SPIFI_Reset-Command shall be called.

Parameters

base	SPIFI peripheral base address.
cmd	SPIFI command structure pointer.

29.5.11 static void SPIFI_EnableInterrupt (SPIFI_Type ∗ *base,* uint32_t *mask*) [inline], [static]

The interrupt is triggered only in command mode, and it means the command now is finished.

Parameters

base	SPIFI peripheral base address.
mask	SPIFI interrupt enable mask. It is a logic OR of members the enumeration :: spifi_-interrupt_enable_t

29.5.12 static void SPIFI_DisableInterrupt (SPIFI_Type ∗ *base,* uint32_t *mask*) [inline], [static]

The interrupt is triggered only in command mode, and it means the command now is finished.

Parameters

base	SPIFI peripheral base address.
mask	SPIFI interrupt enable mask. It is a logic OR of members the enumeration :: spifi_-interrupt_enable_t

29.5.13 static uint32_t SPIFI_GetStatusFlag (SPIFI_Type * *base*) [inline], [static]

Parameters

base	SPIFI peripheral base address.

Returns

SPIFI flag status

29.5.14 static void SPIFI_EnableDMA (SPIFI_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	SPIFI peripheral base address.
enable	True means enable DMA and false means disable DMA.

29.5.15 static uint32_t SPIFI_GetDataRegisterAddress (SPIFI_Type * *base*) [inline], [static]

This API is used to provide a transfer address for the SPIFI DMA transfer configuration.

Parameters

base	SPIFI base pointer

Returns

data register address

29.5.16 static void SPIFI_WriteData (SPIFI_Type * *base*, uint32_t *data*) [inline], [static]

Users can write a page or at least a word data into SPIFI address. Beware: certain SPIFI implementations (such as that of JN5189/QN9090/K32W061) require that the data do not exceed the actual size of the command, so cannot call SPIFI_WriteData when less than 32 bits are expected.

Parameters

base	SPIFI peripheral base address.
data	Data that need to be written.

29.5.17 void SPIFI_WriteBuffer (SPIFI_Type * *base*, uint8_t * *buf*, size_t *size_to_write*)

Used for transaction requiring less than 32 bits of data

Parameters

base	SPIFI peripheral base address.
buf	pointer on octet buffer to be written to the SPIFI.
size_to_write	size of buffer.

29.5.18 static void SPIFI_WriteDataByte (SPIFI_Type * *base*, uint8_t *data*) [inline], [static]

Users can write a byte data into SPIFI address.

Parameters

base	SPIFI peripheral base address.
data	Data need be write.

29.5.19 void SPIFI_WriteDataHalfword (SPIFI_Type * *base*, uint16_t *data*)

Users can write a halfword data into SPIFI address.

Parameters

base	SPIFI peripheral base address.
data	Data need be write.

29.5.20 static uint32_t SPIFI_ReadData (SPIFI_Type * *base*) [inline], [static]

Users should notice before call this function, the data length field in command register shall be larger than 4, otherwise a hard fault will happen.

Parameters

base	SPIFI peripheral base address.

Returns

Data input from flash.

29.5.21 static uint8_t SPIFI_ReadDataByte (SPIFI_Type * *base*) [inline], [static]

Parameters

base	SPIFI peripheral base address.

Returns

Data input from flash.

29.5.22 uint16_t SPIFI_ReadDataHalfword (SPIFI_Type * *base*)

Parameters

base	SPIFI peripheral base address.

Returns

Data input from flash.

29.5.23 void SPIFI_TransferTxCreateHandleDMA (SPIFI_Type * *base*, spifi_dma_handle_t * *handle*, spifi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *dmaHandle*)

Parameters

base	SPIFI peripheral base address
handle	Pointer to spifi_dma_handle_t structure
callback	SPIFI callback, NULL means no callback.
userData	User callback function data.
rxDmaHandle	User requested DMA handle for DMA transfer

29.5.24 void SPIFI_TransferRxCreateHandleDMA (SPIFI_Type * *base*, spifi_dma_handle_t * *handle*, spifi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *dmaHandle*)

Parameters

base	SPIFI peripheral base address
handle	Pointer to spifi_dma_handle_t structure
callback	SPIFI callback, NULL means no callback.
userData	User callback function data.
rxDmaHandle	User requested DMA handle for DMA transfer

29.5.25 status_t SPIFI_TransferSendDMA (SPIFI_Type * *base*, spifi_dma_handle_t * *handle*, spifi_transfer_t * *xfer*)

This function writes data to the SPIFI transmit FIFO. This function is non-blocking.

Parameters

base	Pointer to QuadSPI Type.
handle	Pointer to spifi_dma_handle_t structure
xfer	SPIFI transfer structure.

29.5.26 status_t SPIFI_TransferReceiveDMA (SPIFI_Type * *base*, spifi_dma_handle_t * *handle*, spifi_transfer_t * *xfer*)

This function receive data from the SPIFI receive buffer/FIFO. This function is non-blocking.

Parameters

base	Pointer to QuadSPI Type.
handle	Pointer to spifi_dma_handle_t structure
xfer	SPIFI transfer structure.

29.5.27 void SPIFI_TransferAbortSendDMA (SPIFI_Type * *base*, spifi_dma_handle_t * *handle*)

This function aborts the sent data using DMA.

Parameters

base	SPIFI peripheral base address.
handle	Pointer to spifi_dma_handle_t structure

29.5.28 void SPIFI_TransferAbortReceiveDMA (SPIFI_Type * *base*, spifi_dma_handle_t * *handle*)

This function abort receive data which using DMA.

Parameters

base	SPIFI peripheral base address.
handle	Pointer to spifi_dma_handle_t structure

29.5.29 status_t SPIFI_TransferGetSendCountDMA (SPIFI_Type * *base*, spifi_dma_handle_t * *handle*, size_t * *count*)

Parameters

base	Pointer to QuadSPI Type.
handle	Pointer to spifi_dma_handle_t structure.
count	Bytes sent.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

29.5.30 status_t SPIFI_TransferGetReceiveCountDMA (SPIFI_Type ∗ *base*, spifi_dma_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	Pointer to QuadSPI Type.
handle	Pointer to spifi_dma_handle_t structure
count	Bytes received.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

29.6 SPIFI Driver

SPIFI driver includes functional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPIFI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPIFI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPIFI functional operation groups provide the functional API set.

29.6.1 Typical use case

29.6.1.1 SPIFI transfer using an polling method

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

29.7 SPIFI DMA Driver

This chapter describes the programming interface of the SPIFI DMA driver. SPIFI DMA driver includes transactional APIs.

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 spifi_handle_t as the first parameter. Initialize the handle by calling the SPIFI_TransferCreateHandleDMA() API.

29.7.1 Typical use case

29.7.1.1 SPIFI Send/receive using a DMA method

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

Chapter 30
WWDT: Windowed Watchdog Timer Driver

30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

30.2 Function groups

30.2.1 Initialization and deinitialization

The function WWDT_Init() initializes the watchdog timer with specified configurations. The configurations include timeout value and whether to enable watchdog after init. The function WWDT_GetDefaultConfig() gets the default configurations.

The function WWDT_Deinit() disables the watchdog and the module clock.

30.2.2 Status

Provides functions to get and clear the WWDT status.

30.2.3 Interrupt

Provides functions to enable/disable WWDT interrupts and get current enabled interrupts.

30.2.4 Watch dog Refresh

The function WWDT_Refresh() feeds the WWDT.

30.3 Typical use case

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

Files

- file fsl_wwdt.h

Data Structures

- struct wwdt_config_t
  Describes WWDT configuration structure. More...

Enumerations

- enum _wwdt_status_flags_t {
kWWDT_TimeoutFlag = WWDT_MOD_WDTOF_MASK,
kWWDT_WarningFlag = WWDT_MOD_WDINT_MASK }
  WWDT status flags.

Driver version

- #define FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  Defines WWDT driver version 2.1.3.

Refresh sequence

- #define WWDT_FIRST_WORD_OF_REFRESH (0xAAU)
  First word of refresh sequence.
- #define WWDT_SECOND_WORD_OF_REFRESH (0x55U)
  Second word of refresh sequence.

WWDT Initialization and De-initialization

- void WWDT_GetDefaultConfig (wwdt_config_t ∗config)
  Initializes WWDT configure structure.
- void WWDT_Init (WWDT_Type ∗base, const wwdt_config_t ∗config)
  Initializes the WWDT.
- void WWDT_Deinit (WWDT_Type ∗base)
  Shuts down the WWDT.

WWDT Functional Operation

- static void WWDT_Enable (WWDT_Type ∗base)
  Enables the WWDT module.
- static void WWDT_Disable (WWDT_Type ∗base)
  Disables the WWDT module.
- static uint32_t WWDT_GetStatusFlags (WWDT_Type ∗base)
  Gets all WWDT status flags.
- void WWDT_ClearStatusFlags (WWDT_Type ∗base, uint32_t mask)
  Clear WWDT flag.
- static void WWDT_SetWarningValue (WWDT_Type ∗base, uint32_t warningValue)
  Set the WWDT warning value.
- static void WWDT_SetTimeoutValue (WWDT_Type ∗base, uint32_t timeoutCount)
  Set the WWDT timeout value.
- static void WWDT_SetWindowValue (WWDT_Type ∗base, uint32_t windowValue)
  Sets the WWDT window value.
- void WWDT_Refresh (WWDT_Type ∗base)
  Refreshes the WWDT timer.

30.4 Data Structure Documentation

30.4.1 struct wwdt_config_t

Data Fields

- bool enableWwdt
  Enables or disables WWDT.
- bool enableWatchdogReset
  true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset
- bool enableWatchdogProtect
  true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time
- bool enableLockOscillator
  true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator
- uint32_t windowValue
  Window value, set this to 0xFFFFFF if windowing is not in effect.
- uint32_t timeoutValue
  Timeout value.
- uint32_t warningValue
  Watchdog time counter value that will generate a warning interrupt.
- uint32_t clockFreq_Hz
  Watchdog clock source frequency.

30.4.1.0.0.32 Field Documentation

30.4.1.0.0.32.1 uint32_t wwdt_config_t::warningValue

Set this to 0 for no warning

30.4.1.0.0.32.2 uint32_t wwdt_config_t::clockFreq_Hz

30.5 Macro Definition Documentation

30.5.1 #define FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

30.6 Enumeration Type Documentation

30.6.1 enum _wwdt_status_flags_t

This structure contains the WWDT status flags for use in the WWDT functions.

Enumerator

kWWDT_TimeoutFlag Time-out flag, set when the timer times out.
kWWDT_WarningFlag Warning interrupt flag, set when timer is below the value WDWARNINT.

30.7 Function Documentation

30.7.1 void WWDT_GetDefaultConfig (wwdt_config_t * *config*)

This function initializes the WWDT configure structure to default value. The default value are:

```
*   config->enableWwdt = true;
*   config->enableWatchdogReset = false;
*   config->enableWatchdogProtect = false;
*   config->enableLockOscillator = false;
*   config->windowValue = 0xFFFFFFU;
*   config->timeoutValue = 0xFFFFFFU;
*   config->warningValue = 0;
*
```

Parameters

config	Pointer to WWDT config structure.

See Also

wwdt_config_t

30.7.2 void WWDT_Init (WWDT_Type * *base*, const wwdt_config_t * *config*)

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
*    wwdt_config_t config;
*    WWDT_GetDefaultConfig(&config);
*    config.timeoutValue = 0x7ffU;
*    WWDT_Init(wwdt_base,&config);
*
```

Parameters

base	WWDT peripheral base address
config	The configuration of WWDT

30.7.3 void WWDT_Deinit (WWDT_Type * *base*)

This function shuts down the WWDT.

Parameters

base	WWDT peripheral base address

30.7.4 static void WWDT_Enable (WWDT_Type * *base*) [inline], [static]

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

base	WWDT peripheral base address

30.7.5 static void WWDT_Disable (WWDT_Type * *base*) [inline], [static]

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

base	WWDT peripheral base address

30.7.6 static uint32_t WWDT_GetStatusFlags (WWDT_Type * *base*) [inline], [static]

This function gets all status flags.

Example for getting Timeout Flag:

```
*   uint32_t status;
*   status = WWDT_GetStatusFlags(wwdt_base) &
     kWWDT_TimeoutFlag;
*
```

Parameters

base	WWDT peripheral base address

Returns

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

30.7.7 void WWDT_ClearStatusFlags (WWDT_Type * *base*, uint32_t *mask*)

This function clears WWDT status flag.

Example for clearing warning flag:

```
*   WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
*
```

Parameters

base	WWDT peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration _wwdt-_status_flags_t

30.7.8 static void WWDT_SetWarningValue (WWDT_Type * *base*, uint32_t *warningValue*) [inline], [static]

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

base	WWDT peripheral base address
warningValue	WWDT warning value.

30.7.9 static void WWDT_SetTimeoutValue (WWDT_Type * *base*, uint32_t *timeoutCount*) [inline], [static]

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is TWDCLK∗256∗4. If enableWatchdogProtect flag is true in wwdt_config_t config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

base	WWDT peripheral base address
timeoutCount	WWDT timeout value, count of WWDT clock tick.

30.7.10 static void WWDT_SetWindowValue (WWDT_Type * *base*, uint32_t *windowValue*) [inline], [static]

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

base	WWDT peripheral base address
windowValue	WWDT window value.

30.7.11 void WWDT_Refresh (WWDT_Type * *base*)

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

base	WWDT peripheral base address

Chapter 31
CMP: Comparator driver

31.1 Overview

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

Data Structures

- struct cmp_config_t
  cmp configurataions More...

Macros

- #define CMP_INT_POL_SHIFT_VALUE (1U)
  cmp level shift value definition

Enumerations

- enum _cmp_status {
  kCMP_In0BiggerThanIn1 = 1U,
  kCMP_In1BiggerThanIn0 = 0U }
  cmp status
- enum cmp_interrupt_mask_t {
  kCMP_EdgeRising = 0U << CMP_INT_POL_SHIFT_VALUE,
  kCMP_EdgeFalling = 1U << CMP_INT_POL_SHIFT_VALUE,
  kCMP_EdgeRisingFalling = 3U << CMP_INT_POL_SHIFT_VALUE,
  kCMP_LevelLow = (0U << CMP_INT_POL_SHIFT_VALUE) | 1U,
  kCMP_LevelHigh = (2U << CMP_INT_POL_SHIFT_VALUE) | 1U }
  cmp interrupt
- enum cmp_mode_t {
  kCMP_FastMode = 0U,
  kCMP_LowpowerMode = 1U }
  cmp work mode
- enum cmp_input_t {
  kCMP_InputAllExternal = 0U,
  kCMP_InputOneExternalOneInternal }
  cmp input source

Driver version

- #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 1U))
  Driver version 2.0.1.

Cmp Initialization and deinitialization

- void CMP_Init (cmp_config_t *config)
  CMP intialization.
- void CMP_Deinit (void)
  CMP deintialization.

cmp functionality

- static void CMP_SwapExtInput (void)
  Swap the external input channel.
- static void CMP_EnableLowePowerMode (bool enable)
  switch cmp work mode.
- static void CMP_EnableInnerInput (bool enable)
  switch input source.
- static void CMP_EnableLowHysteresis (bool enable)
  cmp enable low hysteresis.
- static uint32_t CMP_GetOutput (void)
  cmp output status.

cmp interrupt

- void CMP_SetInterruptConfig (cmp_interrupt_mask_t mask)
  cmp set interrupt configurations.
- static void CMP_EnableInterrupt (void)
  cmp enable interrupt.
- static void CMP_DisableInterrupt (void)
  cmp disable interrupt.
- static bool CMP_GetStatus (void)
  cmp get status.
- static void CMP_ClearStatus (void)
  cmp clear interrupt status.
- static bool CMP_GetInterruptStatus (void)
  cmp get interrupt status.

31.2 Data Structure Documentation

31.2.1 struct cmp_config_t

Data Fields

- bool enLowHysteris
  low hysteresis
- cmp_input_t src
  input source select
- cmp_mode_t mode
  cmp work mode

31.3 Macro Definition Documentation

31.3.1 #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 1U))

31.4 Enumeration Type Documentation

31.4.1 enum _cmp_status

Enumerator

kCMP_In0BiggerThanIn1 comparator input 0 is bigger than input 1
kCMP_In1BiggerThanIn0 comparator input 1 is bigger than input 0

31.4.2 enum cmp_interrupt_mask_t

Enumerator

kCMP_EdgeRising Edge sensitive, falling edge.
kCMP_EdgeFalling Edge sensitive, rising edge.
kCMP_EdgeRisingFalling Edge sensitive, rising and falling edge.
kCMP_LevelLow Level sensitive, low level.
kCMP_LevelHigh Level sensitive, high level.

31.4.3 enum cmp_mode_t

Enumerator

kCMP_FastMode Used in an active or deep sleep mode, this mode requires PMU bias enabled.
kCMP_LowpowerMode Used for all power mode, doesn't require PMU bias enabled.

31.4.4 enum cmp_input_t

Enumerator

kCMP_InputAllExternal Cmp input from two external source.
kCMP_InputOneExternalOneInternal Cmp input from one external input and one internal voltage reference 0.8V.

31.5 Function Documentation

31.5.1 void CMP_Init (cmp_config_t * *config*)

Note: The cmp initial function not responsible for cmp power, application shall handle it.

Parameters

config	init configurations.

31.5.2 void CMP_Deinit (void)

Note: The cmp deinit function not responsible for cmp power, application shall handle it.

31.5.3 static void CMP_SwapExtInput (void) [inline], [static]

Parameters

base	CMP base address.

31.5.4 static void CMP_EnableLowePowerMode (bool *enable*) [inline], [static]

Parameters

enable	true is enter low power mode, false is enter fast mode

31.5.5 static void CMP_EnableInnerInput (bool *enable*) [inline], [static]

Parameters

enable	true is one external and one internal, false is all external.

31.5.6 static uint32_t CMP_GetOutput (void) [inline], [static]

Returns

0 is kCMP_In1BiggerThanIn0, 1 is kCMP_In0BiggerThanIn1.

31.5.7 void CMP_SetInterruptConfig (cmp_interrupt_mask_t *mask*)

Parameters

mask	interrupt mask.

31.5.8 static bool CMP_GetStatus (void) [inline], [static]

Returns

true is interrupt pending, false is no interrupt pending.

31.5.9 static void CMP_ClearStatus (void) [inline], [static]

Returns

true is interrupt pending, false is no interrupt pending.

31.5.10 static bool CMP_GetInterruptStatus (void) [inline], [static]

Returns

true is interrupt pending, false is no interrupt pending.

Chapter 32
FLASHIAP: Flash In Application Programming Driver

32.1 Overview

The MCUXpresso SDK provides a driver for the Flash In Application Programming (FLASHIAP).

It provides a set of functions to call the on-chip in application flash programming interface. User code executing from on-chip flash or RAM can call these function to erase and write the flash memory.

32.2 GFlash In Application Programming operation

FLASHIAP_PrepareSectorForWrite() prepares a sector for write or erase operation.

FLASHIAP_CopyRamToFlash() function programs the flash memory.

FLASHIAP_EraseSector() function erase a flash sector. A sector must be erased before write operation.

32.3 Typical use case

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

Files

- file fsl_flashiap.h

Typedefs

- typedef void(∗ IAP_ENTRY_T)(uint32_t cmd[5], uint32_t stat[4])
  IAP_ENTRY API function type.

Enumerations

- enum _flashiap_status {
kStatus_FLASHIAP_Success = kStatus_Success,
kStatus_FLASHIAP_InvalidCommand = MAKE_STATUS(kStatusGroup_FLASHIAP, 1U),
kStatus_FLASHIAP_SrcAddrError,
kStatus_FLASHIAP_DstAddrError,
kStatus_FLASHIAP_SrcAddrNotMapped,
kStatus_FLASHIAP_DstAddrNotMapped,
kStatus_FLASHIAP_CountError,
kStatus_FLASHIAP_InvalidSector,
kStatus_FLASHIAP_SectorNotblank = MAKE_STATUS(kStatusGroup_FLASHIAP, 8U),
kStatus_FLASHIAP_NotPrepared,
kStatus_FLASHIAP_CompareError,
kStatus_FLASHIAP_Busy,
kStatus_FLASHIAP_ParamError,
kStatus_FLASHIAP_AddrError = MAKE_STATUS(kStatusGroup_FLASHIAP, 13U),
kStatus_FLASHIAP_AddrNotMapped,
kStatus_FLASHIAP_NoPower = MAKE_STATUS(kStatusGroup_FLASHIAP, 24U),
kStatus_FLASHIAP_NoClock }
Flashiap status codes.
- enum _flashiap_commands {
kIapCmd_FLASHIAP_PrepareSectorforWrite = 50U,
kIapCmd_FLASHIAP_CopyRamToFlash = 51U,
kIapCmd_FLASHIAP_EraseSector = 52U,
kIapCmd_FLASHIAP_BlankCheckSector = 53U,
kIapCmd_FLASHIAP_ReadPartId = 54U,
kIapCmd_FLASHIAP_Read_BootromVersion = 55U,
kIapCmd_FLASHIAP_Compare = 56U,
kIapCmd_FLASHIAP_ReinvokeISP = 57U,
kIapCmd_FLASHIAP_ReadUid = 58U,
kIapCmd_FLASHIAP_ErasePage = 59U,
kIapCmd_FLASHIAP_ReadMisr = 70U,
kIapCmd_FLASHIAP_ReinvokeI2cSpiISP = 71U }
Flashiap command codes.

Functions

- static void iap_entry (uint32_t ∗cmd_param, uint32_t ∗status_result)
IAP_ENTRY API function type.
- status_t FLASHIAP_PrepareSectorForWrite (uint32_t startSector, uint32_t endSector)
Prepare sector for write operation.
- status_t FLASHIAP_CopyRamToFlash (uint32_t dstAddr, uint32_t ∗srcAddr, uint32_t numOfBytes, uint32_t systemCoreClock)
Copy RAM to flash.
- status_t FLASHIAP_EraseSector (uint32_t startSector, uint32_t endSector, uint32_t systemCoreClock)

Erase sector.
- status_t FLASHIAP_ErasePage (uint32_t startPage, uint32_t endPage, uint32_t systemCoreClock)
  This function erases page(s).
- status_t FLASHIAP_BlankCheckSector (uint32_t startSector, uint32_t endSector)
  Blank check sector(s)
- status_t FLASHIAP_Compare (uint32_t dstAddr, uint32_t *srcAddr, uint32_t numOfBytes)
  Compare memory contents of flash with ram.

Driver version

- #define FSL_FLASHIAP_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  Version 2.0.3.

32.4 Macro Definition Documentation

32.4.1 #define FSL_FLASHIAP_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

32.5 Enumeration Type Documentation

32.5.1 enum _flashiap_status

Enumerator

kStatus_FLASHIAP_Success Api is executed successfully.
kStatus_FLASHIAP_InvalidCommand Invalid command.
kStatus_FLASHIAP_SrcAddrError Source address is not on word boundary.
kStatus_FLASHIAP_DstAddrError Destination address is not on a correct boundary.
kStatus_FLASHIAP_SrcAddrNotMapped Source address is not mapped in the memory map.
kStatus_FLASHIAP_DstAddrNotMapped Destination address is not mapped in the memory map.
kStatus_FLASHIAP_CountError Byte count is not multiple of 4 or is not a permitted value.
kStatus_FLASHIAP_InvalidSector Sector number is invalid or end sector number is greater than start sector number.
kStatus_FLASHIAP_SectorNotblank One or more sectors are not blank.
kStatus_FLASHIAP_NotPrepared Command to prepare sector for write operation was not executed.
kStatus_FLASHIAP_CompareError Destination and source memory contents do not match.
kStatus_FLASHIAP_Busy Flash programming hardware interface is busy.
kStatus_FLASHIAP_ParamError Insufficient number of parameters or invalid parameter.
kStatus_FLASHIAP_AddrError Address is not on word boundary.
kStatus_FLASHIAP_AddrNotMapped Address is not mapped in the memory map.
kStatus_FLASHIAP_NoPower Flash memory block is powered down.
kStatus_FLASHIAP_NoClock Flash memory block or controller is not clocked.

32.5.2 enum _flashiap_commands

Enumerator

kIapCmd_FLASHIAP_PrepareSectorforWrite Prepare Sector for write.
kIapCmd_FLASHIAP_CopyRamToFlash Copy RAM to flash.
kIapCmd_FLASHIAP_EraseSector Erase Sector.
kIapCmd_FLASHIAP_BlankCheckSector Blank check sector.
kIapCmd_FLASHIAP_ReadPartId Read part id.
kIapCmd_FLASHIAP_Read_BootromVersion Read bootrom version.
kIapCmd_FLASHIAP_Compare Compare.
kIapCmd_FLASHIAP_ReinvokeISP Reinvoke ISP.
kIapCmd_FLASHIAP_ReadUid Read Uid isp.
kIapCmd_FLASHIAP_ErasePage Erase Page.
kIapCmd_FLASHIAP_ReadMisr Read Misr.
kIapCmd_FLASHIAP_ReinvokeI2cSpiISP Reinvoke I2C/SPI isp.

32.6 Function Documentation

32.6.1 static void iap_entry (uint32_t * *cmd_param,* uint32_t * *status_result*) [inline], [static]

Wrapper for rom iap call

Parameters

cmd_param	IAP command and relevant parameter array.
status_result	IAP status result array.

Return values

None.	Status/Result is returned via status_result array.

32.6.2 status_t FLASHIAP_PrepareSectorForWrite (uint32_t *startSector,* uint32_t *endSector*)

This function prepares sector(s) for write/erase operation. This function must be called before calling the FLASHIAP_CopyRamToFlash() or FLASHIAP_EraseSector() or FLASHIAP_ErasePage() function. The end sector must be greater than or equal to start sector number.

Parameters

startSector	Start sector number.
endSector	End sector number.

Return values

kStatus_FLASHIAP_-Success	Api was executed successfully.
kStatus_FLASHIAP_No-Power	Flash memory block is powered down.
kStatus_FLASHIAP_No-Clock	Flash memory block or controller is not clocked.
kStatus_FLASHIAP_-InvalidSector	Sector number is invalid or end sector number is greater than start sector number.
kStatus_FLASHIAP_Busy	Flash programming hardware interface is busy.

32.6.3 status_t FLASHIAP_CopyRamToFlash (uint32_t *dstAddr,* uint32_t ∗ *srcAddr,* uint32_t *numOfBytes,* uint32_t *systemCoreClock*)

This function programs the flash memory. Corresponding sectors must be prepared via FLASHIAP_-PrepareSectorForWrite before calling calling this function. The addresses should be a 256 byte boundary and the number of bytes should be 256 | 512 | 1024 | 4096.

Parameters

dstAddr	Destination flash address where data bytes are to be written.
srcAddr	Source ram address from where data bytes are to be read.
numOfBytes	Number of bytes to be written.
systemCore-Clock	SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function.

Return values

kStatus_FLASHIAP_-Success	Api was executed successfully.

kStatus_FLASHIAP_NoPower	Flash memory block is powered down.
kStatus_FLASHIAP_NoClock	Flash memory block or controller is not clocked.
kStatus_FLASHIAP_SrcAddrError	Source address is not on word boundary.
kStatus_FLASHIAP_DstAddrError	Destination address is not on a correct boundary.
kStatus_FLASHIAP_SrcAddrNotMapped	Source address is not mapped in the memory map.
kStatus_FLASHIAP_DstAddrNotMapped	Destination address is not mapped in the memory map.
kStatus_FLASHIAP_CountError	Byte count is not multiple of 4 or is not a permitted value.
kStatus_FLASHIAP_NotPrepared	Command to prepare sector for write operation was not executed.
kStatus_FLASHIAP_Busy	Flash programming hardware interface is busy.

32.6.4 status_t FLASHIAP_EraseSector (uint32_t *startSector*, uint32_t *endSector*, uint32_t *systemCoreClock*)

This function erases sector(s). The end sector must be greater than or equal to start sector number. FLASHIAP_PrepareSectorForWrite must be called before calling this function.

Parameters

startSector	Start sector number.
endSector	End sector number.
systemCoreClock	SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function.

Return values

kStatus_FLASHIAP_Success	Api was executed successfully.

kStatus_FLASHIAP_NoPower	Flash memory block is powered down.
kStatus_FLASHIAP_NoClock	Flash memory block or controller is not clocked.
kStatus_FLASHIAP_InvalidSector	Sector number is invalid or end sector number is greater than start sector number.
kStatus_FLASHIAP_NotPrepared	Command to prepare sector for write operation was not executed.
kStatus_FLASHIAP_Busy	Flash programming hardware interface is busy.

32.6.5 status_t FLASHIAP_ErasePage (uint32_t *startPage,* uint32_t *endPage,* uint32_t *systemCoreClock*)

The end page must be greater than or equal to start page number. Corresponding sectors must be prepared via FLASHIAP_PrepareSectorForWrite before calling calling this function.

Parameters

startPage	Start page number
endPage	End page number
systemCoreClock	SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function.

Return values

kStatus_FLASHIAP_Success	Api was executed successfully.
kStatus_FLASHIAP_NoPower	Flash memory block is powered down.
kStatus_FLASHIAP_NoClock	Flash memory block or controller is not clocked.
kStatus_FLASHIAP_InvalidSector	Page number is invalid or end page number is greater than start page number

kStatus_FLASHIAP_Not-Prepared	Command to prepare sector for write operation was not executed.
kStatus_FLASHIAP_Busy	Flash programming hardware interface is busy.

32.6.6 status_t FLASHIAP_BlankCheckSector (uint32_t *startSector,* uint32_t *endSector*)

Blank check single or multiples sectors of flash memory. The end sector must be greater than or equal to start sector number. It can be used to verify the sector eraseure after FLASHIAP_EraseSector call.

Parameters

startSector	: Start sector number. Must be greater than or equal to start sector number
endSector	: End sector number

Return values

kStatus_FLASHIAP_-Success	One or more sectors are in erased state.
kStatus_FLASHIAP_No-Power	Flash memory block is powered down.
kStatus_FLASHIAP_No-Clock	Flash memory block or controller is not clocked.
kStatus_FLASHIAP_-SectorNotblank	One or more sectors are not blank.

32.6.7 status_t FLASHIAP_Compare (uint32_t *dstAddr,* uint32_t ∗ *srcAddr,* uint32_t *numOfBytes*)

This function compares the contents of flash and ram. It can be used to verify the flash memory contents after FLASHIAP_CopyRamToFlash call.

Parameters

dstAddr	Destination flash address.

srcAddr	Source ram address.
numOfBytes	Number of bytes to be compared.

Return values

kStatus_FLASHIAP_-Success	Contents of flash and ram match.
kStatus_FLASHIAP_No-Power	Flash memory block is powered down.
kStatus_FLASHIAP_No-Clock	Flash memory block or controller is not clocked.
kStatus_FLASHIAP_-AddrError	Address is not on word boundary.
kStatus_FLASHIAP_-AddrNotMapped	Address is not mapped in the memory map.
kStatus_FLASHIAP_-CountError	Byte count is not multiple of 4 or is not a permitted value.
kStatus_FLASHIAP_-CompareError	Destination and source memory contents do not match.

Chapter 33
SHA: SHA encryption decryption driver

33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the SHA module in MCUXpresso SDK devices.

The driver provides blocking synchronous APIs. The SHA operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions do not return until an S-HA operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

33.2 SHA Driver Initialization and Configuration

Clock to the SHA module has to be enabled before using the driver API.

33.3 Comments about API usage in RTOS

SHA operations provided by this driver are not re-entrant. Therefore, the application software should ensure the SHA module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

33.4 SHA Driver Example

Typical use case Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOAR-D>/driver_examples/sha

Modules

- Sha_algorithm_level_api

Files

- file fsl_sha.h

Data Structures

- struct sha_ctx_t
  Storage type used to save hash context. More...

Macros

- #define SHA_CTX_SIZE 20
  SHA Context size.

Enumerations

- enum sha_algo_t {
  kSHA_Sha1,
  kSHA_Sha256 }
  Supported cryptographic block cipher functions for HASH creation.

Driver version

- #define FSL_SHA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  Defines LPC SHA driver version 2.1.0.

33.5 Data Structure Documentation

33.5.1 struct sha_ctx_t

33.6 Macro Definition Documentation

33.6.1 #define FSL_SHA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

33.6.2 #define SHA_CTX_SIZE 20

33.7 Enumeration Type Documentation

33.7.1 enum sha_algo_t

Enumerator

kSHA_Sha1 SHA_1.
kSHA_Sha256 SHA_256.

Chapter 34
Serial Manager

34.1 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, USB CDC 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_None = 0U,
kSerialPort_Uart = 1U,
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.

34.2 Data Structure Documentation

34.2.1 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.

34.2.1.0.0.33 Field Documentation

34.2.1.0.0.33.1 uint8_t∗ serial_manager_config_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

34.2.2 struct serial_manager_callback_message_t

Data Fields

- uint8_t ∗ buffer
  Transferred buffer.
- uint32_t length
  Transferred data length.

34.3 Enumeration Type Documentation

34.3.1 enum serial_port_type_t

Enumerator

kSerialPort_None Serial port is none.
kSerialPort_Uart Serial port UART.
kSerialPort_Uart Serial port UART.
kSerialPort_UsbCdc Serial port USB CDC.
kSerialPort_Swo Serial port SWO.
kSerialPort_UsbCdcVirtual Serial port USB CDC Virtual.

34.3.2 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.

34.4 Function Documentation

34.4.1 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.

34.4.2 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.

34.4.3 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.

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);
*
```

34.4.4 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.

34.4.5 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));
*
```

34.4.6 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.

34.4.7 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.

34.4.8 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.

34.4.9 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.

34.4.10 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.

34.5 Serial Port Uart

34.5.1 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

34.5.2 Data Structure Documentation

34.5.2.1 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.

34.5.2.1.0.34 Field Documentation

34.5.2.1.0.34.1 uint8_t serial_port_uart_config_t::instance

34.5.3 Enumeration Type Documentation

34.5.3.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

34.5.3.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

34.6 Serial Port USB

34.6.1 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.

34.6.2 Data Structure Documentation

34.6.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- serial_port_usb_cdc_controller_index_t controllerIndex
  controller index

34.6.3 Enumeration Type Documentation

34.6.3.1 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.

34.6.4 USB Device Configuration

34.6.4.1 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.

34.6.4.2 Macro Definition Documentation

34.6.4.2.1 #define USB_DEVICE_CONFIG_SELF_POWER (1U)

1U supported, 0U not supported

34.6.4.2.2 #define USB_DEVICE_CONFIG_ENDPOINTS (4U)

34.6.4.2.3 #define USB_DEVICE_CONFIG_USE_TASK (0U)

34.6.4.2.4 #define USB_DEVICE_CONFIG_MAX_MESSAGES (8U)

34.6.4.2.5 #define USB_DEVICE_CONFIG_USB20_TEST_MODE (0U)

34.6.4.2.6 #define USB_DEVICE_CONFIG_CV_TEST (0U)

34.6.4.2.7 #define USB_DEVICE_CONFIG_COMPLIANCE_TEST (0U)

If the macro is enabled, the test mode and CV test macroes will be set.

34.6.4.2.8 #define USB_DEVICE_CONFIG_KEEP_ALIVE_MODE (0U)

34.6.4.2.9 #define USB_DEVICE_CONFIG_BUFFER_PROPERTY_CACHEABLE (0U)

34.6.4.2.10 #define USB_DEVICE_CONFIG_LOW_POWER_MODE (0U)

34.6.4.2.11 #define USB_DEVICE_CONFIG_REMOTE_WAKEUP (0U)

34.6.4.2.12 #define USB_DEVICE_CONFIG_DETACH_ENABLE (0U)

34.6.4.2.13 #define USB_DEVICE_CONFIG_ERROR_HANDLING (0U)

34.6.4.2.14 #define USB_DEVICE_CHARGER_DETECT_ENABLE (0U)

34.7 Serial Port SWO

34.7.1 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

34.7.2 Data Structure Documentation

34.7.2.1 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.

34.7.3 Enumeration Type Documentation

34.7.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

34.8 Serial Port Virtual USB

34.8.1 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 USB 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::controllerIndex
controller index

34.8.2 Data Structure Documentation

34.8.2.1 struct serial_port_usb_cdc_virtual_config_t

Data Fields

- serial_port_usb_cdc_virtual_controller_index_t controllerIndex
controller index

34.8.3 Enumeration Type Documentation

34.8.3.1 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 35
NTAG: integrated NTAG

35.1 Overview

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

Data Structures

- struct ntag_config_t
  NTAG user configuration. More...

Macros

- #define RFT1503
  FD polling implementation options.
- #define NTAG_IRQ NFCTag_IRQn
  NTAG FD interrupt line.

Typedefs

- typedef void(∗ ntag_field_detect_callback_t)(ntag_field_detect_t fd, void ∗userData)
  NTAG Field Detect callback typedef.

Enumerations

- enum ntag_state_t {
  kNTAG_StateActive,
  kNTAG_StateInactive }
  ntag operating state
- enum ntag_field_detect_t {
  kNTAG_FieldDetectIn,
  kNTAG_FieldDetectOut }
  Field Detect line state.

Functions

- void NTAG_GetDefaultConfig (ntag_config_t ∗config)
  Sets the NTAG configuration structure to default values.
- void NTAG_Init (const ntag_config_t ∗config)
  Initialize the internal NTAG peripheral.
- ntag_field_detect_t NTAG_PollFieldDetect (void)
  Poll state of Field Detect line.
- void NTAG_SetState (ntag_state_t state)
  Configure NTAG operating state.

35.2 Data Structure Documentation

35.2.1 struct ntag_config_t

Data Fields

- ntag_field_detect_callback_t callback
  A callback function called at the transfer event.
- void * userData
  A callback parameter passed to the callback function.

35.2.1.0.0.1 Field Documentation

35.2.1.0.0.1.1 ntag_field_detect_callback_t ntag_config_t::callback

35.2.1.0.0.1.2 void* ntag_config_t::userData

35.3 Macro Definition Documentation

35.3.1 #define RFT1503

35.4 Typedef Documentation

35.4.1 typedef void(* ntag_field_detect_callback_t)(ntag_field_detect_t fd, void *userData)

35.5 Enumeration Type Documentation

35.5.1 enum ntag_state_t

Enumerator

kNTAG_StateActive NTAG powered on (ready for I2C communication)
kNTAG_StateInactive NTAG powered off or MCU in low power state.

35.5.2 enum ntag_field_detect_t

Enumerator

kNTAG_FieldDetectIn NTAG is in field.
kNTAG_FieldDetectOut NTAG is out of field.

35.6 Function Documentation

35.6.1 void NTAG_GetDefaultConfig (ntag_config_t * *config*)

Parameters

config	A pointer to the configuration structure.

35.6.2 void NTAG_Init (const ntag_config_t * *config*)

Parameters

config	A pointer to the NTAG configuration structure

35.6.3 ntag_field_detect_t NTAG_PollFieldDetect (void)

35.6.4 void NTAG_SetState (ntag_state_t *state*)

Parameters

state	NTAG operating state

Chapter 36
Wtimer

36.1 Overview

Files

- file fsl_wtimer.h

Driver version

Wake timers provide wakeup capabilities in sleep modes where 32KHz clock is kept active. Wake timer 0 is a 48bit based counter while wake timer 1 is 32bit based counter. A special API functions WTIMER_StartTimerLarge(0 and WTIMER_StartTimerLarge() are provided to access the 48bit counter. The Wake timer 1 is to be used bu the PWRM framework. It shall not be used by the Application directly. API provides the capability to enable and disable interrupts. The application shall implement the Wake timer ISR on its side. The wake timer ISR prototypes are : void WAKE_UP_TIMER0_IRQHandler(void); and void WAKE_UP_TIMER1_IRQHandler(void); The Application shall correctly the 32KHz source amoung the FRO32 or Crystal 32KHz using CLOCK_EnableClock() API in fsl_clock.h The APi provides the capability to calibrate the 32KHz clock versus a high reference clock (32MHz crystal).

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

Initialization and deinitialization

- void WTIMER_Init (void)
  Enable the clocks to the peripheral (functional clock and AHB clock)
- void WTIMER_DeInit (void)
  Disable the clocks to the peripheral (functional clock and AHB clock)
- void WTIMER_EnableInterrupts (WTIMER_timer_id_t timer_id)
  Enable the selected Timer interrupts.
- WTIMER_status_t WTIMER_GetStatusFlags (WTIMER_timer_id_t timer_id)
  Gets the Timer status flags.
- void WTIMER_ClearStatusFlags (WTIMER_timer_id_t timer_id)
  Clears the Timer status flags if expired and clear the pendng interrupt if active it needs to be called in ISR.
- void WTIMER_StartTimer (WTIMER_timer_id_t timer_id, uint32_t count)
  Starts the Timer counter.
- void WTIMER_StopTimer (WTIMER_timer_id_t timer_id)
  Stops the Timer counter.
- uint32_t WTIMER_CalibrateTimer (void)
  Calibrate the 32KHz clock to be used by the wake timer versus the 32MHz crystal clock source The Applicaton shall switches OFF the 32MHz clock if no longer used by the chip using CLOCK_DisableClock() in fsl_clock.h.
- uint32_t WTIMER_ReadTimer (WTIMER_timer_id_t timer_id)
  Read the LSB counter of the wake timer This API is unsafe.

- uint32_t WTIMER_ReadTimerSafe (WTIMER_timer_id_t timer_id)
  Read the LSB counter of the wake timer API checks the next counter update (next 32KHz clock edge) so the value is uptodate Important note : The counter shall be running otherwise, the API gets locked and never return.

36.2 Function Documentation

36.2.1 void WTIMER_Init (void)

Note

This function does not reset the wake timer peripheral. Wake timer reset is done in PWRM_vColdStart() from the PWRM framework module if integrated If PWRM framework module is integrated, WTIMER_Init() is called in PWRM_vInit() for power modes with Oscillator ON.

36.2.2 void WTIMER_DeInit (void)

Note

This function does not reset the wake timer peripheral.

36.2.3 void WTIMER_EnableInterrupts (WTIMER_timer_id_t *timer_id*)

The application shall implement the Wake timer ISR

Parameters

timer_id	Wtimer Id

36.2.4 WTIMER_status_t WTIMER_GetStatusFlags (WTIMER_timer_id_t *timer_id*)

Parameters

timer_id	Wtimer Id

Returns

The status flags.

36.2.5 void WTIMER_ClearStatusFlags (WTIMER_timer_id_t *timer_id*)

Parameters

timer_id	Wtimer Id

36.2.6 void WTIMER_StartTimer (WTIMER_timer_id_t *timer_id*, uint32_t *count*)

The function performs: -stop the timer if running, clear the status and interrupt flag if set (WTIMER_-ClearStatusFlags()) -set the counter value -start the timer

Parameters

timer_id	Wtimer Id
count	number of 32KHz clock periods before expiration

36.2.7 void WTIMER_StopTimer (WTIMER_timer_id_t *timer_id*)

Parameters

timer_id	Wtimer Id

36.2.8 uint32_t WTIMER_CalibrateTimer (void)

Returns

32KHz clock frequency (number of 32KHz clock in one sec) - expect to have 32768

36.2.9 uint32_t WTIMER_ReadTimer (WTIMER_timer_id_t *timer_id*)

If the counter has just been started, the counter value may not be up to date until the next 32KHz clock edge. Use WTIMER_ReadTimerSafe() instead

Parameters

timer_id	Wtimer Id

Returns

counter value - number of ticks before expiration if running

36.2.10 uint32_t WTIMER_ReadTimerSafe (WTIMER_timer_id_t *timer_id*)

Parameters

timer_id	Wtimer Id

Returns

32KHz clock frequency (number of 32KHz clock in one sec) - expect to have 32768

Chapter 37
ROM_API

37.1 Overview

Files

- file rom_aes.h
- file rom_api.h
- file rom_isp.h
- file rom_lowpower.h
- file rom_mpu.h
- file rom_pmc.h
- file rom_psector.h
- file rom_secure.h

Data Structures

- struct IMAGE_DATA_T
  IMAGE_DATA_T image node : element of single link chained list of images found in flash. More...
- struct ISP_MEM_FUNC_T
  ISP_MEM_FUNC_T structure of ops method pointers instantiated per memory type. More...
- struct ISP_MEM_INFO_T
  ISP_MEM_INFO_T structure of memory characteristics. More...
- struct ISP_ENC_STATE_T
  ISP_ENC_STATE_T ISP structure for ciphering options : TODO check poorly tested should we advertise this ? More...
- struct ISP_STATE_T
  ISP_STATE_T structure holding the context the the curent ISP command. More...
- struct LPC_LOWPOWER_T
  Low Power Main Structure. More...
- struct LPC_LOWPOWER_LDOVOLTAGE_T
  Low Power Main Structure. More...
- struct MPU_Settings_t
  MPU_Settings_t structure in RAM to retrieve current MPU configuration see . More...
- struct image_directory_entry_t
  image_directory_entry_t image directory found in PAGE0 (PSECT) when SSBL is involved in the loading process More...
- struct psector_header_t
  psector_header_t psector header. More...
- struct IMAGE_CERT_T
  IMAGE_CERT_T structure. More...

Macros

- #define AES_INB_FSEL(n) ((n) << 16)
  n->1=Input Text, n->2=Holding, n->3=Input Text XOR Holding
- #define AES_HOLD_FSEL(n) ((n) << 20)

n->0=Counter, n->1=Input Text, n->2=Output Block, n->3=Input Text XOR Output Block
- #define AES_OUTT_FSEL(n) ((n) << 24)
  n->0=OUTT, n->1=Output Block XOR Input Text, n->2=Output Block XOR Holding
- #define ISP_INVALID_EXTENSION (0)
  Each ISP extension function invalid : 0 corresponds to a NULL pointer.
- #define ISP_FLAG_HAS_CRC32 (1 << 0)
  Each ISP command is preceded by a 'flag' byte that tell how to verify the command.
- #define ISP_FLAG_SIGNED (1 << 1)
  tells that command is RSA signed and authentication is checked, if unset, the SHA256 is computed and compared against the one held in the message, which guarantees integrity
- #define ISP_FLAG_HAS_NEXT_HASH (1 << 2)
  tells to hold the computed hash
- #define LOWPOWER_CFG_MODE_ACTIVE 0
  ACTIVE mode.
- #define LOWPOWER_CFG_MODE_DEEPSLEEP 1
  DEEP SLEEP mode.
- #define LOWPOWER_CFG_MODE_POWERDOWN 2
  POWER DOWN mode.
- #define LOWPOWER_CFG_MODE_DEEPPOWERDOWN 3
  DEEP POWER DOWN mode.
- #define LOWPOWER_CFG_XTAL32MSTART_DISABLE 0
  Disable Crystal 32 MHz automatic start when waking up from POWER DOWN and DEEP POWER DOWN modes.
- #define LOWPOWER_CFG_XTAL32MSTART_ENABLE 1
  Enable Crystal 32 MHz automatic start when waking up from POWER DOWN and DEEP POWER DOWN modes.
- #define LOWPOWER_CFG_FLASHPWDNMODE_FLASHPWND 0
  Power down the Flash only (send CMD_POWERDOWN to Flash controller).
- #define LOWPOWER_CFG_FLASHPWDNMODE_LDOSHUTOFF 1
  Power down the Flash ((send CMD_POWERDOWN to Flash controller) and shutoff both Flash LDOs (Core and NV) \\ (only valid in DEEP SLEEP mode)
- #define LOWPOWER_PMUPWDN_DCDC (1UL << 0)
  Analog Power Domains (analog components in Power Management Unit) Low Power Modes control.
- #define LOWPOWER_PMUPWDN_BIAS (1UL << 1)
  Power Down all Bias and references.
- #define LOWPOWER_PMUPWDN_LDOMEM (1UL << 2)
  Power Down Memories LDO.
- #define LOWPOWER_PMUPWDN_BODVBAT (1UL << 3)
  Power Down VBAT Brown Out Detector.
- #define LOWPOWER_PMUPWDN_FRO192M (1UL << 4)
  Power Down FRO 192 MHz.
- #define LOWPOWER_PMUPWDN_FRO1M (1UL << 5)
  Power Down FRO 1 MHz.
- #define LOWPOWER_PMUPWDN_GPADC (1UL << 22)
  Power Down General Purpose ADC.
- #define LOWPOWER_PMUPWDN_BODMEM (1UL << 23)
  Power Down Memories Brown Out Detector.
- #define LOWPOWER_PMUPWDN_BODCORE (1UL << 24)
  Power Down Core Logic Brown Out Detector.
- #define LOWPOWER_PMUPWDN_FRO32K (1UL << 25)
  Power Down FRO 32 KHz.

- #define LOWPOWER_PMUPWDN_XTAL32K (1UL << 26)
  Power Down Crystal 32 KHz.
- #define LOWPOWER_PMUPWDN_ANACOMP (1UL << 27)
  Power Down Analog Comparator.
- #define LOWPOWER_PMUPWDN_XTAL32M (1UL << 28)
  Power Down Crystal 32 MHz.
- #define LOWPOWER_PMUPWDN_TEMPSENSOR (1UL << 29)
  Power Down Temperature Sensor.
- #define LOWPOWER_DIGPWDN_FLASH (1UL << 6)
  Digital Power Domains Low Power Modes control.
- #define LOWPOWER_DIGPWDN_COMM0 (1UL << 7)
  Power Down Digital COMM0 power domain (USART0, I2C0 and SPI0)
- #define LOWPOWER_DIGPWDN_MCU_RET (1UL << 8)
  Power Down MCU Retention Power Domain (Disable Zigbee IP retention, ES1:Disable CPU retention \\ flip-flops)
- #define LOWPOWER_DIGPWDN_ZIGBLE_RET (1UL << 9)
  Power Down ZIGBEE/BLE retention Power Domain (Disable ZIGBEE/BLE retention flip-flops)
- #define LOWPOWER_DIGPWDN_SRAM0 (1UL << LOWPOWER_DIGPWDN_SRAM0_INDEX)
  Power Down SRAM 0 instance [Bank 0, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM1 (1UL << 11)
  Power Down SRAM 1 instance [Bank 0, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM2 (1UL << 12)
  Power Down SRAM 2 instance [Bank 0, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM3 (1UL << 13)
  Power Down SRAM 3 instance [Bank 0, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM4 (1UL << 14)
  Power Down SRAM 4 instance [Bank 0, 8 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM5 (1UL << 15)
  Power Down SRAM 5 instance [Bank 0, 8 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM6 (1UL << 16)
  Power Down SRAM 6 instance [Bank 0, 4 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM7 (1UL << 17)
  Power Down SRAM 7 instance [Bank 0, 4 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM8 (1UL << 18)
  Power Down SRAM 8 instance [Bank 1, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM9 (1UL << 19)
  Power Down SRAM 9 instance [Bank 1, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM10 (1UL << 20)
  Power Down SRAM 10 instance [Bank 1, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_SRAM11 (1UL << 21)
  Power Down SRAM 11 instance [Bank 1, 16 KB], (no retention)
- #define LOWPOWER_DIGPWDN_IO (1UL << LOWPOWER_DIGPWDN_IO_INDEX)
  Power Down.
- #define LOWPOWER_DIGPWDN_NTAG_FD (1UL << LOWPOWER_DIGPWDN_NTAG_FD_INDEX)
  NTAG FD field detect Disable - need the IO source to be set too.
- #define LOWPOWER_SRAM_LPMODE_MASK (0xFUL)
  LDO Voltage control in Low Power Modes.
- #define LOWPOWER_VOLTAGE_LDO_PMU_INDEX 0
  LDO Voltage control in Low Power Modes.

- #define LOWPOWER_WAKEUPSRCINT0_SYSTEM_IRQ (1UL << 0)
  Low Power Modes Wake up Interrupt sources.
- #define LOWPOWER_WAKEUPSRCINT0_DMA_IRQ (1UL << 1)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_GINT_IRQ (1UL << 2)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_IRBLASTER_IRQ (1UL << 3)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PINT0_IRQ (1UL << 4)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PINT1_IRQ (1UL << 5)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PINT2_IRQ (1UL << 6)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PINT3_IRQ (1UL << 7)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_SPIFI_IRQ (1UL << 8)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_TIMER0_IRQ (1UL << 9)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_TIMER1_IRQ (1UL << 10)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_USART0_IRQ (1UL << 11)
  [DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT0_USART1_IRQ (1UL << 12)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_I2C0_IRQ (1UL << 13)
  [DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT0_I2C1_IRQ (1UL << 14)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_SPI0_IRQ (1UL << 15)
  [DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT0_SPI1_IRQ (1UL << 16)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM0_IRQ (1UL << 17)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM1_IRQ (1UL << 18)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM2_IRQ (1UL << 19)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM3_IRQ (1UL << 20)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM4_IRQ (1UL << 21)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM5_IRQ (1UL << 22)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM6_IRQ (1UL << 23)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM7_IRQ (1UL << 24)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM8_IRQ (1UL << 25)

[DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM9_IRQ (1UL << 26)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_PWM10_IRQ (1UL << 27)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_I2C2_IRQ (1UL << 28)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_RTC_IRQ (1UL << 29)
  [DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT0_NFCTAG_IRQ (1UL << 30)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT0_MAILBOX_IRQ (1UL << 31)
  Mailbox, Wake-up from DEEP SLEEP and POWER DOWN low power mode [DEEP SLEEP, POWER DOWN].
- #define LOWPOWER_WAKEUPSRCINT1_ADC_SEQA_IRQ (1UL << 0)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_ADC_SEQB_IRQ (1UL << 1)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_ADC_THCMP_OVR_IRQ (1UL << 2)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_DMIC_IRQ (1UL << 3)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_HWVAD_IRQ (1UL << 4)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_BLE_DP_IRQ (1UL << 5)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_BLE_DP0_IRQ (1UL << 6)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_BLE_DP1_IRQ (1UL << 7)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_BLE_DP2_IRQ (1UL << 8)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_BLE_LL_ALL_IRQ (1UL << 9)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_ZIGBEE_MAC_IRQ (1UL << 10)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_ZIGBEE_MODEM_IRQ (1UL << 11)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_RFP_TMU_IRQ (1UL << 12)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_RFP_AGC_IRQ (1UL << 13)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_ISO7816_IRQ (1UL << 14)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_ANA_COMP_IRQ (1UL << 15)
  [DEEP SLEEP]
- #define LOWPOWER_WAKEUPSRCINT1_WAKE_UP_TIMER0_IRQ (1UL << 16)
  [DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT1_WAKE_UP_TIMER1_IRQ (1UL << 17)
  [DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT1_BLE_WAKE_TIMER_IRQ (1UL << 22)

[DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT1_BLE_OSC_EN_IRQ (1UL << 23)
  [DEEP SLEEP, POWER DOWN]
- #define LOWPOWER_WAKEUPSRCINT1_IO_IRQ (1UL << 31)
  [POWER DOWN, DEEP DOWN]
- #define LOWPOWER_SLEEPPOSTPONE_FORCED (1UL << 0)
  Sleep Postpone.
- #define LOWPOWER_SLEEPPOSTPONE_PERIPHERALS (1UL << 1)
  USART0, USART1, SPI0, SPI1, I2C0, I2C1, I2C2 interrupts can postpone power down modes in case an \\ interrupt is pending when the processor request low power mode.
- #define LOWPOWER_SLEEPPOSTPONE_DMIC (1UL << 0)
  DMIC interrupt can postpone power down modes in case an interrupt is pending when the processor \\ request low power mode.
- #define LOWPOWER_SLEEPPOSTPONE_SDMA (1UL << 1)
  System DMA interrupt can postpone power down modes in case an interrupt is pending when the \\ processor request low power mode.
- #define LOWPOWER_SLEEPPOSTPONE_NFCTAG (1UL << 0)
  NFC Tag interrupt can postpone power down modes in case an interrupt is pending when the \\ processor request low power mode.
- #define LOWPOWER_SLEEPPOSTPONE_BLEOSC (1UL << 1)
  BLE_OSC_EN interrupt can postpone power down modes in case an interrupt is pending when the \\ processor request low power mode.
- #define LOWPOWER_WAKEUPIOSRC_PIO0 (1UL << 0)
  Wake up I/O sources.
- #define LOWPOWER_GPIOLATCH_PIO0 (1UL << 0)
  I/O whose state must be kept in Power Down mode.
- #define LOWPOWER_TIMERCFG_ENABLE_INDEX 0
  Wake up timers configuration in Low Power Modes.
- #define LOWPOWER_TIMERCFG_TIMER_ENABLE 1
  Wake Timer Enable.
- #define LOWPOWER_TIMERCFG_TIMER_WAKEUPTIMER0 0
  Primary Wake up timers configuration in Low Power Modes.
- #define LOWPOWER_TIMERCFG_TIMER_WAKEUPTIMER1 1
  Zigbee Wake up Counter 1 used as wake up source.
- #define LOWPOWER_TIMERCFG_TIMER_BLEWAKEUPTIMER 2
  BLE Wake up Counter used as wake up source.
- #define LOWPOWER_TIMERCFG_TIMER_RTC1KHZ 3
  1 KHz Real Time Counter (RTC) used as wake up source
- #define LOWPOWER_TIMERCFG_TIMER_RTC1HZ 4
  1 Hz Real Time Counter (RTC) used as wake up source
- #define LOWPOWER_TIMERCFG_2ND_TIMER_WAKEUPTIMER0 0
  Secondary Wake up timers configuration in Low Power Modes.
- #define LOWPOWER_TIMERCFG_2ND_TIMER_WAKEUPTIMER1 1
  Zigbee Wake up Counter 1 used as secondary wake up source.
- #define LOWPOWER_TIMERCFG_2ND_TIMER_BLEWAKEUPTIMER 2
  BLE Wake up Counter used as secondary wake up source.
- #define LOWPOWER_TIMERCFG_2ND_TIMER_RTC1KHZ 3
  1 KHz Real Time Counter (RTC) used as secondary wake up source
- #define LOWPOWER_TIMERCFG_2ND_TIMER_RTC1HZ 4
  1 Hz Real Time Counter (RTC) used as secondary wake up source
- #define LOWPOWER_TIMERCFG_OSC32K_FRO32KHZ 0

Wake up Timers uses FRO 32 KHz as clock source.
- #define LOWPOWER_TIMERCFG_OSC32K_XTAL32KHZ 1
  Wake up Timers uses Chrystal 32 KHz as clock source.
- #define LOWPOWER_TIMERBLECFG_RADIOEN_INDEX 0
  BLE Wake up timers configuration in Low Power Modes.
- #define RD_RIGHT (1 << 0)
  bits for access right
- #define PSECTOR_PAGE_WORDS 30
  PSECTOR_PAGE_WORDS number of 16 byte words available in page A page is 512 bytes in size.
- #define PSECTOR_PAGE0_MAGIC 0xc51d8ca9
  PSECTOR_PAGE0_MAGIC magic word to identify PAGE0 page in header.
- #define PSECTOR_PFLASH_MAGIC 0xa7b4353d
  PSECTOR_PFLASH_MAGIC magic word to identify PFLASH page in header.
- #define IMG_DIRECTORY_MAX_SIZE 8
  IMG_DIRECTORY_MAX_SIZE max number of entries in image directory Concerns Secondary Stage Bootloader only.
- #define CERTIFICATE_MARKER (0xCE27CE27)
  CERTIFICATE_MARKER magic value identifying certificate.

Typedefs

- typedef uint32_t ErrorCode_t
  enum defined in error.h
- typedef uint32_t(∗ IMAGE_VERIFY_T)(IMAGE_DATA_T ∗list_head)
  IMAGE_VERIFY_T function pointer : verification function e.g.
- typedef ISP_STATUS_T(∗ ISP_EXTENSION_T)(ISP_STATE_T ∗state, teFlashProgCommand request, uint8_t ∗in_data, uint16_t in_len, teFlashProgCommand ∗response, uint8_t ∗out_data, uint16_t ∗out_len)
  ISP_EXTENSION_T ISP extension function pointer prototype.

Enumerations

- enum AES_MODE_T { , AES_MODE_UNUSED = 0x7FFFFFFF }
  AES setup modes.
- enum AES_KEY_SIZE_T {
  AES_KEY_128BITS = 0,
  AES_KEY_192BITS,
  AES_KEY_256BITS,
  AES_FVAL = 0x7FFFFFFF }
  Size of the AES key.
- enum teFlashProgCommand { ,

TYPE_SET_RESET_REQUEST = 20,
TYPE_SET_RESET_RESPONSE ,
TYPE_FP_RUN_REQUEST,
TYPE_FP_RUN_RESPONSE ,
TYPE_FL_SET_BAUD_REQUEST,
TYPE_FL_SET_BAUD_RESPONSE ,
TYPE_REG_READ_REQUEST,
TYPE_REG_READ_RESPONSE,
TYPE_REG_WRITE_REQUEST,
TYPE_REG_WRITE_RESPONSE,
TYPE_GET_CHIP_ID_REQUEST,
TYPE_GET_CHIP_ID_RESPONSE,
TYPE_GET_FUSE_SECURED_REQUEST,
TYPE_GET_FUSE_SECURED_RESPONSE,
TYPE_MEM_OPEN_REQUEST = 0x40,
TYPE_MEM_OPEN_RESPONSE,
TYPE_MEM_ERASE_REQUEST,
TYPE_MEM_ERASE_RESPONSE,
TYPE_MEM_BLANK_CHECK_REQUEST,
TYPE_MEM_BLANK_CHECK_RESPONSE,
TYPE_MEM_READ_REQUEST,
TYPE_MEM_READ_RESPONSE,
TYPE_MEM_WRITE_REQUEST,
TYPE_MEM_WRITE_RESPONSE,
TYPE_MEM_CLOSE_REQUEST,
TYPE_MEM_CLOSE_RESPONSE,
TYPE_MEM_GET_INFO_REQUEST,
TYPE_MEM_GET_INFO_RESPONSE,
TYPE_UNLOCK_ISP_REQUEST,
TYPE_UNLOCK_ISP_RESPONSE,
TYPE_USE_CERTIFICATE_REQUEST,
TYPE_USE_CERTIFICATE_RESPONSE,
TYPE_START_ENCRYPTION_REQUEST,
TYPE_START_ENCRYPTION_RESPONSE }

ISP Message types Only a subset of message types below is supported.

- enum ISP_STATUS_T {

ISP_OK,
NOT_SUPPORTED = -1,
WRITE_FAIL = -2,
INVALID_RESPONSE = -3,
CRC_ERROR = -4,
ASSERT_FAIL = -5,
USER_INTERRUPT = -6,
READ_FAIL = -7,
TST_ERR = -8,
ISP_NOT_AUTHORISED = -9,
NO_RESPONSE = -10,
ISP_MEM_INVALID = -11,
ISP_MEM_NOT_SUPPORTED = -12,
ISP_MEM_NO_ACCESS = -13,
ISP_MEM_OUT_OF_RANGE = -14,
ISP_MEM_TOO_LONG = -15,
ISP_MEM_BAD_STATE = -16,
ISP_MEM_INVALID_MODE = -17 }
ISP Status types.
- enum ISP_MEMORY_TYPE_E { , ISP_MEM_SPIFI }
- enum MpuRegion_t {
MPU_REGION_0,
MPU_REGION_1,
MPU_REGION_2,
MPU_REGION_3,
MPU_REGION_4 ,
MPU_REGION_5,
MPU_REGION_6,
MPU_REGION_7 }
enum MpuRegion_t index of ARM CM4 MPU regions The MPU can describe up to 8 region rules.
- enum psector_partition_id_t {
PSECTOR_PAGE0_PART,
PSECTOR_PFLASH_PART }
psector_partition_id_t describes the 2 partitions of psectors.
- enum psector_page_state_t {
PAGE_STATE_BLANK,
PAGE_STATE_ERROR,
PAGE_STATE_DEGRADED,
PAGE_STATE_OK }
psector_page_state_t describes the possible states of the psector partitions.
- enum psector_write_status_t {

WRITE_OK = 0x0,
WRITE_ERROR_BAD_MAGIC,
WRITE_ERROR_INVALID_PAGE_NUMBER,
WRITE_ERROR_BAD_VERSION,
WRITE_ERROR_BAD_CHECKSUM,
WRITE_ERROR_INCORRECT_UPDATE_MODE,
WRITE_ERROR_UPDATE_INVALID,
WRITE_ERROR_PAGE_ERROR }
psector_write_status_t status code of writes to update page.
- enum AuthMode_t {
AUTH_NONE = 0,
AUTH_ON_FW_UPDATE = 1,
AUTH_ALWAYS = 2 }
AuthMode_t authentication options.

Functions

- static ErrorCode_t aesInit (void)
Initialize the AES.
- static void aesWriteByte (uint32_t offset, uint8_t val8)
AES control function, byte write (useful for writing configuration register)
- static void aesWrite (uint32_t offset, uint32_t val32)
AES control function, word write.
- static void aesRead (uint32_t offset, uint32_t ∗pVal32)
AES control function, word read.
- static void aesWriteBlock (uint32_t offset, uint32_t ∗pVal32, uint32_t numBytes)
AES control function, block write (used for multi-register block writes)
- static void aesReadBlock (uint32_t offset, uint32_t ∗pVal32, uint32_t numBytes)
AES control function, block read (used for multi-register block read)
- static ErrorCode_t aesMode (AES_MODE_T modeVal, uint32_t flags)
Sets up the AES mode.
- static ErrorCode_t aesAbort (int wipe)
Aborts optional AES operation and wipes AES engine.
- static ErrorCode_t aesLoadCounter (uint32_t counter)
Loads the increment that is used when in counter modes in the AES block.
- static ErrorCode_t aesLoadKeyFromSW (AES_KEY_SIZE_T keySize, uint32_t ∗key)
Loads the passed (software) key into the AES block.
- static ErrorCode_t aesLoadIV (uint32_t ∗pIv)
Loads the Initialization Vector (IV) into the AES block.
- static ErrorCode_t aesProcess (uint32_t ∗pBlockIn, uint32_t ∗pBlockOut, uint32_t numBlocks)
Process AES blocks (descrypt or encrypt)
- static ErrorCode_t aesWriteYInputGf128 (uint32_t ∗pYGf128)
Sets the Y input of the GF128 hash used in GCM mode.
- static ErrorCode_t aesReadGf128Hash (uint32_t ∗pGf128Hash)
Reads the results of the GF128(Z) hash used in GCM mode.
- static ErrorCode_t aesReadGcmTag (uint32_t ∗pGcmTag)
Reads the GCM tag.
- static uint32_t aesGetDriverVersion (void)
Returns the version of the AES driver in ROM.
- static ErrorCode_t aesIsSupported (void)

Returns status of AES IP block (supported or not)
- static uint32_t BOOT_RemapAddress (uint32_t address)
  Convert logical address into physical address, based on SYSCOM MEMORYREMAP register.
- static uint32_t boot_Verify_eScoreImageList (IMAGE_DATA_T ∗list_head)
  Parse the image chained list and select the first valid entry.
- static uint32_t BOOT_FindImage (uint32_t start_addr, uint32_t end_addr, uint32_t signature, IMAGE_VERIFY_T verify)
  Search for a valid executable image between boundaries in internal flash.
- static uint32_t BOOT_GetStartPowerMode (void)
  Retrieve LPMode value that has been saved previously in retained RAM bank.
- static void BOOT_SetResumeStackPointer (uint32_t stack_pointer)
  Sets the value of stack pointer to be restored on warm boot.
- static void ROM_GetFlash (uint32_t ∗address, uint32_t ∗size)
  Retrieve Internal flash address and size.
- static void ROM_GetSRAM0 (uint32_t ∗address, uint32_t ∗size)
  Retrieve SRAM0 address and size.
- static void ROM_GetSRAM1 (uint32_t ∗address, uint32_t ∗size)
  Retrieve SRAM1 address and size.
- static int ISP_Entry (ISP_EXTENSION_T isp_extension)
  This function is invoked when ISP mode is requested.
- static void Chip_LOWPOWER_SetUpLowPowerModeWakeUpTimer (LPC_LOWPOWER_T ∗p_lowpower_cfg)
  Configure Wake or RTC timers. used for testing only.
- static int Chip_LOWPOWER_SetSystemFrequency (uint32_t frequency)
  Configure CPU and System Bus clock frequency.
- static int Chip_LOWPOWER_SetMemoryLowPowerMode (uint32_t p_sram_instance, uint32_t p_sram_lp_mode)
  Configure Memory instances Low Power Mode.
- static void Chip_LOWPOWER_GetSystemVoltages (LPC_LOWPOWER_LDOVOLTAGE_T ∗p_ldo_voltage)
  Get System Voltages.
- static void Chip_LOWPOWER_SetSystemVoltages (LPC_LOWPOWER_LDOVOLTAGE_T ∗p_ldo_voltage)
  Configure System Voltages.
- static void Chip_LOWPOWER_SetLowPowerMode (LPC_LOWPOWER_T ∗p_lowpower_cfg)
  Configure and enters in low power mode.
- static void Chip_LOWPOWER_ChipSoftwareReset (void)
  Perform a Full chip reset using Software reset bit in PMC.
- static void Chip_LOWPOWER_ArmSoftwareReset (void)
  Perform a digital System reset.
- static int MPU_pSectorGrantAccessRights (uint32_t addr, size_t sz, MPU_reg_settings_t ∗save_rule)
  This function is used to grant access to the pSector region.
- static int MPU_pSectorWithdrawAccessRights (MPU_reg_settings_t ∗save_rule)
  This function is used to withdraw access to the pSector region.
- static void MPU_GetCurrentSettings (MPU_Settings_t ∗settings)
  This function is used to read the MPU settings into a RAM structure.
- static int MPU_AllocateRegionDesc (void)
  This function is used to select the first available rule.
- static uint32_t pmc_reset_get_cause (void)

Get the cause of the reset.
- static void pmc_reset_clear_cause (uint32_t mask)
  Clear the cause of the reset.
- static psector_write_status_t psector_WriteUpdatePage (psector_partition_id_t part_index, psector-_page_t *page)
  This function is used to validate a page content and write it to the update page.
- static void psector_EraseUpdate (void)
  This function is used to validate a page content and write it to the update page.
- static psector_page_state_t psector_ReadData (psector_partition_id_t part_index, int page_number, uint32_t offset, uint32_t size, void *data)
  This function is used to read data from a psector partition.
- static uint32_t psector_CalculateChecksum (psector_page_t *psector_page)
  This function is used to calculate a page checksum.
- static uint64_t psector_Read_CustomerId (void)
  This function returns the CustomerId field.
- static int psector_Read_RomPatchInfo (uint32_t *patch_region_sz, uint32_t *patch_region_addr, uint32_t *patch_checksum, uint32_t *patch_checksum_valid)
  This function returns the ROM patch information read from the PFLASH.
- static uint16_t psector_Read_ImgAuthLevel (void)
  This function returns the image authentication level from the PFLASH.
- static uint32_t psector_Read_AppSearchGranularity (void)
  This function returns the app search granularity value from the PFLASH.
- static uint32_t psector_Read_QspiAppSearchGranularity (void)
  This function returns the Qspi app search granularity value from the PFLASH.
- static uint64_t psector_Read_DeviceId (void)
  This function returns the DeviceId value from the PFLASH.
- static int psector_Read_UnlockKey (int *valid, uint8_t key[256], bool raw)
  This function returns the unlock key value from the PFLASH.
- static int psector_Read_ISP_protocol_key (uint8_t key[16])
  This function returns the ISP protocol AES key from PFLASH.
- static uint64_t psector_ReadIeee802_15_4_MacId1 (void)
  This function returns the IEEE-802.15.4 Mac address first instance from PFLASH.
- static uint64_t psector_ReadIeee802_15_4_MacId2 (void)
  This function returns the IEEE-802.15.4 Mac address second instance from PFLASH.
- static uint64_t psector_Read_MinDeviceId (void)
  This function returns the Min Device id from PFLASH.
- static uint64_t psector_Read_MaxDeviceId (void)
  This function returns the Max Device id from PFLASH.
- static uint32_t psector_Read_MinVersion (void)
  This function returns the Min Version from PAGE0.
- static psector_write_status_t psector_SetEscoreImageData (uint32_t image_addr, uint32_t min_-version)
  This function is used to set the selected image address and MinVersion into PAGE0.
- static psector_page_state_t psector_ReadEscoreImageData (uint32_t *image_addr, uint32_t *min-_version)
  This function returns the image address and min version value from PAGE0.
- static int psector_Read_ImagePubKey (int *valid, uint8_t key[256], bool raw)
  This function returns the unlock key value from PAGE0.
- static uint32_t secure_VerifySignature (uint8_t *hash, const uint8_t *signature, const uint32_t *key)
  This function performs an RSA 2048 signature verification.
- static uint32_t secure_VerifyBlock (uint8_t *start, uint32_t length, const uint32_t *key, const uint8-

_t *signature)

This function performs an RSA 2048 signature verification over specified data block.

- static uint32_t secure_VerifyCertificate (const IMAGE_CERT_T *certificate, const uint32_t *key, const uint8_t *cert_signature)

  This function performs an RSA 2048 signature verification.
- static uint32_t secure_VerifyImage (uint32_t image_addr, const IMAGE_CERT_T *root_cert)

  This function verifies image authenticity.

Variables

- uint32_t IMAGE_DATA_T::version

  version number found in image
- uint32_t IMAGE_DATA_T::address

  start address of image
- struct _image_data_t * IMAGE_DATA_T::next

  pointer on next IMAGE_DATA_T in list
- uint32_t ISP_MEM_INFO_T::base_address

  base address of memory bank
- uint32_t ISP_MEM_INFO_T::length

  total size
- uint32_t ISP_MEM_INFO_T::block_size

  block size : flash page size
- uint16_t ISP_MEM_INFO_T::flags

  unused
- ISP_MEMORY_TYPE_E ISP_MEM_INFO_T::type

  memory type : note that EFUSE bank is not a memory as such - SPIFI is unimplemented
- uint8_t ISP_MEM_INFO_T::access

  bitfield of access rights:
- uint8_t ISP_MEM_INFO_T::auth_access

  similar to access for authenticated commands
- ISP_MEM_FUNC_T * ISP_MEM_INFO_T::func

  set of function pointers of this memory type see @ ISP_MEM_FUNC_T
- const char * ISP_MEM_INFO_T::name

  name of memory bank
- uint32_t ISP_ENC_STATE_T::mode

  0: none - 1: AES CTR
- uint32_t ISP_ENC_STATE_T::start

  start address of cipher/decipher operation
- uint32_t ISP_ENC_STATE_T::end

  end address of cipher/decipher operation
- uint32_t ISP_ENC_STATE_T::iv [4]

  Initialization vector IV : 16 bytes.
- uint32_t ISP_ENC_STATE_T::key [8]

  AES Key - key[4..7] unused.
- ISP_GET_MEMORY_T ISP_STATE_T::get_memory

  Function pointer to get_memory.
- ISP_EXTENSION_T ISP_STATE_T::extension

  Function pointer to extension.
- uint32_t * ISP_STATE_T::buffer

  buffer holding command (in stack)
- ISP_ENC_STATE_T ISP_STATE_T::enc_state

Embedded ciphering structure see @ ISP_ENC_STATE_T.
- IMAGE_CERT_T ISP_STATE_T::certificate
  Certificate used to authenticate ISP commands it is composed of the custumer identifier and the unlock public key found in PFLASH.
- uint8_t ISP_STATE_T::stored_hash [32]
  SHA=256 hash storage.
- uint8_t ISP_STATE_T::mode
  mode 0x00: inactive
- uint8_t ISP_STATE_T::isp_level
  ISP level as restrained by EFUSE configuation and PFLASH parameter.
- uint16_t ISP_STATE_T::buffer_size
  size of buffer : normally 1024
- uint8_t ISP_STATE_T::unlock_disable
  unlock forbidden by EFUSE
- uint8_t ISP_STATE_T::SWD_disable
  SWD Debug interface disabled.
- uint32_t MPU_Settings_t::ctrl
  MPU Ctrl register.
- uint32_t MPU_Settings_t::rbar [8]
  MPU RBAR array for the 8 rules.
- uint32_t MPU_Settings_t::rasr [8]
  MPU RASR array for the 8 rules.
- uint32_t image_directory_entry_t::img_base_addr
  image start address in internal Flash or QSPI flash
- uint16_t image_directory_entry_t::img_nb_pages
  image number of 512 byte pages
- uint8_t image_directory_entry_t::flags
  IMG_FLAG_BOOTABLE : bit 0, other TBD.
- uint8_t image_directory_entry_t::img_type
  image type
- uint32_t psector_header_t::checksum
  page checksum
- uint32_t psector_header_t::magic
  magic: PSECTOR_PAGE0_MAGIC or PSECTOR_PFLASH_MAGIC
- uint16_t psector_header_t::page_number
  should be 0 because both partitions contain a single page
- struct {
  } psector_page_data_t::page0_v2

  Deprecated form kept for backward compatibility.
- uint32_t psector_page_data_t::SelectedImageAddress
  Address of image to be loaded by boot ROM offset 0x20.
- uint32_t psector_page_data_t::preferred_app_index
  for use with SSBL: index of application to select from image directory value 0..8 offset 0x24
- image_directory_entry_t psector_page_data_t::ota_entry
  New image written by OTA : SSBL to check validity and authentication offset 0x28.
- uint32_t psector_page_data_t::MinVersion
  Minimum version accepted : application's version number must be greater than this one to be accepted.
- uint32_t psector_page_data_t::img_pk_valid
  Image public key valid offset 0x34.
- uint32_t psector_page_data_t::flash_audit_done

Flash audit done: already sought for wrongly initialized pages offset 0x38.
- uint32_t psector_page_data_t::RESERVED1
  padding reserved word
- uint8_t psector_page_data_t::image_pubkey [256]
  RSA Public Key to be used to verify authenticity offset 0x40.
- uint8_t psector_page_data_t::zigbee_install_code [36]
  Zigbee install code offset 0x140.
- uint32_t psector_page_data_t::RESERVED3 [3]
  padding reserved wordes
- uint8_t psector_page_data_t::zigbee_password [16]
  Zigbee password offset 0x170.
- image_directory_entry_t psector_page_data_t::img_directory [IMG_DIRECTORY_MAX_SIZE]
  < Image directory entries array, used by OTA process to locate images and/or blobs offset 0x180
- uint32_t psector_page_data_t::rom_patch_region_addr
  ROM patch entry point address.
- uint32_t psector_page_data_t::rom_patch_checksum_valid
  ROM patch checksum valid: 0 means invalid Any other value means valid.
- uint32_t psector_page_data_t::ISP_access_level
  ISP access level: 0 means full access, unsecure 0x01010101 means full access, secure 0x02020202 means write only, unsecure 0x03030303 means write only, secure 0x04040404 means locked Any other value means disabled.
- uint16_t psector_page_data_t::application_flash_sz
  Application flash size, in kilobytes.
- uint16_t psector_page_data_t::image_authentication_level
  Image authentication level: 0 means check only header validity 1 means check signature of whole image if image has changed 2 means check signature of whole image on every cold start.
- uint16_t psector_page_data_t::unlock_key_valid
  0: unlock key is not valid, >= 1: is present
- uint16_t psector_page_data_t::ram1_bank_sz
  RAM bank 1 size, in kilobytes.
- uint32_t psector_page_data_t::app_search_granularity
  Application search granularity (increment), in bytes.
- uint8_t psector_page_data_t::ISP_protocol_key [16]
  ISP protocol key: key used to encrypt messages over ISP UART with secure access level.
- uint64_t psector_page_data_t::ieee_mac_id1
  IEEE_MAC_ID_1 (Used to over-ride MAC ID_1 in N-2 page)
- uint64_t psector_page_data_t::ieee_mac_id2
  IEEE_MAC_ID_2 if second MAC iID is required.
- uint64_t psector_page_data_t::ble_mac_id
  BLE device address : only 6 LSB bytes are significant.
- uint8_t psector_page_data_t::reserved2 [104]
  Reserved for future use.
- uint64_t psector_page_data_t::customer_id
  Customer ID, used for secure handshake.
- uint64_t psector_page_data_t::min_device_id
  Min Device ID, used for secure handshake - Certificate compatibility.
- uint64_t psector_page_data_t::device_id
  Device ID, used for secure handshake.
- uint64_t psector_page_data_t::max_device_id
  Max Device ID, used for secure handshake - Certificate compatibility.
- uint8_t psector_page_data_t::unlock_key [256]

2048-bit public key for secure handshake (equivalent to 'unlock' key).
- uint32_t IMAGE_CERT_T::certificate_marker
  Certificate marker: magic see @ CERTIFICATE_MARKER.
- uint32_t IMAGE_CERT_T::certificate_id
  Certificate id.
- uint32_t IMAGE_CERT_T::usage_flags
  Usage flags: mostly used in the unlocking procedure.
- uint64_t IMAGE_CERT_T::customer_id
  Customer Id: customer chosen identifier.
- uint64_t IMAGE_CERT_T::min_device_id
  Min device id: min device version from which certificate applies.
- uint64_t IMAGE_CERT_T::max_device_id
  Max device id: max device version up to which certificate applies.
- uint32_t IMAGE_CERT_T::public_key [SIGNATURE_LEN/4]
  RSA-2048 public key.
- IMAGE_CERT_T ImageAuthTrailer_t::certificate
  The certificate see @ IMAGE_CERT_T.
- uint8_t ImageAuthTrailer_t::cert_signature [SIGNATURE_LEN]
  The signature of the certificate.
- uint8_t ImageAuthTrailer_t::img_signature [SIGNATURE_LEN]
  The image siganture.

37.2 Data Structure Documentation

37.2.1 struct IMAGE_DATA_T

Data Fields

- uint32_t version
  version number found in image
- uint32_t address
  start address of image
- struct _image_data_t ∗ next
  pointer on next IMAGE_DATA_T in list

37.2.2 struct ISP_MEM_FUNC_T

37.2.3 struct ISP_MEM_INFO_T

Data Fields

- uint32_t base_address
  base address of memory bank
- uint32_t length
  total size
- uint32_t block_size
  block size : flash page size

- uint16_t flags
  unused
- ISP_MEMORY_TYPE_E type
  memory type : note that EFUSE bank is not a memory as such - SPIFI is unimplemented
- uint8_t access
  bitfield of access rights:
- uint8_t auth_access
  similar to access for authenticated commands
- ISP_MEM_FUNC_T ∗ func
  set of function pointers of this memory type see @ ISP_MEM_FUNC_T
- const char ∗ name
  name of memory bank

37.2.4 struct ISP_ENC_STATE_T

Data Fields

- uint32_t mode
  0: none - 1: AES CTR
- uint32_t start
  start address of cipher/decipher operation
- uint32_t end
  end address of cipher/decipher operation
- uint32_t iv [4]
  Initialization vector IV : 16 bytes.
- uint32_t key [8]
  AES Key - key[4..7] unused.

37.2.5 struct ISP_STATE_T

Note: this context is held in RAM is the stack so is lost after ISP_Entry is exited.

Data Fields

- ISP_GET_MEMORY_T get_memory
  Function pointer to get_memory.
- ISP_EXTENSION_T extension
  Function pointer to extension.
- uint32_t ∗ buffer
  buffer holding command (in stack)
- ISP_ENC_STATE_T enc_state
  Embedded ciphering structure see @ ISP_ENC_STATE_T.
- IMAGE_CERT_T certificate
  Certificate used to authenticate ISP commands it is composed of the custumer identifier and the unlock public key found in PFLASH.
- uint8_t stored_hash [32]

SHA=256 hash storage.
- uint8_t mode
  mode 0x00: inactive
- uint8_t isp_level
  ISP level as restrained by EFUSE configuation and PFLASH parameter.
- uint16_t buffer_size
  size of buffer : normally 1024
- uint8_t unlock_disable
  unlock forbidden by EFUSE
- uint8_t SWD_disable
  SWD Debug interface disabled.

37.2.6 struct LPC_LOWPOWER_T

Data Fields

- uint32_t CFG
  Low Power Mode Configuration, and miscallenous options.
- uint32_t PMUPWDN
  Analog Power Domains (analog components in Power Management Unit) Low Power Modes.
- uint32_t DIGPWDN
  Digital Power Domains Low Power Modes.
- uint32_t VOLTAGE
  LDO Voltage control in Low Power Modes.
- uint32_t WAKEUPSRCINT0
  Wake up sources Interrupt control.
- uint32_t WAKEUPSRCINT1
  Wake up sources Interrupt control.
- uint32_t SLEEPPOSTPONE
  Interrupt that can postpone power down modes in case an interrupt is pending when the processor request deepsleep.
- uint32_t WAKEUPIOSRC
  Wake up I/O sources.
- uint32_t GPIOLATCH
  I/Os which outputs level must be kept (in Power Down mode)
- uint32_t TIMERCFG
  Wake up timers configuration.
- uint32_t TIMERBLECFG
  BLE wake up timer configuration (OSC_EN and RADIO_EN)
- uint32_t TIMERCOUNTLSB
  Wake up Timer LSB.
- uint32_t TIMERCOUNTMSB
  Wake up Timer MSB.
- uint32_t TIMER2NDCOUNTLSB
  Second Wake up Timer LSB.
- uint32_t TIMER2NDCOUNTMSB
  Second Wake up Timer MSB.

37.2.7 struct LPC_LOWPOWER_LDOVOLTAGE_T

Data Fields

- uint8_t LDOPMU
  Always-ON domain LDO voltage configuration.
- uint8_t LDOPMUBOOST
  Always-ON domain LDO Boost voltage configuration.
- uint8_t LDOMEM
  Memories LDO voltage configuration.
- uint8_t LDOMEMBOOST
  Memories LDO Boost voltage configuration.
- uint8_t LDOCORE
  Core Logic domain LDO voltage configuration.
- uint8_t LDOFLASHNV
  Flash NV domain LDO voltage configuration.
- uint8_t LDOFLASHCORE
  Flash Core domain LDO voltage configuration.
- uint8_t LDOADC
  General Purpose ADC LDO voltage configuration.
- uint8_t LDOPMUBOOST_ENABLE
  Force Boost activation on LDOPMU.

37.2.8 struct MPU_Settings_t

Data Fields

- uint32_t ctrl
  MPU Ctrl register.
- uint32_t rbar [8]
  MPU RBAR array for the 8 rules.
- uint32_t rasr [8]
  MPU RASR array for the 8 rules.

37.2.9 struct image_directory_entry_t

Data Fields

- uint32_t img_base_addr
  image start address in internal Flash or QSPI flash
- uint16_t img_nb_pages
  image number of 512 byte pages
- uint8_t flags
  IMG_FLAG_BOOTABLE : bit 0, other TBD.
- uint8_t img_type
  image type

37.2.10 struct psector_header_t

Data Fields

- uint32_t checksum
  page checksum
- uint32_t magic
  magic: PSECTOR_PAGE0_MAGIC or PSECTOR_PFLASH_MAGIC
- uint16_t page_number
  should be 0 because both partitions contain a single page

37.2.11 struct IMAGE_CERT_T

Data Fields

- uint32_t certificate_marker
  Certificate marker: magic see @ CERTIFICATE_MARKER.
- uint32_t certificate_id
  Certificate id.
- uint32_t usage_flags
  Usage flags: mostly used in the unlocking procedure.
- uint64_t customer_id
  Customer Id: customer chosen identifier.
- uint64_t min_device_id
  Min device id: min device version from which certificate applies.
- uint64_t max_device_id
  Max device id: max device version up to which certificate applies.
- uint32_t public_key [SIGNATURE_LEN/4]
  RSA-2048 public key.

37.3 Macro Definition Documentation

37.3.1 #define ISP_FLAG_HAS_CRC32 (1 << 0)

CRC32 Now deprecated

37.3.2 #define LOWPOWER_CFG_FLASHPWDNMODE_FLASHPWND 0

Only valid in DEEP SLEEP mode

37.3.3 #define LOWPOWER_PMUPWDN_DCDC (1UL << 0)

Power Down DCDC Converter

37.3.4 #define LOWPOWER_DIGPWDN_FLASH (1UL << 6)

Power Down Flash Power Domain (Flash Macro, Flash Controller and/or FLash LDOs, depending on \ \ LOWPOWER_CFG_FLASHPWDNMODE parameter)

37.3.5 #define LOWPOWER_WAKEUPSRCINT0_SYSTEM_IRQ (1UL << 0)

BOD, Watchdog Timer, Flash controller, Firewall [DEEP SLEEP] BOD [POWER_DOWN]

37.3.6 #define LOWPOWER_SLEEPPOSTPONE_FORCED (1UL << 0)

Forces postpone of power down modes in case the processor request low power mode

37.3.7 #define LOWPOWER_TIMERCFG_TIMER_WAKEUPTIMER0 0

Zigbee Wake up Counter 0 used as wake up source

37.3.8 #define LOWPOWER_TIMERCFG_2ND_TIMER_WAKEUPTIMER0 0

Zigbee Wake up Counter 0 used as secondary wake up source

37.3.9 #define PSECTOR_PAGE_WORDS 30

That is 32x16 bytes. The first 32 bytes contain the page header, which leaves 30x16bytes for storage. Thence the 30.

37.4 Typedef Documentation

37.4.1 typedef uint32_t(* IMAGE_VERIFY_T)(IMAGE_DATA_T *list_head)

boot_Verify_eScoreImageList

37.5 Enumeration Type Documentation

37.5.1 enum AES_MODE_T

Enumerator

AES_MODE_UNUSED Not used, but forces enum to 32-bit size.

37.5.2 enum AES_KEY_SIZE_T

Enumerator

AES_KEY_128BITS KEY size 128 bits.
AES_KEY_192BITS KEY size 192 bits.
AES_KEY_256BITS KEY size 256 bits.
AES_FVAL Not used, but forces enum to 32-bit size and unsigned.

37.5.3 enum teFlashProgCommand

Enumerator

TYPE_SET_RESET_REQUEST ISP Set Reset request.
TYPE_SET_RESET_RESPONSE ISP Set Reset response.
TYPE_FP_RUN_REQUEST ISP FP Run request : jump to address if ISP access level and authentication allow it.
TYPE_FP_RUN_RESPONSE ISP FP Run response.
TYPE_FL_SET_BAUD_REQUEST ISP FL Set Baud request : set UART speed.
TYPE_FL_SET_BAUD_RESPONSE ISP FL Set Baud response.
TYPE_REG_READ_REQUEST not implemented
TYPE_REG_READ_RESPONSE not implemented
TYPE_REG_WRITE_REQUEST not implemented
TYPE_REG_WRITE_RESPONSE not implemented
TYPE_GET_CHIP_ID_REQUEST ISP chip id request.
TYPE_GET_CHIP_ID_RESPONSE ISP chip id response.
TYPE_GET_FUSE_SECURED_REQUEST not implemented
TYPE_GET_FUSE_SECURED_RESPONSE not implemented
TYPE_MEM_OPEN_REQUEST ISP memory open request.
TYPE_MEM_OPEN_RESPONSE ISP memory open response.
TYPE_MEM_ERASE_REQUEST ISP memory erase request, applies to internal flash only.
TYPE_MEM_ERASE_RESPONSE ISP memory erase response.
TYPE_MEM_BLANK_CHECK_REQUEST ISP memory blank check request, applies to internal flash only.
TYPE_MEM_BLANK_CHECK_RESPONSE ISP memory blank check response to request.
TYPE_MEM_READ_REQUEST ISP memory read request, applies to all memory types.
TYPE_MEM_READ_RESPONSE ISP memory read response.
TYPE_MEM_WRITE_REQUEST ISP memory write request, applies to all memory types except EFUSE.
TYPE_MEM_WRITE_RESPONSE ISP memory read response.
TYPE_MEM_CLOSE_REQUEST ISP memory close request.
TYPE_MEM_CLOSE_RESPONSE ISP memory close response.
TYPE_MEM_GET_INFO_REQUEST ISP memory get information of memory geometry and accessibility.

TYPE_MEM_GET_INFO_RESPONSE ISP memory get information response.
TYPE_UNLOCK_ISP_REQUEST ISP unlock request: reset a locked device to its pristine state.
TYPE_UNLOCK_ISP_RESPONSE ISP unlock response.
TYPE_USE_CERTIFICATE_REQUEST ISP Use Certificate request.
TYPE_USE_CERTIFICATE_RESPONSE ISP Use Certificate response.
TYPE_START_ENCRYPTION_REQUEST ISP Start Encryption request.
TYPE_START_ENCRYPTION_RESPONSE ISP Start Encryption response.

37.5.4 enum ISP_STATUS_T

Enumerator

ISP_OK ISP operation successful.
NOT_SUPPORTED ISP operation not supported.
WRITE_FAIL ISP write failure when writing to FLASH, PSECT,r PFLASH.
INVALID_RESPONSE ISP invalid response : not used.
CRC_ERROR ISP command received CRC incorrect.
ASSERT_FAIL ISP received too long a message.
USER_INTERRUPT ISP User aborted operation: not used.
READ_FAIL ISP Flash blank check error or Flash excessive ECC errors.
TST_ERR not used
ISP_NOT_AUTHORISED ISP order authentification failure.
NO_RESPONSE not used
ISP_MEM_INVALID ISP message malformed : addressed to non existant memory.
ISP_MEM_NOT_SUPPORTED ISP order not supported for memory.
ISP_MEM_NO_ACCESS ISP access level insufficient.
ISP_MEM_OUT_OF_RANGE ISP order addressing memory outisde the intended range.
ISP_MEM_TOO_LONG ISP buffer insufficient to read requested amount of memory.
ISP_MEM_BAD_STATE Memory in a state that cannot support operation e.g. opening an errored PSECT or PFLASH, closing a memory that was not opened
ISP_MEM_INVALID_MODE ISP order is malformed : mode incorrect.

37.5.5 enum ISP_MEMORY_TYPE_E

Enumerator

ISP_MEM_SPIFI Unused SPIFI not handled by ISP.

37.5.6 enum MpuRegion_t

This function is used to set access rights to a region.

Note that a higher order rule prevails over the lower ones. The boot ROM makes use of upper order rules : 5 .. 7. Rule 0 is a ’background’ rule that opens the whole memory plane.

Parameters

region_id,:	0 .. 7 see
addr,:	address of region
sz,:	region size
rwx_rights,:	bit field RD_RIGHT(0) - WR_RIGHT(1) - X_RIGHT(2)
save_rule,:	save a copy of previous rule

Returns

-1 if failure, if succesful return the size of the region.

Called after previous call to see

Parameters

region_id,:	0 .. 7 see
save_rule,:	saved copy of previous rule to be restored. if this parameter is NULL, RBAR and RASR of the given region_id are cleared.

Returns

-1 if failure, if succesful return the size of the region.

Enumerator

MPU_REGION_0 Boot Reserved: background rule
MPU_REGION_1 General purpose rule
MPU_REGION_2 General purpose rule
MPU_REGION_3 General purpose rule
MPU_REGION_4 General purpose rule
MPU_REGION_5 Boot Reserved
MPU_REGION_6 Boot Reserved
MPU_REGION_7 Boot Reserved

37.5.7 enum psector_partition_id_t

Note: PAGE0 is termed PSECT in the FlashProgrammer, whereas PFLASH remains PFLASH.

Enumerator

PSECTOR_PAGE0_PART Page0 partition : termed PSECT by FLashProgramemr tool Image related data.
PSECTOR_PFLASH_PART PFLASH : Custommer configuration data.

37.5.8 enum psector_page_state_t

Enumerator

PAGE_STATE_BLANK Page has never been programmed or has been erased.
PAGE_STATE_ERROR Both subpages constituting the psector contain unrecoverable errors that ECC/parity cannot mend.
PAGE_STATE_DEGRADED One subpage contains unrecoverable errors or is blank.
PAGE_STATE_OK Both subpages are correct.

37.5.9 enum psector_write_status_t

Enumerator

WRITE_OK Succeded in writing page.
WRITE_ERROR_BAD_MAGIC Magic word incorrect in page header.
WRITE_ERROR_INVALID_PAGE_NUMBER Invalid page number (higher than partition size)
WRITE_ERROR_BAD_VERSION Invalid version number: must increment monotonically.
WRITE_ERROR_BAD_CHECKSUM Invalid checksum.
WRITE_ERROR_INCORRECT_UPDATE_MODE Update mode incorrect.
WRITE_ERROR_UPDATE_INVALID Update invalid.
WRITE_ERROR_PAGE_ERROR Failure to program page in flash.

37.5.10 enum AuthMode_t

Enumerator

AUTH_NONE no authentication is performed
AUTH_ON_FW_UPDATE authentication is performed on firmware update
AUTH_ALWAYS authentication is performed at each Cold boot

37.6 Function Documentation

37.6.1 static ErrorCode_t aesInit (void) `[inline]`, `[static]`

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

Note

Driver does not enable AES clock, power, or perform reset peripheral (if needed).

37.6.2 static void aesWriteByte (uint32_t *offset*, uint8_t *val8*) [inline], [static]

offset : Register offset in AES, 32-bit aligned value val8 : 8-bit value to write

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can used it perform byte level write access to a register. This function is not meant to be public.

37.6.3 static void aesWrite (uint32_t *offset*, uint32_t *val32*) [inline], [static]

offset : Register offset in AES, 32-bit aligned value val32 : 32-bit value to write

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can used it for write access to a register. This function is not meant to be public.

37.6.4 static void aesRead (uint32_t *offset*, uint32_t * *pVal32*) [inline], [static]

offset : Register offset in AES, 32-bit aligned value pVal32 : Pointer to 32-bit area to read into

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can used it for read access to a register. This function is not meant to be public.

37.6.5 static void aesWriteBlock (uint32_t *offset*, uint32_t * *pVal32*, uint32_t *numBytes*) [inline], [static]

offset : Register offset in AES, 32-bit aligned value pVal32 : Pointer to 32-bit array to write numBytes : Number of bytes to write, must be 32-bit aligned

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can used it for write access to a register. This function is not meant to be public. Writes occur in 32-bit chunks.

37.6.6 static void aesReadBlock (uint32_t *offset*, uint32_t * *pVal32*, uint32_t *numBytes*) [inline], [static]

offset : Register offset in AES, 32-bit aligned value pVal32 : Pointer to 32-bit array to read into numBytes : Number of bytes to read, must be 32-bit aligned

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can used it for read access to a register. This function is not meant to be public. Reads occur in 32-bit chunks. Read data if undefined if AES not present.

37.6.7 static ErrorCode_t aesMode (AES_MODE_T *modeVal*, uint32_t *flags*) [inline], [static]

Parameters

wipe	: use true to invalidate AES key and disable cipher
flags	: Applies extra flags (Or'ed in config), normally should be 0, useful for swap bits only

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

37.6.8 static ErrorCode_t aesAbort (int *wipe*) [inline], [static]

Parameters

wipe	: use true to invalidate AES key and disable cipher

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

37.6.9 static ErrorCode_t aesLoadCounter (uint32_t *counter*) [inline], [static]

Parameters

counter	: 32-bit initial increment counter value

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

37.6.10 static ErrorCode_t aesLoadKeyFromSW (AES_KEY_SIZE_T *keySize*, uint32_t * *key*) [inline], [static]

Parameters

keySize	: 0 = 128-bits, 1 = 192-bits, 2 = 256-bits, all other values are invalid (AES_KEY_SIZE_T)
key	: Pointer to up to a 256-bit key array

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

37.6.11 static ErrorCode_t aesLoadIV (uint32_t * *pIv*) [inline], [static]

Parameters

iv	: 32-bit initialization vector

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

37.6.12 static ErrorCode_t aesProcess (uint32_t * *pBlockIn,* uint32_t * *pBlockOut,* uint32_t *numBlocks*) [inline], [static]

Parameters

pBlockIn	: 32-bit aligned pointer to input block of data
pBlockOut	: 32-bit aligned pointer to output block of data
numBlocks	: Number of blocks to process, block size = 128 bits

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

Note

The AES mode and key must be setup prior to calling this function. For encryption. the plain text is used as the input and encrypted text is output. For descryption, plain text is output while encrypted text is input.

37.6.13 static ErrorCode_t aesWriteYInputGf128 (uint32_t * *pYGf128*) [inline], [static]

Parameters

pYGf128	: Y input of GF128 hash (4x32-bit words)

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

Note

Calling this function will reset the hash logic.

37.6.14 static ErrorCode_t aesReadGf128Hash (uint32_t ∗ *pGf128Hash*) [inline], [static]

Parameters

pGf128Hash	: Array of 4x32-bit words to read hash into

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

Note

Value is undefined if AES is not present.

37.6.15 static ErrorCode_t aesReadGcmTag (uint32_t ∗ *pGcmTag*) [inline], [static]

Parameters

pGcmTag	: Array of 4x32-bit words to read GCM tage into

Returns

LPC_OK on success, or an error code (ERRORCODE_T) on failure

Note

The GCM tage is an XOR value of the Output Text and GF128(Z) hash value. Value is undefined if AES is not present.

37.6.16 static uint32_t aesGetDriverVersion (void) [inline], [static]

Returns

Driver version, example 0x00000100 = v1.0

37.6.17 static ErrorCode_t aesIsSupported (void) [inline], [static]

Returns

LPC_OK if enabled, ERR_SEC_AES_NOT_SUPPORTED if not supported

37.6.18 static uint32_t BOOT_RemapAddress (uint32_t *address*) [inline], [static]

The chip has a remapping capability that allows to remap internal flash areas. This feature is part of the firmware update mechanism (OTA).

Parameters

address	logical address to convert

Returns

physical address

37.6.19 static uint32_t boot_Verify_eScoreImageList (IMAGE_DATA_T * *list_head*) [inline], [static]

The image list is already sorted by version number. Compare image version against Min version read from PSECT. If it is greater than or equal to Min version, perform the RSA authentication over the image using the ket found in PFLASH if any. see IMAGE_VERIFY_T

Parameters

list_head	sorted list of images

Returns

selected image start address

37.6.20 static uint32_t BOOT_FindImage (uint32_t *start_addr*, uint32_t *end_addr*, uint32_t *signature*, IMAGE_VERIFY_T *verify*) [inline], [static]

This function is involved in the search of a bootable image. It is called by the boot ROM on Cold boot but can be called by the Selective OTA.

The application granularity parameter is read from the PSECT, this is used as the increment used to hop to next position in case of failure. The function builds up a chained list of image descriptors that it sorts by version number. The intent is that the most recent version is at the head of the list.

Parameters

start_addr	address from which to start search
end_addr	address from which to start search
signature	magic identifier : constant 0x98447902
IMAGE_VERIFY_T	verification function pointer (see) This parameter cannot be NULL. The implementer may opt for a version that simply returns the head of the chained list.

Returns

image address if valid, IMAGE_INVALID_ADDR (0xffffffff) otherwise

37.6.21 static uint32_t BOOT_GetStartPowerMode (void) [inline], [static]

This is mostly used to determine in which power mode the PMC was before reset, i.e. whether is is a cold or warm reset. This is to be invoked from ResetISR2

Parameters

none	

Returns

LPMode

37.6.22 static void BOOT_SetResumeStackPointer (uint32_t *stack_pointer*) [inline], [static]

Parameters

stack_pointer	address to be written in retained RAM bank so that boot ROM restores value on warm start

Returns

none

37.6.23 static void ROM_GetFlash (uint32_t * *address*, uint32_t * *size*) [inline], [static]

The internal flash start address is necessarily 0. Its size may vary depending on chip options. The size returned is the number of bytes usable for program and data. The maximum possible value is 0x9dc00.

Parameters

address	pointer on location to store returned address
size	pointer on location to store returned size

Returns

*address is 0x00000000UL and *size is up to 0x9dc00

37.6.24 static void ROM_GetSRAM0 (uint32_t * *address*, uint32_t * *size*) [inline], [static]

Parameters

address	pointer on location to store returned address
size	pointer on location to store returned size

Returns

*address is 0x04000000UL and *size is 88k (0x16000)

37.6.25 static void ROM_GetSRAM1 (uint32_t * *address*, uint32_t * *size*) [inline], [static]

```
SRAM1 presence is optional depending on chip variant
```

Parameters

address	pointer on location to store returned address
size	pointer on location to store returned size

Returns

if SRAM1 not present ∗address is 0 and ∗size is 0, otherwise ∗address is 0x04020000UL and ∗size is up to 64k (0x10000)

37.6.26 static int ISP_Entry (ISP_EXTENSION_T *isp_extension*) [inline], [static]

The ISP mode is requested when GPIO 5 is held down on rest or when no valid image can be found in the in the internal flash.

Parameters

isp_extension	function pointer on extension function. ISP_INVALID_EXTENSION (0) : no extension requested is the only implemented choice Note: ISP_Entry reads from vector table [13] in order to find a possible extension funcion. The boot ROM has a 0 value at that location.

Returns

status 0: ISP entered successfully, otherwise error was detected (ISP disabled)

37.6.27 static void Chip_LOWPOWER_SetUpLowPowerModeWakeUpTimer (LPC_LOWPOWER_T ∗ *p_lowpower_cfg*) [inline], [static]

Parameters

p_lowpower_-cfg,:	pointer to a structure that contains all low power mode parameters

Returns

Nothing

37.6.28 static int Chip_LOWPOWER_SetSystemFrequency (uint32_t *frequency*) [inline], [static]

Parameters

Frequency	:

Returns

Nothing

37.6.29 static int Chip_LOWPOWER_SetMemoryLowPowerMode (uint32_t *p_sram_instance,* uint32_t *p_sram_lp_mode*) [inline], [static]

Parameters

p_sram_-instance,:	SRAM instance number, between 0 and 11.
p_sram_lp_-mode	: Low power mode : LOWPOWER_SRAM_LPMODE_ACTIVE, LOWPOWER_S-RAM_LPMODE_SLEEP, LOWPOWER_SRAM_LPMODE_DEEPSLEEP, LOWP-OWER_SRAM_LPMODE_SHUTDOWN

Returns

Status code

37.6.30 static void Chip_LOWPOWER_GetSystemVoltages (LPC_LOWPOWER_LDOVOLTAGE_T ∗ *p_ldo_voltage*) [inline], [static]

Parameters

p_ldo_voltage,:	pointer to a structure to fill with current voltages on the chip

Returns

Nothing

37.6.31 static void Chip_LOWPOWER_SetSystemVoltages (LPC_LOWPOWER_LDOVOLTAGE_T ∗ *p_ldo_voltage*) [inline], [static]

Parameters

p_ldo_voltage,:	pointer to a structure that contains new voltages to be applied

Returns

Nothing

37.6.32 static void Chip_LOWPOWER_SetLowPowerMode (LPC_LOWPOWER_T * *p_lowpower_cfg*) [inline], [static]

Parameters

p_lowpower_-cfg,:	pointer to a structure that contains all low power mode parameters

Returns

Nothing

37.6.33 static void Chip_LOWPOWER_ChipSoftwareReset (void) [inline], [static]

Power down the flash then perform the full chip reset as POR or Watchdog do, The reset includes JTAG debugger, Digital units and Analog modules. Use the Software reset bit in PMC

Returns

Nothing

37.6.34 static void Chip_LOWPOWER_ArmSoftwareReset (void) [inline], [static]

Power down the flash then perform the Full chip reset as POR or Watchdog, The reset includes the digital units but excludes the JTAG debugger, and the analog modules. Use the system reset bit in PMC and ARM reset

Returns

Nothing

37.6.35 static int MPU_pSectorGrantAccessRights (uint32_t *addr,* size_t *sz,* MPU_reg_settings_t * *save_rule*) [inline], [static]

Note: The pSector region is 'special' because counter intuitively it requires Write access in order to be able to read from it using the flash controller indirect method. The previosuly applied policy. pSector region is protected under rule 7 (highest precedence). The previous rule 7 is saved in RAM before changing it.

Parameters

addr,:	address of area to grant access to.
sz,:	size in number of bytes of area.
save_rule,:	save a copy of previous rule

Returns

-1 if failure, if succesful return the size of the region.

37.6.36 static int MPU_pSectorWithdrawAccessRights (MPU_reg_settings_t * *save_rule*) [inline], [static]

The pSector region is 'special' because counter intuitively it requires Write access in order to be able to read from it using the flash controller indirect method.

Parameters

save_rule,:	pointer on RAM MPU_reg_settings_t structure saved by MPU_pSectorGrantAccess-Rights used to restore previous settings of region 7 and restrict access to pSector.

Returns

-1 if failure, if succesful return the size of the region.

37.6.37 static void MPU_GetCurrentSettings (MPU_Settings_t * *settings*) [inline], [static]

Parameters

settings,:	pointer of structure to receive the MPU register valkues.

Returns

none

37.6.38 static int MPU_AllocateRegionDesc (void) [inline], [static]

Checks if rules have their RASR enable bit set. This for the application to find free riules that were left unused by the ROM code. Implicitly MPU_ClearRegionSetting releases an allocate rule.

Parameters

none	

Returns

-1 if none free, value between 1..4 if succesful.

37.6.39 static uint32_t pmc_reset_get_cause (void) [inline], [static]

Returns

Reset cause value.

Return values

0x1	POR - The last chip reset was caused by a Power On Reset.
0x2	PADRESET - The last chip reset was caused by a Pad Reset.
0x4	BODRESET - The last chip reset was caused by a Brown Out Detector.
0x8	SYSTEMRESET - The last chip reset was caused by a System Reset requested by the ARM CPU.

0x10	WDTRESET - The last chip reset was caused by the Watchdog Timer.
0x20	WAKEUPIORESET - The last chip reset was caused by a Wake-up I/O (GPIO or internal NTAG FD INT).
0x40	WAKEUPPWDNRESET - The last CPU reset was caused by a Wake-up from Power down (many sources possible: timer, IO, ...).
0x80	SWRRESET - The last chip reset was caused by a Software.

37.6.40 static void pmc_reset_clear_cause (uint32_t *mask*) [inline], [static]

Parameters

mask	The mask of reset cause which you want to clear.

Returns

none

37.6.41 static psector_write_status_t psector_WriteUpdatePage (psector_partition_id_t *part_index,* psector_page_t ∗ *page*) [inline], [static]

The actual write to the partition will be effective after a reset only. Among other checks, the page must have a correct magic, a correct checksum

Parameters

part_index,:	PSECTOR_PAGE0_PART or PFLASH_PAGE0_PART.
page,:	psector_page_t RAM buffer to be written to update page

Returns

status code see @ psector_write_status_t.

37.6.42 static void psector_EraseUpdate (void) [inline], [static]

The actual write to the partition will be effective after a reset only.

Parameters

part_index,:	PSECTOR_PAGE0_PART or PFLASH_PAGE0_PART.
page,:	psector_page_t RAM buffer to be written to update page

Returns

status code see @ psector_write_status_t.

37.6.43 static psector_page_state_t psector_ReadData (psector_partition_id_t *part_index*, int *page_number*, uint32_t *offset*, uint32_t *size*, void * *data*) [inline], [static]

Parameters

part_index,:	PSECTOR_PAGE0_PART or PFLASH_PAGE0_PART.
page_number,:	necessarily 0 since partitions now contain 1 single page.
offset,:	offset of data from which data is to be read
size,:	number of bytes to be read
data,:	pointer on RAM buffer used to copy retrived data.

Returns

status code see @ psector_page_state_t if PAGE_STATE_DEGRADED or PAGE_STATE_OK, data is available. if PAGE_STATE_ERROR or PAGE_STATE_BLANK, no data was read

37.6.44 static uint32_t psector_CalculateChecksum (psector_page_t * *psector_page*) [inline], [static]

It is essential to recalculate the checksum when performing a psector page update, failing to update this field, the write operation would be rejected.

Parameters

psector_page,:	pointer on page over which computation is required.

Returns

checksum value to be checked or to replace checksum field of psector header

37.6.45 static uint64_t psector_Read_CustomerId (void) [inline], [static]

Parameters

none	

Returns

CustomerId on 64 bit word

37.6.46 static int psector_Read_RomPatchInfo (uint32_t * *patch_region_sz,* uint32_t * *patch_region_addr,* uint32_t * *patch_checksum,* uint32_t * *patch_checksum_valid*) [inline], [static]

Parameters

patch_region_-sz,:	pointer on unsigned long to return ROM patch size
patch_region_-addr,:	pointer on unsigned long to return ROM patch address
patch_-checksum,:	pointer on unsigned long to return ROM patch checksum value
patch_-checksum_-valid,:	pointer on unsigned long to return ROM patch checksum validity (0..1)

Returns

-1 if erro is found (any of the input parameters is NULL) or PFLASH is unreadable.

37.6.47 static uint16_t psector_Read_ImgAuthLevel (void) [inline], [static]

Parameters

none.	

Returns

AUH_NONE if PFLASH unreadable, or the image_authentication_level field value if readable.

37.6.48 static uint32_t psector_Read_AppSearchGranularity (void) [inline], [static]

Parameters

none.	

Returns

0 if PFLASH unreadable, or the app_search_granularity field value if not 0 or 4096 if 0.

37.6.49 static uint32_t psector_Read_QspiAppSearchGranularity (void) [inline], [static]

Parameters

none.	

Returns

0 if PFLASH unreadable, or the qspi_app_search_granularity field value.

37.6.50 static uint64_t psector_Read_DeviceId (void) [inline], [static]

Parameters

none.	

Returns

0 if PFLASH unreadable, or the device_id field value.

37.6.51 static int psector_Read_UnlockKey (int * *valid*, uint8_t *key[256]*, bool *raw*) [inline], [static]

Parameters

valid,:	pointer on int to store validity of key (unlock_key_valid field)
key,:	pointer on 256 byte storage to receive the key read from PFLASH
raw,:	raw if raw is not requested (0), the key is deciphered using the internal AES fused key.

Returns

-1 if read error occurred, 0 otherwise

37.6.52 static int psector_Read_ISP_protocol_key (uint8_t *key[16]*) [inline], [static]

Parameters

key,:	pointer on 16 byte storage to receive the key read from PFLASH.

Returns

-1 if read error occurred, 0 otherwise

37.6.53 static uint64_t psector_ReadIeee802_15_4_MacId1 (void) [inline], [static]

Parameters

none.	

Returns

64 bit word 0 if field unreadable, otherwise MAC address contained in ieee_mac_id1 field.

37.6.54 static uint64_t psector_ReadIeee802_15_4_MacId2 (void) [inline], [static]

Parameters

none.	

Returns

64 bit word 0 if field unreadable, otherwise MAC address contained in ieee_mac_id2 field.

37.6.55 static uint64_t psector_Read_MinDeviceId (void) [inline], [static]

Parameters

none.	

Returns

0 if PFLASH unreadable, otherwise min_device_id field content.

37.6.56 static uint64_t psector_Read_MaxDeviceId (void) [inline], [static]

Parameters

none.	

Returns

0 if PFLASH unreadable, otherwise max_device_id field content.

37.6.57 static uint32_t psector_Read_MinVersion (void) [inline], [static]

Parameters

none.	

Returns

if PAGE0 unreadable, otherwise MinVersion field content.

37.6.58 static psector_write_status_t psector_SetEscoreImageData (uint32_t *image_addr,* uint32_t *min_version*) [inline], [static]

Parameters

image_addr,:	32 bit value to be written to SelectImageAddress.
min_version,:	32 bit value to be written to MinVersion.

Returns

psector_write_status_t status of operation see

37.6.59 static psector_page_state_t psector_ReadEscoreImageData (uint32_t * *image_addr*, uint32_t * *min_version*) [inline], [static]

Parameters

image_addr,:	pointer on 32 bit word to receive SelectImageAddress value.
min_versionm,:	pointer on 32 bit word to receive SelectImageAddress value.

Returns

-1 if read error occurred, 0 otherwise

37.6.60 static int psector_Read_ImagePubKey (int * *valid*, uint8_t *key[256]*, bool *raw*) [inline], [static]

Parameters

valid,:	pointer on int to store validity of key (img_pk_valid field)
key,:	pointer on 256 byte storage to receive the key read from PAGE0
raw,:	raw if raw is not requested (0), the key is deciphered using the internal AES fused key.

Returns

-1 if read error occurred, 0 otherwise

37.6.61 static uint32_t secure_VerifySignature (uint8_t * *hash*, const uint8_t * *signature*, const uint32_t * *key*) [inline], [static]

Verify a signature by encrypting it using the provided public key and validating the output matches the provided hash resulting from the SHA-256

Parameters

hash,:	pointer on computed SHA-256 hash
signature,:	pointer on RSA-2048 signature to be checked
key,:	pointer on public key

Returns

1: if correct, 0: otherwise.

37.6.62 static uint32_t secure_VerifyBlock (uint8_t * *start,* uint32_t *length,* const uint32_t * *key,* const uint8_t * *signature*) [inline], [static]

Verify a data block with appended signature. Computes the SHA-256 hash. calls see

Parameters

start,:	pointer on start of data block
length,:	length of data block
key,:	pointer on public key
signature,:	pointer on RSA-2048 signature to be checked

Returns

1: if correct, 0: otherwise.

37.6.63 static uint32_t secure_VerifyCertificate (const IMAGE_CERT_T * *certificate,* const uint32_t * *key,* const uint8_t * *cert_signature*) [inline], [static]

Verify certificate is valid for this device and is authentic. Certificate is checked for validity against customer and device ID stored in PFLASH.

Parameters

certificate,:	pointer on computed SHA-256 hash
key,:	pointer on public key
cert_-signature,:	pointer on RSA-2048 signature to be checked

Returns

1: if certificate is valid, 0: otherwise.

37.6.64 static uint32_t secure_VerifyImage (uint32_t *image_addr,* const IMAGE_CERT_T * *root_cert*) [inline], [static]

The function retrieves the certificate pointed by the boot block certificate offset field. If present it has to be verified using the root certificate.

Parameters

image_addr,:	pointer on start of image to be checked.
root_cert,:	pointer on root certificate (that contains a public key). If the root certificate is present the key is gotten from it.

Returns

1: if certificate is valid, 0: otherwise.

37.7 Variable Documentation

37.7.1 uint8_t ISP_MEM_INFO_T::access

- bit 0: Read access
- bit 1: Write access
- bit 2: Erase right
- bit 3: Erase all right
- bit 4: blank check right A value of 0 denotes that access is closed

37.7.2 uint8_t ISP_STATE_T::mode

- 0x01: Default ISP mode
- 0x7f: unlock mode
- 0x80 or higher: treated by extension function if any

37.7.3 uint32_t psector_page_data_t::MinVersion

offset 0x30

37.7.4 uint32_t { ... } ::MinVersion

offset 0x30

37.7.5 uint32_t psector_page_data_t::rom_patch_region_addr

A value outside of the address range used to store the ROM patch binary shall be deemed invalid

37.7.6 uint32_t { ... } ::rom_patch_region_addr

A value outside of the address range used to store the ROM patch binary shall be deemed invalid

37.7.7 uint16_t psector_page_data_t::application_flash_sz

0 is interpreted as maximum (640). This is intended to provide an alternative way of restricting the flash size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

37.7.8 uint16_t { ... } ::application_flash_sz

0 is interpreted as maximum (640). This is intended to provide an alternative way of restricting the flash size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

37.7.9 uint16_t psector_page_data_t::ram1_bank_sz

This is intended to provide an alternative way of restricting the RAM size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

37.7.10 uint16_t { ... } ::ram1_bank_sz

This is intended to provide an alternative way of restricting the RAM size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

37.7.11 uint32_t psector_page_data_t::app_search_granularity

Value of 0 shall be equated to 4096. Other values are to be used directly; configurations that are not using hardware remapping do not require hard restrictions

37.7.12 uint32_t { ... } ::app_search_granularity

Value of 0 shall be equated to 4096. Other values are to be used directly; configurations that are not using hardware remapping do not require hard restrictions

37.7.13 uint8_t psector_page_data_t::unlock_key[256]

Stored encrypted, using the AES key in eFuse

37.7.14 uint8_t { ... } ::unlock_key[256]

Stored encrypted, using the AES key in eFuse

37.8 Sha_algorithm_level_api

37.8.1 Overview

SHA Functional Operation

- status_t SHA_Init (SHA_Type *base, sha_ctx_t *ctx, sha_algo_t algo)
  Initialize HASH context.
- status_t SHA_Update (SHA_Type *base, sha_ctx_t *ctx, const uint8_t *message, size_t messageSize)
  Add data to current HASH.
- status_t SHA_Finish (SHA_Type *base, sha_ctx_t *ctx, uint8_t *output, size_t *outputSize)
  Finalize hashing.
- void SHA_ClkInit (SHA_Type *base)
  Start SHA clock.
- void SHA_ClkDeinit (SHA_Type *base)
  Stop SHA clock.

37.8.2 Function Documentation

37.8.2.1 status_t SHA_Init (SHA_Type * *base*, sha_ctx_t * *ctx*, sha_algo_t *algo*)

This function initializes new hash context.

Parameters

	base	SHA peripheral base address
out	*ctx*	Output hash context
	algo	Underlaying algorithm to use for hash computation. Either SHA-1 or SHA-256.

Returns

Status of initialization

37.8.2.2 status_t SHA_Update (SHA_Type * *base*, sha_ctx_t * *ctx*, const uint8_t * *message*, size_t *messageSize*)

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

Parameters

	base	SHA peripheral base address
in,out	*ctx*	HASH context
	message	Input message
	messageSize	Size of input message in bytes

Returns

Status of the hash update operation

37.8.2.3 status_t SHA_Finish (SHA_Type * *base,* sha_ctx_t * *ctx,* uint8_t * *output,* size_t * *outputSize*)

Outputs the final hash and erases the context. SHA-1 or SHA-256 padding bits are automatically added by this function.

Parameters

	base	SHA peripheral base address
in,out	*ctx*	HASH context
out	*output*	Output hash data
in,out	*outputSize*	On input, determines the size of bytes of the output array. On output, tells how many bytes have been written to output.

Returns

Status of the hash finish operation

37.8.2.4 void SHA_ClkInit (SHA_Type * *base*)

Start SHA clock

Parameters

base	SHA peripheral base address

37.8.2.5 void SHA_ClkDeinit (SHA_Type * *base*)

Stop SHA clock

Parameters

base	SHA peripheral base address

Chapter 38
GenericList

38.1 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.

38.2 Data Structure Documentation

38.2.1 struct list_label_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

38.2.2 struct list_element_t

Data Fields

- struct list_element_tag ∗ next
  next list element
- struct list_element_tag ∗ prev
  previous list element
- struct list_label ∗ list
  pointer to the list

38.3 Enumeration Type Documentation

38.3.1 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.

38.4 Function Documentation

38.4.1 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.

38.4.2 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.

38.4.3 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.

38.4.4 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.

38.4.5 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.

38.4.6 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.

38.4.7 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.

38.4.8 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.

38.4.9 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.

38.4.10 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.

38.4.11 uint32_t LIST_GetSize (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

Current	size of the list.

38.4.12 uint32_t LIST_GetAvailableSize (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

Available	size of the list.

Chapter 39
UART_Adapter

39.1 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.

39.2 Data Structure Documentation

39.2.1 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.

39.2.1.0.0.2 Field Documentation

39.2.1.0.0.2.1 uint8_t hal_uart_config_t::instance

Invalid instance value will cause initialization failure.

39.2.2 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.

39.2.2.0.0.3 Field Documentation

39.2.2.0.0.3.1 uint8_t* hal_uart_transfer_t::data

39.2.2.0.0.3.2 size_t hal_uart_transfer_t::dataSize

39.3 Macro Definition Documentation

39.3.1 #define HAL_UART_TRANSFER_MODE (0U)

(0 - disable, 1 - enable)

39.4 Typedef Documentation

39.4.1 typedef void(* hal_uart_transfer_callback_t)(hal_uart_handle_t handle, hal_uart_status_t status, void *callbackParam)

39.5 Enumeration Type Documentation

39.5.1 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.

39.5.2 enum hal_uart_parity_mode_t

Enumerator

kHAL_UartParityDisabled Parity disabled.
kHAL_UartParityEven Parity even enabled.
kHAL_UartParityOdd Parity odd enabled.

39.5.3 enum hal_uart_stop_bit_count_t

Enumerator

kHAL_UartOneStopBit One stop bit.
kHAL_UartTwoStopBit Two stop bits.

39.6 Function Documentation

39.6.1 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

39.6.2 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_UartSuccess	UART de-initialization succeed

39.6.3 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_UartTransferReceiveNonBlocking 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_UartParityError	A parity error occurred while receiving data.
kStatus_HAL_UartSuccess	Successfully received all data.

39.6.4 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