Document Number: MCUXSDKAPIRM
Rev 2.12.0
Jul 2022

MCUXpresso SDK API Reference Manual

NXP Semiconductors

Contents

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Section No.	Title	Page No.

Chapter 1
Introduction

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

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

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

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

MCUXpresso SDK Folder Structure

Chapter 2
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: 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 3
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

Kinetis_SDK_Block_Diagram.jpg

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

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

Xpresso SDK drivers, users can redefine the first layer weak function and implement their own interrupt handler functions to suit their implementation.

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

Feature Header Files

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

Application

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

Chapter 4
ACMP: Analog Comparator Driver

4.1 Overview

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

The ACMP driver is created to help the user operate the ACMP module better. This driver can be considered as a basic comparator with advanced features. The APIs for basic comparator can make the C-MP work as a general comparator, which compares the two input channel's voltage and creates the output of the comparator result immediately. The APIs for advanced feature can be used as the plug-in function based on the basic comparator, and can provide more ways to process the comparator's output.

4.2 Typical use case

4.2.1 Normal Configuration

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

4.2.2 Interrupt Configuration

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

4.2.3 Round robin Configuration

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

Data Structures

- struct acmp_config_t
  Configuration for ACMP. More...
- struct acmp_channel_config_t
  Configuration for channel. More...
- struct acmp_filter_config_t
  Configuration for filter. More...
- struct acmp_dac_config_t
  Configuration for DAC. More...
- struct acmp_round_robin_config_t
  Configuration for round robin mode. More...

Macros

- #define CMP_C0_CFx_MASK (CMP_C0_CFR_MASK | CMP_C0_CFF_MASK)
  The mask of status flags cleared by writing 1.

Enumerations

- enum _acmp_interrupt_enable {
  kACMP_OutputRisingInterruptEnable = (1U << 0U),
  kACMP_OutputFallingInterruptEnable = (1U << 1U),
  kACMP_RoundRobinInterruptEnable = (1U << 2U) }
  Interrupt enable/disable mask.
- enum _acmp_status_flags {
  kACMP_OutputRisingEventFlag = CMP_C0_CFR_MASK,
  kACMP_OutputFallingEventFlag = CMP_C0_CFF_MASK,
  kACMP_OutputAssertEventFlag = CMP_C0_COUT_MASK }
  Status flag mask.
- enum acmp_offset_mode_t {
  kACMP_OffsetLevel0 = 0U,
  kACMP_OffsetLevel1 = 1U }
  Comparator hard block offset control.
- enum acmp_hysteresis_mode_t {
  kACMP_HysteresisLevel0 = 0U,
  kACMP_HysteresisLevel1 = 1U,
  kACMP_HysteresisLevel2 = 2U,
  kACMP_HysteresisLevel3 = 3U }
  Comparator hard block hysteresis control.
- enum acmp_reference_voltage_source_t {
  kACMP_VrefSourceVin1 = 0U,
  kACMP_VrefSourceVin2 = 1U }
  CMP Voltage Reference source.
- enum acmp_port_input_t {
  kACMP_PortInputFromDAC = 0U,
  kACMP_PortInputFromMux = 1U }
  Port input source.
- enum acmp_fixed_port_t {
  kACMP_FixedPlusPort = 0U,
  kACMP_FixedMinusPort = 1U }
  Fixed mux port.

Driver version

- #define FSL_ACMP_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 6U))
  ACMP driver version 2.0.6.

Initialization and deinitialization

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

Initializes the ACMP.
- void ACMP_Deinit (CMP_Type ∗base)
  Deinitializes the ACMP.
- void ACMP_GetDefaultConfig (acmp_config_t ∗config)
  Gets the default configuration for ACMP.

Basic Operations

- void ACMP_Enable (CMP_Type ∗base, bool enable)
  Enables or disables the ACMP.
- void ACMP_SetChannelConfig (CMP_Type ∗base, const acmp_channel_config_t ∗config)
  Sets the channel configuration.

Advanced Operations

- void ACMP_EnableDMA (CMP_Type ∗base, bool enable)
  Enables or disables DMA.
- void ACMP_EnableWindowMode (CMP_Type ∗base, bool enable)
  Enables or disables window mode.
- void ACMP_SetFilterConfig (CMP_Type ∗base, const acmp_filter_config_t ∗config)
  Configures the filter.
- void ACMP_SetDACConfig (CMP_Type ∗base, const acmp_dac_config_t ∗config)
  Configures the internal DAC.
- void ACMP_SetRoundRobinConfig (CMP_Type ∗base, const acmp_round_robin_config_t ∗config)
  Configures the round robin mode.
- void ACMP_SetRoundRobinPreState (CMP_Type ∗base, uint32_t mask)
  Defines the pre-set state of channels in round robin mode.
- static uint32_t ACMP_GetRoundRobinStatusFlags (CMP_Type ∗base)
  Gets the channel input changed flags in round robin mode.
- void ACMP_ClearRoundRobinStatusFlags (CMP_Type ∗base, uint32_t mask)
  Clears the channel input changed flags in round robin mode.
- static uint32_t ACMP_GetRoundRobinResult (CMP_Type ∗base)
  Gets the round robin result.

Interrupts

- void ACMP_EnableInterrupts (CMP_Type ∗base, uint32_t mask)
  Enables interrupts.
- void ACMP_DisableInterrupts (CMP_Type ∗base, uint32_t mask)
  Disables interrupts.

Status

- uint32_t ACMP_GetStatusFlags (CMP_Type ∗base)
  Gets status flags.
- void ACMP_ClearStatusFlags (CMP_Type ∗base, uint32_t mask)
  Clears status flags.

4.3 Data Structure Documentation

4.3.1 struct acmp_config_t

Data Fields

- acmp_offset_mode_t offsetMode
  Offset mode.
- acmp_hysteresis_mode_t hysteresisMode
  Hysteresis mode.
- bool enableHighSpeed
  Enable High Speed (HS) comparison mode.
- bool enableInvertOutput
  Enable inverted comparator output.
- bool useUnfilteredOutput
  Set compare output(COUT) to equal COUTA(true) or COUT(false).
- bool enablePinOut
  The comparator output is available on the associated pin.

Field Documentation

(1) acmp_offset_mode_t acmp_config_t::offsetMode

(2) acmp_hysteresis_mode_t acmp_config_t::hysteresisMode

(3) bool acmp_config_t::enableHighSpeed

(4) bool acmp_config_t::enableInvertOutput

(5) bool acmp_config_t::useUnfilteredOutput

(6) bool acmp_config_t::enablePinOut

4.3.2 struct acmp_channel_config_t

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

Data Fields

- acmp_port_input_t positivePortInput
  Input source of the comparator's positive port.
- uint32_t plusMuxInput
  Plus mux input channel(0~7).
- acmp_port_input_t negativePortInput
  Input source of the comparator's negative port.
- uint32_t minusMuxInput
  Minus mux input channel(0~7).

Field Documentation

(1) acmp_port_input_t acmp_channel_config_t::positivePortInput

(2) uint32_t acmp_channel_config_t::plusMuxInput

(3) acmp_port_input_t acmp_channel_config_t::negativePortInput

(4) uint32_t acmp_channel_config_t::minusMuxInput

4.3.3 struct acmp_filter_config_t

Data Fields

- bool enableSample
  Using external SAMPLE as sampling clock input, or using divided bus clock.
- uint32_t filterCount
  Filter Sample Count.
- uint32_t filterPeriod
  Filter Sample Period.

Field Documentation

(1) bool acmp_filter_config_t::enableSample

(2) uint32_t acmp_filter_config_t::filterCount

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

(3) uint32_t acmp_filter_config_t::filterPeriod

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

4.3.4 struct acmp_dac_config_t

Data Fields

- acmp_reference_voltage_source_t referenceVoltageSource
  Supply voltage reference source.
- uint32_t DACValue
  Value for DAC Output Voltage.

Field Documentation

(1) acmp_reference_voltage_source_t acmp_dac_config_t::referenceVoltageSource

(2) uint32_t acmp_dac_config_t::DACValue

Available range is 0-255.

4.3.5 struct acmp_round_robin_config_t

Data Fields

- acmp_fixed_port_t fixedPort
  Fixed mux port.
- uint32_t fixedChannelNumber
  Indicates which channel is fixed in the fixed mux port.
- uint32_t checkerChannelMask
  Mask of checker channel index.
- uint32_t sampleClockCount
  Specifies how many round-robin clock cycles(0∼3) later the sample takes place.
- uint32_t delayModulus
  Comparator and DAC initialization delay modulus.

Field Documentation

(1) acmp_fixed_port_t acmp_round_robin_config_t::fixedPort

(2) uint32_t acmp_round_robin_config_t::fixedChannelNumber

(3) uint32_t acmp_round_robin_config_t::checkerChannelMask

Available range is channel0:0x01 to channel7:0x80 for round-robin checker.

(4) uint32_t acmp_round_robin_config_t::sampleClockCount

(5) uint32_t acmp_round_robin_config_t::delayModulus

4.4 Macro Definition Documentation

4.4.1 #define FSL_ACMP_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 6U))

4.4.2 #define CMP_C0_CFx_MASK (CMP_C0_CFR_MASK | CMP_C0_CFF_MASK)

4.5 Enumeration Type Documentation

4.5.1 enum _acmp_interrupt_enable

Enumerator

kACMP_OutputRisingInterruptEnable Enable the interrupt when comparator outputs rising.
kACMP_OutputFallingInterruptEnable Enable the interrupt when comparator outputs falling.
kACMP_RoundRobinInterruptEnable Enable the Round-Robin interrupt.

4.5.2 enum _acmp_status_flags

Enumerator

kACMP_OutputRisingEventFlag Rising-edge on compare output has occurred.
kACMP_OutputFallingEventFlag Falling-edge on compare output has occurred.
kACMP_OutputAssertEventFlag Return the current value of the analog comparator output.

4.5.3 enum acmp_offset_mode_t

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

Enumerator

kACMP_OffsetLevel0 The comparator hard block output has level 0 offset internally.
kACMP_OffsetLevel1 The comparator hard block output has level 1 offset internally.

4.5.4 enum acmp_hysteresis_mode_t

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

Enumerator

kACMP_HysteresisLevel0 Offset is level 0 and Hysteresis is level 0.
kACMP_HysteresisLevel1 Offset is level 0 and Hysteresis is level 1.
kACMP_HysteresisLevel2 Offset is level 0 and Hysteresis is level 2.
kACMP_HysteresisLevel3 Offset is level 0 and Hysteresis is level 3.

4.5.5 enum acmp_reference_voltage_source_t

Enumerator

kACMP_VrefSourceVin1 Vin1 is selected as resistor ladder network supply reference Vin.
kACMP_VrefSourceVin2 Vin2 is selected as resistor ladder network supply reference Vin.

4.5.6 enum acmp_port_input_t

Enumerator

kACMP_PortInputFromDAC Port input from the 8-bit DAC output.
kACMP_PortInputFromMux Port input from the analog 8-1 mux.

4.5.7 enum acmp_fixed_port_t

Enumerator

kACMP_FixedPlusPort Only the inputs to the Minus port are swept in each round.
kACMP_FixedMinusPort Only the inputs to the Plus port are swept in each round.

4.6 Function Documentation

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

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

Parameters

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

4.6.2 void ACMP_Deinit (CMP_Type ∗ *base*)

Parameters

base	ACMP peripheral base address.

4.6.3 void ACMP_GetDefaultConfig (acmp_config_t ∗ *config*)

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

Example:

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

Parameters

config	Pointer to ACMP configuration structure.

4.6.4 void ACMP_Enable (CMP_Type ∗ *base*, bool *enable*)

Parameters

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

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

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

Example:

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

Parameters

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

4.6.6 void ACMP_EnableDMA (CMP_Type ∗ *base*, bool *enable*)

Parameters

base	ACMP peripheral base address.

enable	True to enable DMA.

4.6.7 void ACMP_EnableWindowMode (CMP_Type ∗ *base,* bool *enable*)

Parameters

base	ACMP peripheral base address.
enable	True to enable window mode.

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

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

Example:

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

Parameters

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

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

Example:

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

Parameters

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

4.6.10 void ACMP_SetRoundRobinConfig (CMP_Type ∗ *base*, const acmp_round_robin_config_t ∗ *config*)

Example:

```
acmp_round_robin_config_t configStruct = {0};
configStruct.fixedPort = kACMP_FixedPlusPort;
configStruct.fixedChannelNumber = 3U;
configStruct.checkerChannelMask = 0xF7U;
configStruct.sampleClockCount = 0U;
configStruct.delayModulus = 0U;
ACMP_SetRoundRobinConfig(CMP0, &configStruct);
```

Parameters

base	ACMP peripheral base address.
config	Pointer to round robin mode configuration structure. "NULL" is for disabling the feature.

4.6.11 void ACMP_SetRoundRobinPreState (CMP_Type ∗ *base*, uint32_t *mask*)

Note: The pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So get-round-robin-result can't return the same value as the value are set by pre-state.

Parameters

base	ACMP peripheral base address.
mask	Mask of round robin channel index. Available range is channel0:0x01 to channel7-:0x80.

4.6.12 static uint32_t ACMP_GetRoundRobinStatusFlags (CMP_Type ∗ *base*) [inline], [static]

Parameters

base	ACMP peripheral base address.

Returns

Mask of channel input changed asserted flags. Available range is channel0:0x01 to channel7:0x80.

4.6.13 void ACMP_ClearRoundRobinStatusFlags (CMP_Type * *base*, uint32_t *mask*)

Parameters

base	ACMP peripheral base address.
mask	Mask of channel index. Available range is channel0:0x01 to channel7:0x80.

4.6.14 static uint32_t ACMP_GetRoundRobinResult (CMP_Type * *base*) [inline], [static]

Note that the set-pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So ACMP_GetRoundRobinResult() can't return the same value as the value are set by ACMP_SetRoundRobinPreState.

Parameters

base	ACMP peripheral base address.

Returns

Mask of round robin channel result. Available range is channel0:0x01 to channel7:0x80.

4.6.15 void ACMP_EnableInterrupts (CMP_Type * *base*, uint32_t *mask*)

Parameters

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

4.6.16 void ACMP_DisableInterrupts (CMP_Type * *base*, uint32_t *mask*)

Parameters

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

4.6.17 uint32_t ACMP_GetStatusFlags (CMP_Type * *base*)

Parameters

base	ACMP peripheral base address.

Returns

Status flags asserted mask. See "_acmp_status_flags".

4.6.18 void ACMP_ClearStatusFlags (CMP_Type * *base*, uint32_t *mask*)

Parameters

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

Chapter 5
ADC12: Analog-to-Digital Converter

5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Analog-to-Digital Converter (ADC12) module of MCUXpresso SDK devices.

The ADC12 driver is created to help the user better operate the ADC12 module. This driver can be considered a basic analog-to-digital converter with advanced features. The APIs for basic operations can make the ADC12 work as a general converter, which can convert the analog input to be a digital value. The APIs for advanced operations can be used as the plug-in function based on the basic operations. They can provide more ways to process the converter's conversion results, such DMA trigger, hardware compare, hardware average, and so on.

Note that channel 26 of ADC12 is connected to a internal temperature sensor of the module. If you want to get the best conversion result of the temperature value, set the field "sampleClockCount" in the structure "adc12_config_t" to be maximum value when you call the API "ADC12_Init()". This field indicates the sample time of the analog input signal. A longer sample time makes the conversion result of the analog input signal more stable and accurate.

5.2 Function groups

5.2.1 Initialization and deinitialization

This function group implement ADC12 initialization and deinitialization API.

5.2.2 Basic Operations

This function group implement basic ADC12 operation API.

5.2.3 Advanced Operations

This function group implement advanced ADC12 operation API.

5.3 Typical use case

5.3.1 Normal Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/adc12

5.3.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/adc12

Data Structures

- struct adc12_config_t
  Converter configuration. More...
- struct adc12_hardware_compare_config_t
  Hardware compare configuration. More...
- struct adc12_channel_config_t
  Channel conversion configuration. More...

Macros

- #define FSL_ADC12_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))
  ADC12 driver version.

Enumerations

- enum _adc12_channel_status_flags { kADC12_ChannelConversionCompletedFlag = ADC_SC1_-COCO_MASK }
  Channel status flags' mask.
- enum _adc12_status_flags {
  kADC12_ActiveFlag = ADC_SC2_ADACT_MASK,
  kADC12_CalibrationFailedFlag = (ADC_SC2_ADACT_MASK << 1U) }
  Converter status flags' mask.
- enum adc12_clock_divider_t {
  kADC12_ClockDivider1 = 0U,
  kADC12_ClockDivider2 = 1U,
  kADC12_ClockDivider4 = 2U,
  kADC12_ClockDivider8 = 3U }
  Clock divider for the converter.
- enum adc12_resolution_t {
  kADC12_Resolution8Bit = 0U,
  kADC12_Resolution12Bit = 1U,
  kADC12_Resolution10Bit = 2U }
  Converter's resolution.
- enum adc12_clock_source_t {
  kADC12_ClockSourceAlt0 = 0U,
  kADC12_ClockSourceAlt1 = 1U,
  kADC12_ClockSourceAlt2 = 2U,
  kADC12_ClockSourceAlt3 = 3U }
  Conversion clock source.
- enum adc12_reference_voltage_source_t {
  kADC12_ReferenceVoltageSourceVref = 0U,
  kADC12_ReferenceVoltageSourceValt = 1U }
  Reference voltage source.

- enum adc12_hardware_average_mode_t {
kADC12_HardwareAverageCount4 = 0U,
kADC12_HardwareAverageCount8 = 1U,
kADC12_HardwareAverageCount16 = 2U,
kADC12_HardwareAverageCount32 = 3U,
kADC12_HardwareAverageDisabled = 4U }
  Hardware average mode.
- enum adc12_hardware_compare_mode_t {
kADC12_HardwareCompareMode0 = 0U,
kADC12_HardwareCompareMode1 = 1U,
kADC12_HardwareCompareMode2 = 2U,
kADC12_HardwareCompareMode3 = 3U }
  Hardware compare mode.

Initialization

- void ADC12_Init (ADC_Type ∗base, const adc12_config_t ∗config)
  Initialize the ADC12 module.
- void ADC12_Deinit (ADC_Type ∗base)
  De-initialize the ADC12 module.
- void ADC12_GetDefaultConfig (adc12_config_t ∗config)
  Gets an available pre-defined settings for converter's configuration.

Basic Operations

- void ADC12_SetChannelConfig (ADC_Type ∗base, uint32_t channelGroup, const adc12_channel_config_t ∗config)
  Configure the conversion channel.
- static uint32_t ADC12_GetChannelConversionValue (ADC_Type ∗base, uint32_t channelGroup)
  Get the conversion value.
- uint32_t ADC12_GetChannelStatusFlags (ADC_Type ∗base, uint32_t channelGroup)
  Get the status flags of channel.

Advanced Operations

- status_t ADC12_DoAutoCalibration (ADC_Type ∗base)
  Automate the hardware calibration.
- static void ADC12_SetOffsetValue (ADC_Type ∗base, uint32_t value)
  Set the offset value for the conversion result.
- static void ADC12_SetGainValue (ADC_Type ∗base, uint32_t value)
  Set the gain value for the conversion result.
- static void ADC12_EnableHardwareTrigger (ADC_Type ∗base, bool enable)
  Enable of disable the hardware trigger mode.
- void ADC12_SetHardwareCompareConfig (ADC_Type ∗base, const adc12_hardware_compare_config_t ∗config)
  Configure the hardware compare mode.
- void ADC12_SetHardwareAverage (ADC_Type ∗base, adc12_hardware_average_mode_t mode)
  Set the hardware average mode.
- uint32_t ADC12_GetStatusFlags (ADC_Type ∗base)

Get the status flags of the converter.

5.4 Data Structure Documentation

5.4.1 struct adc12_config_t

Data Fields

- adc12_reference_voltage_source_t referenceVoltageSource
  Select the reference voltage source.
- adc12_clock_source_t clockSource
  Select the input clock source to converter.
- adc12_clock_divider_t clockDivider
  Select the divider of input clock source.
- adc12_resolution_t resolution
  Select the sample resolution mode.
- uint32_t sampleClockCount
  Select the sample clock count.
- bool enableContinuousConversion
  Enable continuous conversion mode.

Field Documentation

(1) adc12_reference_voltage_source_t adc12_config_t::referenceVoltageSource

(2) adc12_clock_source_t adc12_config_t::clockSource

(3) adc12_clock_divider_t adc12_config_t::clockDivider

(4) adc12_resolution_t adc12_config_t::resolution

(5) uint32_t adc12_config_t::sampleClockCount

Add its value may improve the stability of the conversion result.

(6) bool adc12_config_t::enableContinuousConversion

5.4.2 struct adc12_hardware_compare_config_t

Data Fields

- adc12_hardware_compare_mode_t hardwareCompareMode
  Select the hardware compare mode.
- int16_t value1
  Setting value1 for hardware compare mode.
- int16_t value2
  Setting value2 for hardware compare mode.

Field Documentation

(1) adc12_hardware_compare_mode_t adc12_hardware_compare_config_t::hardwareCompareMode

(2) int16_t adc12_hardware_compare_config_t::value1

(3) int16_t adc12_hardware_compare_config_t::value2

5.4.3 struct adc12_channel_config_t

Data Fields

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

Field Documentation

(1) uint32_t adc12_channel_config_t::channelNumber

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

(2) bool adc12_channel_config_t::enableInterruptOnConversionCompleted

5.5 Macro Definition Documentation

5.5.1 #define FSL_ADC12_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))

Version 2.0.6.

5.6 Enumeration Type Documentation

5.6.1 enum _adc12_channel_status_flags

Enumerator

kADC12_ChannelConversionCompletedFlag Conversion done.

5.6.2 enum _adc12_status_flags

Enumerator

kADC12_ActiveFlag Converter is active.
kADC12_CalibrationFailedFlag Calibration is failed.

5.6.3 enum adc12_clock_divider_t

Enumerator

kADC12_ClockDivider1 For divider 1 from the input clock to the module.
kADC12_ClockDivider2 For divider 2 from the input clock to the module.
kADC12_ClockDivider4 For divider 4 from the input clock to the module.
kADC12_ClockDivider8 For divider 8 from the input clock to the module.

5.6.4 enum adc12_resolution_t

Enumerator

kADC12_Resolution8Bit 8 bit resolution.
kADC12_Resolution12Bit 12 bit resolution.
kADC12_Resolution10Bit 10 bit resolution.

5.6.5 enum adc12_clock_source_t

Enumerator

kADC12_ClockSourceAlt0 Alternate clock 1 (ADC_ALTCLK1).
kADC12_ClockSourceAlt1 Alternate clock 2 (ADC_ALTCLK2).
kADC12_ClockSourceAlt2 Alternate clock 3 (ADC_ALTCLK3).
kADC12_ClockSourceAlt3 Alternate clock 4 (ADC_ALTCLK4).

5.6.6 enum adc12_reference_voltage_source_t

Enumerator

kADC12_ReferenceVoltageSourceVref For external pins pair of VrefH and VrefL.
kADC12_ReferenceVoltageSourceValt For alternate reference pair of ValtH and ValtL.

5.6.7 enum adc12_hardware_average_mode_t

Enumerator

kADC12_HardwareAverageCount4 For hardware average with 4 samples.
kADC12_HardwareAverageCount8 For hardware average with 8 samples.
kADC12_HardwareAverageCount16 For hardware average with 16 samples.
kADC12_HardwareAverageCount32 For hardware average with 32 samples.
kADC12_HardwareAverageDisabled Disable the hardware average feature.

5.6.8 enum adc12_hardware_compare_mode_t

Enumerator

kADC12_HardwareCompareMode0 x < value1.
kADC12_HardwareCompareMode1 x > value1.
kADC12_HardwareCompareMode2 if value1 <= value2, then x < value1 || x > value2; else, value1 > x > value2.
kADC12_HardwareCompareMode3 if value1 <= value2, then value1 <= x <= value2; else x >= value1 || x <= value2.

5.7 Function Documentation

5.7.1 void ADC12_Init (ADC_Type * *base*, const adc12_config_t * *config*)

Parameters

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

5.7.2 void ADC12_Deinit (ADC_Type * *base*)

Parameters

base	ADC12 peripheral base address.

5.7.3 void ADC12_GetDefaultConfig (adc12_config_t * *config*)

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

Example:

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

Parameters

config	Pointer to "adc12_config_t" structure.

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

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

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

Parameters

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

5.7.5 static uint32_t ADC12_GetChannelConversionValue (ADC_Type * *base,* uint32_t *channelGroup*) [inline], [static]

Parameters

base	ADC12 peripheral base address.

channelGroup	Channel group index.

Returns

Conversion value.

5.7.6 uint32_t ADC12_GetChannelStatusFlags (ADC_Type ∗ *base,* uint32_t *channelGroup*)

Parameters

base	ADC12 peripheral base address.
channelGroup	Channel group index.

Returns

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

5.7.7 status_t ADC12_DoAutoCalibration (ADC_Type ∗ *base*)

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

Parameters

base	ADC12 peripheral base address.

Return values

kStatus_Success	Calibration is done successfully.
kStatus_Fail	Calibration is failed.

5.7.8 static void ADC12_SetOffsetValue (ADC_Type ∗ *base,* uint32_t *value*) [inline], [static]

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

Parameters

base	ADC12 peripheral base address.
value	Offset value.

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

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

Parameters

base	ADC12 peripheral base address.
value	Gain value.

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

Parameters

base	ADC12 peripheral base address.
enable	Switcher of hardware trigger feature. "true" means to enable, "false" means not.

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

The hardware compare mode provides a way to process the conversion result automatically by hardware. Only the result in compare range is available. To compare the range, see "adc12_hardware_compare_-mode_t", or the reference manual document for more detailed information.

Parameters

base	ADC12 peripheral base address.

config	Pointer to "adc12_hardware_compare_config_t" structure. Pass "NULL" to disable the feature.

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

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

Parameters

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

5.7.13 uint32_t ADC12_GetStatusFlags (ADC_Type * *base*)

Parameters

base	ADC12 peripheral base address.

Returns

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

Chapter 6
CRC: Cyclic Redundancy Check Driver

6.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Cyclic Redundancy Check (CRC) module of MCUXpresso SDK devices.

The cyclic redundancy check (CRC) module generates 16/32-bit CRC code for error detection. The CRC module also provides a programmable polynomial, seed, and other parameters required to implement a 16-bit or 32-bit CRC standard.

6.2 CRC Driver Initialization and Configuration

CRC_Init() function enables the clock gate for the CRC module in the SIM module and fully (re-)configures the CRC module according to the configuration structure. The seed member of the configuration structure is the initial checksum for which new data can be added to. When starting a new checksum computation, the seed is set to the initial checksum per the CRC protocol specification. For continued checksum operation, the seed is set to the intermediate checksum value as obtained from previous calls to CRC_Get16bitResult() or CRC_Get32bitResult() function. After calling the CRC_Init(), one or multiple CRC_WriteData() calls follow to update the checksum with data and CRC_Get16bitResult() or CRC_Get32bitResult() follow to read the result. The crcResult member of the configuration structure determines whether the CRC_Get16bitResult() or CRC_Get32bitResult() return value is a final checksum or an intermediate checksum. The CRC_Init() function can be called as many times as required allowing for runtime changes of the CRC protocol.

CRC_GetDefaultConfig() function can be used to set the module configuration structure with parameters for CRC-16/CCIT-FALSE protocol.

6.3 CRC Write Data

The CRC_WriteData() function adds data to the CRC. Internally, it tries to use 32-bit reads and writes for all aligned data in the user buffer and 8-bit reads and writes for all unaligned data in the user buffer. This function can update the CRC with user-supplied data chunks of an arbitrary size, so one can update the CRC byte by byte or with all bytes at once. Prior to calling the CRC configuration function CRC_Init() fully specifies the CRC module configuration for the CRC_WriteData() call.

6.4 CRC Get Checksum

The CRC_Get16bitResult() or CRC_Get32bitResult() function reads the CRC module data register. Depending on the prior CRC module usage, the return value is either an intermediate checksum or the final checksum. For example, for 16-bit CRCs the following call sequences can be used.

CRC_Init() / CRC_WriteData() / CRC_Get16bitResult() to get the final checksum.

CRC_Init() / CRC_WriteData() / ... / CRC_WriteData() / CRC_Get16bitResult() to get the final checksum.

CRC_Init() / CRC_WriteData() / CRC_Get16bitResult() to get an intermediate checksum.

CRC_Init() / CRC_WriteData() / ... / CRC_WriteData() / CRC_Get16bitResult() to get an intermediate checksum.

6.5 Comments about API usage in RTOS

If multiple RTOS tasks share the CRC module to compute checksums with different data and/or protocols, the following needs to be implemented by the user.

The triplets

CRC_Init() / CRC_WriteData() / CRC_Get16bitResult() or CRC_Get32bitResult()

The triplets are protected by the RTOS mutex to protect the CRC module against concurrent accesses from different tasks. This is an example. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crc

Data Structures

- struct crc_config_t
  CRC protocol configuration. More...

Macros

- #define CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT 1
  Default configuration structure filled by CRC_GetDefaultConfig().

Enumerations

- enum crc_bits_t {
  kCrcBits16 = 0U,
  kCrcBits32 = 1U }
  CRC bit width.
- enum crc_result_t {
  kCrcFinalChecksum = 0U,
  kCrcIntermediateChecksum = 1U }
  CRC result type.

Functions

- void CRC_Init (CRC_Type ∗base, const crc_config_t ∗config)
  Enables and configures the CRC peripheral module.
- static void CRC_Deinit (CRC_Type ∗base)
  Disables the CRC peripheral module.
- void CRC_GetDefaultConfig (crc_config_t ∗config)

Loads default values to the CRC protocol configuration structure.
- void CRC_WriteData (CRC_Type *base, const uint8_t *data, size_t dataSize)
  Writes data to the CRC module.
- uint32_t CRC_Get32bitResult (CRC_Type *base)
  Reads the 32-bit checksum from the CRC module.
- uint16_t CRC_Get16bitResult (CRC_Type *base)
  Reads a 16-bit checksum from the CRC module.

Driver version

- #define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  CRC driver version.

6.6 Data Structure Documentation

6.6.1 struct crc_config_t

This structure holds the configuration for the CRC protocol.

Data Fields

- uint32_t polynomial
  CRC Polynomial, MSBit first.
- uint32_t seed
  Starting checksum value.
- bool reflectIn
  Reflect bits on input.
- bool reflectOut
  Reflect bits on output.
- bool complementChecksum
  True if the result shall be complement of the actual checksum.
- crc_bits_t crcBits
  Selects 16- or 32- bit CRC protocol.
- crc_result_t crcResult
  Selects final or intermediate checksum return from CRC_Get16bitResult() or CRC_Get32bitResult()

Field Documentation

(1) uint32_t crc_config_t::polynomial

Example polynomial: 0x1021 = 1_0000_0010_0001 = x^12+x^5+1

(2) bool crc_config_t::reflectIn

(3) bool crc_config_t::reflectOut

(4) bool crc_config_t::complementChecksum

(5) crc_bits_t crc_config_t::crcBits

6.7 Macro Definition Documentation

6.7.1 #define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Version 2.0.3.

Current version: 2.0.3

Change log:

- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues
- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

6.7.2 #define CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT 1

Use CRC16-CCIT-FALSE as defeault.

6.8 Enumeration Type Documentation

6.8.1 enum crc_bits_t

Enumerator

kCrcBits16 Generate 16-bit CRC code.
kCrcBits32 Generate 32-bit CRC code.

6.8.2 enum crc_result_t

Enumerator

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

6.9 Function Documentation

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

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

Parameters

base	CRC peripheral address.
config	CRC module configuration structure.

6.9.2 static void CRC_Deinit (CRC_Type * *base*) [inline], [static]

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

Parameters

base	CRC peripheral address.

6.9.3 void CRC_GetDefaultConfig (crc_config_t * *config*)

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

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

Parameters

config	CRC protocol configuration structure.

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

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

Parameters

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

6.9.5 uint32_t CRC_Get32bitResult (CRC_Type * *base*)

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

Parameters

base	CRC peripheral address.

Returns

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

6.9.6 uint16_t CRC_Get16bitResult (CRC_Type * *base*)

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

Parameters

base	CRC peripheral address.

Returns

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

Chapter 7
EWM: External Watchdog Monitor Driver

7.1 Overview

The MCUXpresso SDK provides a peripheral driver for the External Watchdog (EWM) Driver module of MCUXpresso SDK devices.

7.2 Typical use case

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

Data Structures

- struct ewm_config_t
  Describes EWM clock source. More...

Enumerations

- enum _ewm_interrupt_enable_t { kEWM_InterruptEnable = EWM_CTRL_INTEN_MASK }
  EWM interrupt configuration structure with default settings all disabled.
- enum _ewm_status_flags_t { kEWM_RunningFlag = EWM_CTRL_EWMEN_MASK }
  EWM status flags.

Driver version

- #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  EWM driver version 2.0.3.

EWM initialization and de-initialization

- void EWM_Init (EWM_Type ∗base, const ewm_config_t ∗config)
  Initializes the EWM peripheral.
- void EWM_Deinit (EWM_Type ∗base)
  Deinitializes the EWM peripheral.
- void EWM_GetDefaultConfig (ewm_config_t ∗config)
  Initializes the EWM configuration structure.

EWM functional Operation

- static void EWM_EnableInterrupts (EWM_Type ∗base, uint32_t mask)
  Enables the EWM interrupt.
- static void EWM_DisableInterrupts (EWM_Type ∗base, uint32_t mask)
  Disables the EWM interrupt.
- static uint32_t EWM_GetStatusFlags (EWM_Type ∗base)
  Gets all status flags.

- void EWM_Refresh (EWM_Type ∗base)
  Services the EWM.

7.3 Data Structure Documentation

7.3.1 struct ewm_config_t

Data structure for EWM configuration.

This structure is used to configure the EWM.

Data Fields

- bool enableEwm
  Enable EWM module.
- bool enableEwmInput
  Enable EWM_in input.
- bool setInputAssertLogic
  EWM_in signal assertion state.
- bool enableInterrupt
  Enable EWM interrupt.
- uint8_t prescaler
  Clock prescaler value.
- uint8_t compareLowValue
  Compare low-register value.
- uint8_t compareHighValue
  Compare high-register value.

7.4 Macro Definition Documentation

7.4.1 #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

7.5 Enumeration Type Documentation

7.5.1 enum _ewm_interrupt_enable_t

This structure contains the settings for all of EWM interrupt configurations.

Enumerator

kEWM_InterruptEnable Enable the EWM to generate an interrupt.

7.5.2 enum _ewm_status_flags_t

This structure contains the constants for the EWM status flags for use in the EWM functions.

Enumerator

kEWM_RunningFlag Running flag, set when EWM is enabled.

7.6 Function Documentation

7.6.1 void EWM_Init (EWM_Type ∗ *base*, const ewm_config_t ∗ *config*)

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
*   ewm_config_t config;
*   EWM_GetDefaultConfig(&config);
*   config.compareHighValue = 0xAAU;
*   EWM_Init(ewm_base,&config);
*
```

Parameters

base	EWM peripheral base address
config	The configuration of the EWM

7.6.2 void EWM_Deinit (EWM_Type ∗ *base*)

This function is used to shut down the EWM.

Parameters

base	EWM peripheral base address

7.6.3 void EWM_GetDefaultConfig (ewm_config_t ∗ *config*)

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
*   ewmConfig->enableEwm = true;
*   ewmConfig->enableEwmInput = false;
*   ewmConfig->setInputAssertLogic = false;
*   ewmConfig->enableInterrupt = false;
*   ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
*   ewmConfig->prescaler = 0;
*   ewmConfig->compareLowValue = 0;
*   ewmConfig->compareHighValue = 0xFEU;
*
```

Parameters

config	Pointer to the EWM configuration structure.

See Also

ewm_config_t

7.6.4 static void EWM_EnableInterrupts (EWM_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the EWM interrupt.

Parameters

base	EWM peripheral base address
mask	The interrupts to enable The parameter can be combination of the following source if defined • kEWM_InterruptEnable

7.6.5 static void EWM_DisableInterrupts (EWM_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the EWM interrupt.

Parameters

base	EWM peripheral base address
mask	The interrupts to disable The parameter can be combination of the following source if defined • kEWM_InterruptEnable

7.6.6 static uint32_t EWM_GetStatusFlags (EWM_Type * *base*) [inline], [static]

This function gets all status flags.

This is an example for getting the running flag.

```
*   uint32_t status;
*   status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
*
```

Parameters

base	EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

_ewm_status_flags_t
- True: a related status flag has been set.
- False: a related status flag is not set.

7.6.7 void EWM_Refresh (EWM_Type ∗ *base*)

This function resets the EWM counter to zero.

Parameters

base	EWM peripheral base address

Chapter 8
C90TFS Flash Driver

8.1 Overview

The flash provides the C90TFS Flash driver of Kinetis devices with the C90TFS Flash module inside. The flash driver provides general APIs to handle specific operations on C90TFS/FTFx Flash module. The user can use those APIs directly in the application. In addition, it provides internal functions called by the driver. Although these functions are not meant to be called from the user's application directly, the APIs can still be used.

Modules

- Ftftx CACHE Driver
- Ftftx FLASH Driver
- Ftftx FLEXNVM Driver
- ftfx controller
- ftfx feature

8.2 Ftftx FLASH Driver

8.2.1 Overview

Data Structures

- union pflash_prot_status_t
  PFlash protection status. More...
- struct flash_config_t
  Flash driver state information. More...

Enumerations

- enum flash_prot_state_t {
  kFLASH_ProtectionStateUnprotected,
  kFLASH_ProtectionStateProtected,
  kFLASH_ProtectionStateMixed }
  Enumeration for the three possible flash protection levels.
- enum flash_property_tag_t {
  kFLASH_PropertyPflash0SectorSize = 0x00U,
  kFLASH_PropertyPflash0TotalSize = 0x01U,
  kFLASH_PropertyPflash0BlockSize = 0x02U,
  kFLASH_PropertyPflash0BlockCount = 0x03U,
  kFLASH_PropertyPflash0BlockBaseAddr = 0x04U,
  kFLASH_PropertyPflash0FacSupport = 0x05U,
  kFLASH_PropertyPflash0AccessSegmentSize = 0x06U,
  kFLASH_PropertyPflash0AccessSegmentCount = 0x07U,
  kFLASH_PropertyPflash1SectorSize = 0x10U,
  kFLASH_PropertyPflash1TotalSize = 0x11U,
  kFLASH_PropertyPflash1BlockSize = 0x12U,
  kFLASH_PropertyPflash1BlockCount = 0x13U,
  kFLASH_PropertyPflash1BlockBaseAddr = 0x14U,
  kFLASH_PropertyPflash1FacSupport = 0x15U,
  kFLASH_PropertyPflash1AccessSegmentSize = 0x16U,
  kFLASH_PropertyPflash1AccessSegmentCount = 0x17U,
  kFLASH_PropertyFlexRamBlockBaseAddr = 0x20U,
  kFLASH_PropertyFlexRamTotalSize = 0x21U }
  Enumeration for various flash properties.

Flash version

- #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(3U, 1U, 2U))
  Flash driver version for SDK.
- #define FSL_FLASH_DRIVER_VERSION_ROM (MAKE_VERSION(3U, 0U, 0U))
  Flash driver version for ROM.

Initialization

- status_t FLASH_Init (flash_config_t *config)
  Initializes the global flash properties structure members.

Erasing

- status_t FLASH_Erase (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)
  Erases the Dflash sectors encompassed by parameters passed into function.
- status_t FLASH_EraseSectorNonBlocking (flash_config_t *config, uint32_t start, uint32_t key)
  Erases the Dflash sectors encompassed by parameters passed into function.
- status_t FLASH_EraseAll (flash_config_t *config, uint32_t key)
  Erases entire flexnvm.

Programming

- status_t FLASH_Program (flash_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)
  Programs flash with data at locations passed in through parameters.
- status_t FLASH_ProgramOnce (flash_config_t *config, uint32_t index, uint8_t *src, uint32_t lengthInBytes)
  Program the Program-Once-Field through parameters.

Reading

- status_t FLASH_ReadResource (flash_config_t *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, ftfx_read_resource_opt_t option)
  Reads the resource with data at locations passed in through parameters.
- status_t FLASH_ReadOnce (flash_config_t *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)
  Reads the Program Once Field through parameters.

Verification

- status_t FLASH_VerifyErase (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)
  Verifies an erasure of the desired flash area at a specified margin level.
- status_t FLASH_VerifyEraseAll (flash_config_t *config, ftfx_margin_value_t margin)
  Verifies erasure of the entire flash at a specified margin level.
- status_t FLASH_VerifyProgram (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)
  Verifies programming of the desired flash area at a specified margin level.

Security

- status_t FLASH_GetSecurityState (flash_config_t ∗config, ftfx_security_state_t ∗state)
  Returns the security state via the pointer passed into the function.
- status_t FLASH_SecurityBypass (flash_config_t ∗config, const uint8_t ∗backdoorKey)
  Allows users to bypass security with a backdoor key.

Protection

- status_t FLASH_IsProtected (flash_config_t ∗config, uint32_t start, uint32_t lengthInBytes, flash_-prot_state_t ∗protection_state)
  Returns the protection state of the desired flash area via the pointer passed into the function.
- status_t FLASH_PflashSetProtection (flash_config_t ∗config, pflash_prot_status_t ∗protectStatus)
  Sets the PFlash Protection to the intended protection status.
- status_t FLASH_PflashGetProtection (flash_config_t ∗config, pflash_prot_status_t ∗protectStatus)
  Gets the PFlash protection status.

Properties

- status_t FLASH_GetProperty (flash_config_t ∗config, flash_property_tag_t whichProperty, uint32-_t ∗value)
  Returns the desired flash property.

commantStatus

- status_t FLASH_GetCommandState (void)
  Get previous command status.

8.2.2 Data Structure Documentation

8.2.2.1 union pflash_prot_status_t

Data Fields

- uint32_t protl
  PROT[31:0] .
- uint32_t proth
  PROT[63:32].
- uint8_t protsl
  PROTS[7:0] .
- uint8_t protsh
  PROTS[15:8] .

Field Documentation

(1) uint32_t pflash_prot_status_t::protl

(2) uint32_t pflash_prot_status_t::proth

(3) uint8_t pflash_prot_status_t::protsl

(4) uint8_t pflash_prot_status_t::protsh

8.2.2.2 struct flash_config_t

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

8.2.3 Macro Definition Documentation

8.2.3.1 #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(3U, 1U, 2U))

Version 3.1.2.

8.2.3.2 #define FSL_FLASH_DRIVER_VERSION_ROM (MAKE_VERSION(3U, 0U, 0U))

Version 3.0.0.

8.2.4 Enumeration Type Documentation

8.2.4.1 enum flash_prot_state_t

Enumerator

kFLASH_ProtectionStateUnprotected Flash region is not protected.
kFLASH_ProtectionStateProtected Flash region is protected.
kFLASH_ProtectionStateMixed Flash is mixed with protected and unprotected region.

8.2.4.2 enum flash_property_tag_t

Enumerator

kFLASH_PropertyPflash0SectorSize Pflash sector size property.
kFLASH_PropertyPflash0TotalSize Pflash total size property.
kFLASH_PropertyPflash0BlockSize Pflash block size property.
kFLASH_PropertyPflash0BlockCount Pflash block count property.

kFLASH_PropertyPflash0BlockBaseAddr Pflash block base address property.
kFLASH_PropertyPflash0FacSupport Pflash fac support property.
kFLASH_PropertyPflash0AccessSegmentSize Pflash access segment size property.
kFLASH_PropertyPflash0AccessSegmentCount Pflash access segment count property.
kFLASH_PropertyPflash1SectorSize Pflash sector size property.
kFLASH_PropertyPflash1TotalSize Pflash total size property.
kFLASH_PropertyPflash1BlockSize Pflash block size property.
kFLASH_PropertyPflash1BlockCount Pflash block count property.
kFLASH_PropertyPflash1BlockBaseAddr Pflash block base address property.
kFLASH_PropertyPflash1FacSupport Pflash fac support property.
kFLASH_PropertyPflash1AccessSegmentSize Pflash access segment size property.
kFLASH_PropertyPflash1AccessSegmentCount Pflash access segment count property.
kFLASH_PropertyFlexRamBlockBaseAddr FlexRam block base address property.
kFLASH_PropertyFlexRamTotalSize FlexRam total size property.

8.2.5 Function Documentation

8.2.5.1 status_t FLASH_Init (flash_config_t ∗ *config*)

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

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Partition-StatusUpdateFailure	Failed to update the partition status.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	The parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	The address is out of range.
kStatus_FTFx_EraseKey-Error	The API erase key is invalid.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	The parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	The address is out of range.
kStatus_FTFx_EraseKey-Error	The API erase key is invalid.

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the all pflash and flexnvm were erased successfully, the swap and eeprom have been reset to unconfigured state.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_EraseKey-Error	API erase key is invalid.

kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during command execution.
kStatus_FTFx_Partition-StatusUpdateFailure	Failed to update the partition status.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the desired data were programed successfully into flash based on desired start address and length.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with the specified baseline.

kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; The index indicating the area of program once field was programed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.

kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.

kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

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

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the specified FLASH region has been erased.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; all program flash and flexnvm were in erased state.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the desired data have been successfully programed into specified FLASH region.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the security state of flash was stored to state.

kStatus_FTFx_Invalid-Argument	An invalid argument is provided.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the protection state of specified FLASH region was stored to protection_state.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Address-Error	The address is out of range.

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the specified FLASH region is protected.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-CommandFailure	Run-time error during command execution.

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

Parameters

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

Return values

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the flash property was stored to value.
kStatus_FTFx_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_UnknownProperty	An unknown property tag.

8.2.5.18 status_t FLASH_GetCommandState (void)

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

Return values

kStatus_FTFx_Success	The previous command is executed successfully.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

8.3 Ftftx CACHE Driver

8.3.1 Overview

Data Structures

- struct ftfx_prefetch_speculation_status_t
  FTFx prefetch speculation status. More...
- struct ftfx_cache_config_t
  FTFx cache driver state information. More...

Enumerations

- enum _ftfx_cache_ram_func_constants { kFTFx_CACHE_RamFuncMaxSizeInWords = 16U }
  Constants for execute-in-RAM flash function.

Functions

- status_t FTFx_CACHE_Init (ftfx_cache_config_t ∗config)
  Initializes the global FTFx cache structure members.
- status_t FTFx_CACHE_ClearCachePrefetchSpeculation (ftfx_cache_config_t ∗config, bool isPreProcess)
  Process the cache/prefetch/speculation to the flash.
- status_t FTFx_CACHE_PflashSetPrefetchSpeculation (ftfx_prefetch_speculation_status_t ∗speculationStatus)
  Sets the PFlash prefetch speculation to the intended speculation status.
- status_t FTFx_CACHE_PflashGetPrefetchSpeculation (ftfx_prefetch_speculation_status_t ∗speculationStatus)
  Gets the PFlash prefetch speculation status.

8.3.2 Data Structure Documentation

8.3.2.1 struct ftfx_prefetch_speculation_status_t

Data Fields

- bool instructionOff
  Instruction speculation.
- bool dataOff
  Data speculation.

Field Documentation

(1) bool ftfx_prefetch_speculation_status_t::instructionOff

(2) bool ftfx_prefetch_speculation_status_t::dataOff

8.3.2.2 struct ftfx_cache_config_t

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

Data Fields

- uint8_t flashMemoryIndex
  0 - primary flash; 1 - secondary flash
- function_bit_operation_ptr_t bitOperFuncAddr
  An buffer point to the flash execute-in-RAM function.

Field Documentation

(1) function_bit_operation_ptr_t ftfx_cache_config_t::bitOperFuncAddr

8.3.3 Enumeration Type Documentation

8.3.3.1 enum _ftfx_cache_ram_func_constants

Enumerator

kFTFx_CACHE_RamFuncMaxSizeInWords The maximum size of execute-in-RAM function.

8.3.4 Function Documentation

8.3.4.1 status_t FTFx_CACHE_Init (ftfx_cache_config_t ∗ *config*)

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

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.

kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	Invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.

8.3.4.3 status_t FTFx_CACHE_PflashSetPrefetchSpeculation (ftfx_prefetch_speculation-_status_t * *speculationStatus*)

Parameters

speculation-Status	The expected protect status to set to the PFlash protection register. Each bit is

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-SpeculationOption	An invalid speculation option argument is provided.

8.3.4.4 status_t FTFx_CACHE_PflashGetPrefetchSpeculation (ftfx_prefetch_speculation-_status_t * *speculationStatus*)

Parameters

speculation-Status	Speculation status returned by the PFlash IP.

Return values

kStatus_FTFx_Success	API was executed successfully.

8.4 Ftftx FLEXNVM Driver

8.4.1 Overview

Data Structures

- struct flexnvm_config_t
  Flexnvm driver state information. More...

Enumerations

- enum flexnvm_property_tag_t {
  kFLEXNVM_PropertyDflashSectorSize = 0x00U,
  kFLEXNVM_PropertyDflashTotalSize = 0x01U,
  kFLEXNVM_PropertyDflashBlockSize = 0x02U,
  kFLEXNVM_PropertyDflashBlockCount = 0x03U,
  kFLEXNVM_PropertyDflashBlockBaseAddr = 0x04U,
  kFLEXNVM_PropertyAliasDflashBlockBaseAddr = 0x05U,
  kFLEXNVM_PropertyFlexRamBlockBaseAddr = 0x06U,
  kFLEXNVM_PropertyFlexRamTotalSize = 0x07U,
  kFLEXNVM_PropertyEepromTotalSize = 0x08U }
  Enumeration for various flexnvm properties.

Functions

- status_t FLEXNVM_EepromWrite (flexnvm_config_t ∗config, uint32_t start, uint8_t ∗src, uint32_t lengthInBytes)
  Programs the EEPROM with data at locations passed in through parameters.

Initialization

- status_t FLEXNVM_Init (flexnvm_config_t ∗config)
  Initializes the global flash properties structure members.

Erasing

- status_t FLEXNVM_DflashErase (flexnvm_config_t ∗config, uint32_t start, uint32_t lengthInBytes, uint32_t key)
  Erases the Dflash sectors encompassed by parameters passed into function.
- status_t FLEXNVM_EraseAll (flexnvm_config_t ∗config, uint32_t key)
  Erases entire flexnvm.

Programming

- status_t FLEXNVM_DflashProgram (flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32-_t lengthInBytes)
  Programs flash with data at locations passed in through parameters.
- status_t FLEXNVM_ProgramPartition (flexnvm_config_t *config, ftfx_partition_flexram_load_-opt_t option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode)
  Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Reading

- status_t FLEXNVM_ReadResource (flexnvm_config_t *config, uint32_t start, uint8_t *dst, uint32-_t lengthInBytes, ftfx_read_resource_opt_t option)
  Reads the resource with data at locations passed in through parameters.

Verification

- status_t FLEXNVM_DflashVerifyErase (flexnvm_config_t *config, uint32_t start, uint32_t length-InBytes, ftfx_margin_value_t margin)
  Verifies an erasure of the desired flash area at a specified margin level.
- status_t FLEXNVM_VerifyEraseAll (flexnvm_config_t *config, ftfx_margin_value_t margin)
  Verifies erasure of the entire flash at a specified margin level.
- status_t FLEXNVM_DflashVerifyProgram (flexnvm_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)
  Verifies programming of the desired flash area at a specified margin level.

Security

- status_t FLEXNVM_GetSecurityState (flexnvm_config_t *config, ftfx_security_state_t *state)
  Returns the security state via the pointer passed into the function.
- status_t FLEXNVM_SecurityBypass (flexnvm_config_t *config, const uint8_t *backdoorKey)
  Allows users to bypass security with a backdoor key.

Flash Protection Utilities

- status_t FLEXNVM_DflashSetProtection (flexnvm_config_t *config, uint8_t protectStatus)
  Sets the DFlash protection to the intended protection status.
- status_t FLEXNVM_DflashGetProtection (flexnvm_config_t *config, uint8_t *protectStatus)
  Gets the DFlash protection status.
- status_t FLEXNVM_EepromSetProtection (flexnvm_config_t *config, uint8_t protectStatus)
  Sets the EEPROM protection to the intended protection status.
- status_t FLEXNVM_EepromGetProtection (flexnvm_config_t *config, uint8_t *protectStatus)

Gets the EEPROM protection status.

Properties

- status_t FLEXNVM_GetProperty (flexnvm_config_t *config, flexnvm_property_tag_t which-Property, uint32_t *value)
  Returns the desired flexnvm property.

8.4.2 Data Structure Documentation

8.4.2.1 struct flexnvm_config_t

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

8.4.3 Enumeration Type Documentation

8.4.3.1 enum flexnvm_property_tag_t

Enumerator

kFLEXNVM_PropertyDflashSectorSize Dflash sector size property.
kFLEXNVM_PropertyDflashTotalSize Dflash total size property.
kFLEXNVM_PropertyDflashBlockSize Dflash block size property.
kFLEXNVM_PropertyDflashBlockCount Dflash block count property.
kFLEXNVM_PropertyDflashBlockBaseAddr Dflash block base address property.
kFLEXNVM_PropertyAliasDflashBlockBaseAddr Dflash block base address Alias property.
kFLEXNVM_PropertyFlexRamBlockBaseAddr FlexRam block base address property.
kFLEXNVM_PropertyFlexRamTotalSize FlexRam total size property.
kFLEXNVM_PropertyEepromTotalSize EEPROM total size property.

8.4.4 Function Documentation

8.4.4.1 status_t FLEXNVM_Init (flexnvm_config_t * *config*)

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

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Partition-StatusUpdateFailure	Failed to update the partition status.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the appropriate number of date flash sectors based on the desired start address and length were erased successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.

kStatus_FTFx_-AlignmentError	The parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	The address is out of range.
kStatus_FTFx_EraseKey-Error	The API erase key is invalid.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the entire flexnvm has been erased successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_EraseKey-Error	API erase key is invalid.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.

kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during command execution.
kStatus_FTFx_Partition-StatusUpdateFailure	Failed to update the partition status.

8.4.4.4 status_t FLEXNVM_DflashProgram (flexnvm_config_t * *config*, uint32_t *start*, uint8_t * *src*, uint32_t *lengthInBytes*)

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the desired date have been successfully programed into specified date flash region.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.

kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

8.4.4.5 status_t FLEXNVM_ProgramPartition (flexnvm_config_t * *config*, ftfx_partition_flexram_load_opt_t *option*, uint32_t *eepromDataSizeCode*, uint32_t *flexnvmPartitionCode*)

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
kStatus_FTFx_Invalid-Argument	Invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.

kStatus_FTFx_-CommandFailure	Run-time error during command execution.

8.4.4.6 status_t FLEXNVM_ReadResource (flexnvm_config_t * *config*, uint32_t *start*, uint8_t * *dst*, uint32_t *lengthInBytes*, ftfx_read_resource_opt_t *option*)

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.

kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

8.4.4.7 status_t FLEXNVM_DflashVerifyErase (flexnvm_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, ftfx_margin_value_t *margin*)

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the specified data flash region is in erased state.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.

kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

8.4.4.8 status_t FLEXNVM_VerifyEraseAll (flexnvm_config_t * *config*, ftfx_margin_value_t *margin*)

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the entire flexnvm region is in erased state.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

8.4.4.9 status_t FLEXNVM_DflashVerifyProgram (flexnvm_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, const uint8_t * *expectedData*, ftfx_margin_value_t *margin*, uint32_t * *failedAddress*, uint32_t * *failedData*)

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

Parameters

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

Return values

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

8.4.4.10 status_t FLEXNVM_GetSecurityState (flexnvm_config_t * *config*, ftfx_security_state_t * *state*)

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the security state of flexnvm was stored to state.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.

8.4.4.11 status_t FLEXNVM_SecurityBypass (flexnvm_config_t * *config*, const uint8_t * *backdoorKey*)

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.

kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

8.4.4.12 status_t FLEXNVM_EepromWrite (flexnvm_config_t * *config*, uint32_t *start*, uint8_t * *src*, uint32_t *lengthInBytes*)

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the desires data have been successfully programed into specified eeprom region.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_Set-FlexramAsEepromError	Failed to set flexram as eeprom.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_Recover-FlexramAsRamError	Failed to recover the FlexRAM as RAM.

8.4.4.13 status_t FLEXNVM_DflashSetProtection (flexnvm_config_t * *config*, uint8_t *protectStatus*)

Parameters

config	A pointer to the storage for the driver runtime state.
protectStatus	The expected protect status to set to the DFlash protection register. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

kStatus_FTFx_Success	API was executed successfully; the specified DFlash region is protected.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-CommandNotSupported	Flash API is not supported.
kStatus_FTFx_-CommandFailure	Run-time error during command execution.

8.4.4.14 status_t FLEXNVM_DflashGetProtection (flexnvm_config_t * *config*, uint8_t * *protectStatus*)

Parameters

config	A pointer to the storage for the driver runtime state.
protectStatus	DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash, and so on. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.

kStatus_FTFx_-CommandNotSupported	Flash API is not supported.

8.4.4.15 status_t FLEXNVM_EepromSetProtection (flexnvm_config_t ∗ *config*, uint8_t *protectStatus*)

Parameters

config	A pointer to the storage for the driver runtime state.
protectStatus	The expected protect status to set to the EEPROM protection register. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of EEPROM, and so on. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-CommandNotSupported	Flash API is not supported.
kStatus_FTFx_-CommandFailure	Run-time error during command execution.

8.4.4.16 status_t FLEXNVM_EepromGetProtection (flexnvm_config_t ∗ *config*, uint8_t ∗ *protectStatus*)

Parameters

config	A pointer to the storage for the driver runtime state.
protectStatus	DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of the EEPROM. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-CommandNotSupported	Flash API is not supported.

8.4.4.17 status_t FLEXNVM_GetProperty (flexnvm_config_t * *config,* flexnvm_property_tag_t *whichProperty,* uint32_t * *value*)

Parameters

config	A pointer to the storage for the driver runtime state.
whichProperty	The desired property from the list of properties in enum flexnvm_property_tag_t
value	A pointer to the value returned for the desired flexnvm property.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_Unknown-Property	An unknown property tag.

8.5 ftfx feature

8.5.1 Overview

Modules

- ftfx adapter

Macros

- #define FTFx_DRIVER_HAS_FLASH1_SUPPORT (0U)
  Indicates whether the secondary flash is supported in the Flash driver.

FTFx configuration

- #define FTFx_DRIVER_IS_FLASH_RESIDENT 1U
  Flash driver location.
- #define FTFx_DRIVER_IS_EXPORTED 0U
  Flash Driver Export option.

Secondary flash configuration

- #define FTFx_FLASH1_HAS_PROT_CONTROL (0U)
  Indicates whether the secondary flash has its own protection register in flash module.
- #define FTFx_FLASH1_HAS_XACC_CONTROL (0U)
  Indicates whether the secondary flash has its own Execute-Only access register in flash module.

8.5.2 Macro Definition Documentation

8.5.2.1 #define FTFx_DRIVER_IS_FLASH_RESIDENT 1U

Used for the flash resident application.

8.5.2.2 #define FTFx_DRIVER_IS_EXPORTED 0U

Used for the MCUXpresso SDK application.

8.5.2.3 #define FTFx_FLASH1_HAS_PROT_CONTROL (0U)

8.5.2.4 #define FTFx_FLASH1_HAS_XACC_CONTROL (0U)

8.5.3 ftfx adapter

8.6 ftfx controller

8.6.1 Overview

Modules

- ftfx utilities

Data Structures

- struct ftfx_spec_mem_t
  ftfx special memory access information. More...
- struct ftfx_mem_desc_t
  Flash memory descriptor. More...
- struct ftfx_ops_config_t
  Active FTFx information for the current operation. More...
- struct ftfx_ifr_desc_t
  Flash IFR memory descriptor. More...
- struct ftfx_config_t
  Flash driver state information. More...

Enumerations

- enum ftfx_partition_flexram_load_opt_t {
  kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData,
  kFTFx_PartitionFlexramLoadOptNotLoaded = 0x01U }
  Enumeration for the FlexRAM load during reset option.
- enum ftfx_read_resource_opt_t {
  kFTFx_ResourceOptionFlashIfr,
  kFTFx_ResourceOptionVersionId = 0x01U }
  Enumeration for the two possible options of flash read resource command.
- enum ftfx_margin_value_t {
  kFTFx_MarginValueNormal,
  kFTFx_MarginValueUser,
  kFTFx_MarginValueFactory,
  kFTFx_MarginValueInvalid }
  Enumeration for supported FTFx margin levels.
- enum ftfx_security_state_t {
  kFTFx_SecurityStateNotSecure = (int)0xc33cc33cu,
  kFTFx_SecurityStateBackdoorEnabled = (int)0x5aa55aa5u,
  kFTFx_SecurityStateBackdoorDisabled = (int)0x5ac33ca5u }
  Enumeration for the three possible FTFx security states.
- enum ftfx_flexram_func_opt_t {
  kFTFx_FlexramFuncOptAvailableAsRam = 0xFFU,
  kFTFx_FlexramFuncOptAvailableForEeprom = 0x00U }
  Enumeration for the two possilbe options of set FlexRAM function command.

- enum _flash_acceleration_ram_property
  Enumeration for acceleration ram property.
- enum ftfx_swap_state_t {
  kFTFx_SwapStateUninitialized = 0x00U,
  kFTFx_SwapStateReady = 0x01U,
  kFTFx_SwapStateUpdate = 0x02U,
  kFTFx_SwapStateUpdateErased = 0x03U,
  kFTFx_SwapStateComplete = 0x04U,
  kFTFx_SwapStateDisabled = 0x05U }
  Enumeration for the possible flash Swap status.
- enum _ftfx_memory_type
  Enumeration for FTFx memory type.

FTFx status

- enum {
  kStatus_FTFx_Success = MAKE_STATUS(kStatusGroupGeneric, 0),
  kStatus_FTFx_InvalidArgument = MAKE_STATUS(kStatusGroupGeneric, 4),
  kStatus_FTFx_SizeError = MAKE_STATUS(kStatusGroupFtfxDriver, 0),
  kStatus_FTFx_AlignmentError,
  kStatus_FTFx_AddressError = MAKE_STATUS(kStatusGroupFtfxDriver, 2),
  kStatus_FTFx_AccessError,
  kStatus_FTFx_ProtectionViolation,
  kStatus_FTFx_CommandFailure,
  kStatus_FTFx_UnknownProperty = MAKE_STATUS(kStatusGroupFtfxDriver, 6),
  kStatus_FTFx_EraseKeyError = MAKE_STATUS(kStatusGroupFtfxDriver, 7),
  kStatus_FTFx_RegionExecuteOnly = MAKE_STATUS(kStatusGroupFtfxDriver, 8),
  kStatus_FTFx_ExecuteInRamFunctionNotReady,
  kStatus_FTFx_PartitionStatusUpdateFailure,
  kStatus_FTFx_SetFlexramAsEepromError,
  kStatus_FTFx_RecoverFlexramAsRamError,
  kStatus_FTFx_SetFlexramAsRamError = MAKE_STATUS(kStatusGroupFtfxDriver, 13),
  kStatus_FTFx_RecoverFlexramAsEepromError,
  kStatus_FTFx_CommandNotSupported = MAKE_STATUS(kStatusGroupFtfxDriver, 15),
  kStatus_FTFx_SwapSystemNotInUninitialized,
  kStatus_FTFx_SwapIndicatorAddressError,
  kStatus_FTFx_ReadOnlyProperty = MAKE_STATUS(kStatusGroupFtfxDriver, 18),
  kStatus_FTFx_InvalidPropertyValue,
  kStatus_FTFx_InvalidSpeculationOption,
  kStatus_FTFx_CommandOperationInProgress }
  FTFx driver status codes.
- #define kStatusGroupGeneric 0
  FTFx driver status group.
- #define **kStatusGroupFtfxDriver** 1

FTFx API key

- enum _ftfx_driver_api_keys { kFTFx_ApiEraseKey = FOUR_CHAR_CODE('k', 'f', 'e', 'k') }
  Enumeration for FTFx driver API keys.

Initialization

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

Erasing

- status_t FTFx_CMD_Erase (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)
  Erases the flash sectors encompassed by parameters passed into function.
- status_t FTFx_CMD_EraseSectorNonBlocking (ftfx_config_t *config, uint32_t start, uint32_t key)
  Erases the flash sectors encompassed by parameters passed into function.
- status_t FTFx_CMD_EraseAll (ftfx_config_t *config, uint32_t key)
  Erases entire flash.
- status_t FTFx_CMD_EraseAllExecuteOnlySegments (ftfx_config_t *config, uint32_t key)
  Erases all program flash execute-only segments defined by the FXACC registers.

Programming

- status_t FTFx_CMD_Program (ftfx_config_t *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)
  Programs flash with data at locations passed in through parameters.
- status_t FTFx_CMD_ProgramOnce (ftfx_config_t *config, uint32_t index, const uint8_t *src, uint32_t lengthInBytes)
  Programs Program Once Field through parameters.

Reading

- status_t FTFx_CMD_ReadOnce (ftfx_config_t *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)
  Reads the Program Once Field through parameters.
- status_t FTFx_CMD_ReadResource (ftfx_config_t *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, ftfx_read_resource_opt_t option)
  Reads the resource with data at locations passed in through parameters.

Verification

- status_t FTFx_CMD_VerifyErase (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)
  Verifies an erasure of the desired flash area at a specified margin level.
- status_t FTFx_CMD_VerifyEraseAll (ftfx_config_t *config, ftfx_margin_value_t margin)
  Verifies erasure of the entire flash at a specified margin level.
- status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments (ftfx_config_t *config, ftfx_margin_value_t margin)
  Verifies whether the program flash execute-only segments have been erased to the specified read margin level.
- status_t FTFx_CMD_VerifyProgram (ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)
  Verifies programming of the desired flash area at a specified margin level.

Security

- status_t FTFx_REG_GetSecurityState (ftfx_config_t *config, ftfx_security_state_t *state)
  Returns the security state via the pointer passed into the function.
- status_t FTFx_CMD_SecurityBypass (ftfx_config_t *config, const uint8_t *backdoorKey)
  Allows users to bypass security with a backdoor key.

8.6.2 Data Structure Documentation

8.6.2.1 struct ftfx_spec_mem_t

Data Fields

- uint32_t base
  Base address of flash special memory.
- uint32_t size
  size of flash special memory.
- uint32_t count
  flash special memory count.

Field Documentation

(1) uint32_t ftfx_spec_mem_t::base

(2) uint32_t ftfx_spec_mem_t::size

(3) uint32_t ftfx_spec_mem_t::count

8.6.2.2 struct ftfx_mem_desc_t

Data Fields

- uint32_t blockBase
  A base address of the flash block.
- uint32_t totalSize
  The size of the flash block.
- uint32_t sectorSize
  The size in bytes of a sector of flash.
- uint32_t blockCount
  A number of flash blocks.
- uint8_t type
  Type of flash block.
- uint8_t index
  Index of flash block.

Field Documentation

(1) uint8_t ftfx_mem_desc_t::type

(2) uint8_t ftfx_mem_desc_t::index

(3) uint32_t ftfx_mem_desc_t::totalSize

(4) uint32_t ftfx_mem_desc_t::sectorSize

(5) uint32_t ftfx_mem_desc_t::blockCount

8.6.2.3 struct ftfx_ops_config_t

Data Fields

- uint32_t convertedAddress
  A converted address for the current flash type.

Field Documentation

(1) uint32_t ftfx_ops_config_t::convertedAddress

8.6.2.4 struct ftfx_ifr_desc_t

8.6.2.5 struct ftfx_config_t

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

Data Fields

- uint32_t flexramBlockBase
  The base address of the FlexRAM/acceleration RAM.
- uint32_t flexramTotalSize
  The size of the FlexRAM/acceleration RAM.
- uint16_t eepromTotalSize
  The size of EEPROM area which was partitioned from FlexRAM.
- function_ptr_t runCmdFuncAddr
  An buffer point to the flash execute-in-RAM function.

Field Documentation

(1) function_ptr_t ftfx_config_t::runCmdFuncAddr

8.6.3 Macro Definition Documentation

8.6.3.1 #define kStatusGroupGeneric 0

8.6.4 Enumeration Type Documentation

8.6.4.1 anonymous enum

Enumerator

kStatus_FTFx_Success API is executed successfully.
kStatus_FTFx_InvalidArgument Invalid argument.
kStatus_FTFx_SizeError Error size.
kStatus_FTFx_AlignmentError Parameter is not aligned with the specified baseline.
kStatus_FTFx_AddressError Address is out of range.
kStatus_FTFx_AccessError Invalid instruction codes and out-of bound addresses.
kStatus_FTFx_ProtectionViolation The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_CommandFailure Run-time error during command execution.
kStatus_FTFx_UnknownProperty Unknown property.
kStatus_FTFx_EraseKeyError API erase key is invalid.
kStatus_FTFx_RegionExecuteOnly The current region is execute-only.
kStatus_FTFx_ExecuteInRamFunctionNotReady Execute-in-RAM function is not available.
kStatus_FTFx_PartitionStatusUpdateFailure Failed to update partition status.
kStatus_FTFx_SetFlexramAsEepromError Failed to set FlexRAM as EEPROM.
kStatus_FTFx_RecoverFlexramAsRamError Failed to recover FlexRAM as RAM.
kStatus_FTFx_SetFlexramAsRamError Failed to set FlexRAM as RAM.
kStatus_FTFx_RecoverFlexramAsEepromError Failed to recover FlexRAM as EEPROM.
kStatus_FTFx_CommandNotSupported Flash API is not supported.
kStatus_FTFx_SwapSystemNotInUninitialized Swap system is not in an uninitialzed state.
kStatus_FTFx_SwapIndicatorAddressError The swap indicator address is invalid.
kStatus_FTFx_ReadOnlyProperty The flash property is read-only.

kStatus_FTFx_InvalidPropertyValue The flash property value is out of range.
kStatus_FTFx_InvalidSpeculationOption The option of flash prefetch speculation is invalid.
kStatus_FTFx_CommandOperationInProgress The option of flash command is processing.

8.6.4.2 enum _ftfx_driver_api_keys

Note

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

Enumerator

kFTFx_ApiEraseKey Key value used to validate all FTFx erase APIs.

8.6.4.3 enum ftfx_partition_flexram_load_opt_t

Enumerator

kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData FlexRAM is loaded with valid EEPROM data during reset sequence.
kFTFx_PartitionFlexramLoadOptNotLoaded FlexRAM is not loaded during reset sequence.

8.6.4.4 enum ftfx_read_resource_opt_t

Enumerator

kFTFx_ResourceOptionFlashIfr Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR.
kFTFx_ResourceOptionVersionId Select code for the version ID.

8.6.4.5 enum ftfx_margin_value_t

Enumerator

kFTFx_MarginValueNormal Use the ’normal’ read level for 1s.
kFTFx_MarginValueUser Apply the ’User’ margin to the normal read-1 level.
kFTFx_MarginValueFactory Apply the ’Factory’ margin to the normal read-1 level.
kFTFx_MarginValueInvalid Not real margin level, Used to determine the range of valid margin level.

8.6.4.6 enum ftfx_security_state_t

Enumerator

kFTFx_SecurityStateNotSecure Flash is not secure.
kFTFx_SecurityStateBackdoorEnabled Flash backdoor is enabled.
kFTFx_SecurityStateBackdoorDisabled Flash backdoor is disabled.

8.6.4.7 enum ftfx_flexram_func_opt_t

Enumerator

kFTFx_FlexramFuncOptAvailableAsRam An option used to make FlexRAM available as RAM.
kFTFx_FlexramFuncOptAvailableForEeprom An option used to make FlexRAM available for E-EPROM.

8.6.4.8 enum ftfx_swap_state_t

Enumerator

kFTFx_SwapStateUninitialized Flash Swap system is in an uninitialized state.
kFTFx_SwapStateReady Flash Swap system is in a ready state.
kFTFx_SwapStateUpdate Flash Swap system is in an update state.
kFTFx_SwapStateUpdateErased Flash Swap system is in an updateErased state.
kFTFx_SwapStateComplete Flash Swap system is in a complete state.
kFTFx_SwapStateDisabled Flash Swap system is in a disabled state.

8.6.5 Function Documentation

8.6.5.1 void FTFx_API_Init (ftfx_config_t * *config*)

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

Parameters

config	Pointer to the storage for the driver runtime state.

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

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

Parameters

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

Return values

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

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

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

Parameters

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

Return values

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_EraseKeyError	API erase key is invalid.

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

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

Parameters

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

Return values

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

kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.

kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

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

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

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

Parameters

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

option	The resource option which indicates which area should be read back.

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FTFx_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_-CommandFailure	Run-time error during the command execution.

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.

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

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

Parameters

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

Return values

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

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

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

Parameters

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

Return values

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

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

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

Parameters

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

Return values

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

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

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

Parameters

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

Return values

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

8.6.6 ftfx utilities

8.6.6.1 Overview

Macros

- #define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
  Constructs the version number for drivers.
- #define MAKE_STATUS(group, code) ((((group)*100) + (code)))
  Constructs a status code value from a group and a code number.
- #define FOUR_CHAR_CODE(a, b, c, d) (((uint32_t)(d) << 24u) | ((uint32_t)(c) << 16u) | ((uint32_t)(b) << 8u) | ((uint32_t)(a)))
  Constructs the four character code for the Flash driver API key.
- #define B1P4(b) (((uint32_t)(b)&0xFFU) << 24U)
  bytes2word utility.

Alignment macros

- #define ALIGN_DOWN(x, a) (((uint32_t)(x)) & ~((uint32_t)(a)-1u))
  Alignment(down) utility.
- #define ALIGN_UP(x, a) ALIGN_DOWN((uint32_t)(x) + (uint32_t)(a)-1u, a)
  Alignment(up) utility.

8.6.6.2 Macro Definition Documentation

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

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

8.6.6.2.3 #define FOUR_CHAR_CODE(*a*, *b*, *c*, *d*) (((uint32_t)(d) << 24u) | ((uint32_t)(c) << 16u) | ((uint32_t)(b) << 8u) | ((uint32_t)(a)))

8.6.6.2.4 #define ALIGN_DOWN(*x*, *a*) (((uint32_t)(x)) & ~((uint32_t)(a)-1u))

8.6.6.2.5 #define ALIGN_UP(*x*, *a*) ALIGN_DOWN((uint32_t)(x) + (uint32_t)(a)-1u, a)

8.6.6.2.6 #define B1P4(*b*) (((uint32_t)(b)&0xFFU) << 24U)

Chapter 9
FTM: FlexTimer Driver

9.1 Overview

The MCUXpresso SDK provides a driver for the FlexTimer Module (FTM) of MCUXpresso SDK devices.

9.2 Function groups

The FTM driver supports the generation of PWM signals, input capture, dual edge capture, output compare, and quadrature decoder modes. The driver also supports configuring each of the FTM fault inputs.

9.2.1 Initialization and deinitialization

The function FTM_Init() initializes the FTM with specified configurations. The function FTM_GetDefaultConfig() gets the default configurations. The initialization function configures the FTM for the requested register update mode for registers with buffers. It also sets up the FTM's fault operation mode and FTM behavior in the BDM mode.

The function FTM_Deinit() disables the FTM counter and turns off the module clock.

9.2.2 PWM Operations

The function FTM_SetupPwm() sets up FTM channels for the PWM output. The function sets up the PWM signal properties for multiple channels. Each channel has its own duty cycle and level-mode specified. However, the same PWM period and PWM mode is applied to all channels requesting 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 FTM_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular FTM channel.

The function FTM_UpdateChnlEdgeLevelSelect() updates the level select bits of a particular FTM channel. This can be used to disable the PWM output when making changes to the PWM signal.

9.2.3 Input capture operations

The function FTM_SetupInputCapture() sets up an FTM channel for the input capture. The user can specify the capture edge and a filter value to be used when processing the input signal.

The function FTM_SetupDualEdgeCapture() can be used to measure the pulse width of a signal. A channel pair is used during capture with the input signal coming through a channel n. The user can specify whether to use one-shot or continuous capture, the capture edge for each channel, and any filter value to be used when processing the input signal.

9.2.4 Output compare operations

The function FTM_SetupOutputCompare() sets up an FTM channel for the output comparison. The user can specify the channel output on a successful comparison and a comparison value.

9.2.5 Quad decode

The function FTM_SetupQuadDecode() sets up FTM channels 0 and 1 for quad decoding. The user can specify the quad decoding mode, polarity, and filter properties for each input signal.

9.2.6 Fault operation

The function FTM_SetupFault() sets up the properties for each fault. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

9.3 Register Update

Some of the FTM registers have buffers. The driver supports various methods to update these registers with the content of the register buffer. The registers can be updated using the PWM synchronized loading or an intermediate point loading. The update mechanism for register with buffers can be specified through the following fields available in the configuration structure. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ftmMultiple PWM synchronization update modes can be used by providing an OR'ed list of options available in the enumeration ftm_pwm_sync_method_t to the pwmSyncMode field.

When using an intermediate reload points, the PWM synchnronization is not required. Multiple reload points can be used by providing an OR'ed list of options available in the enumeration ftm_reload_point_t to the reloadPoints field.

The driver initialization function sets up the appropriate bits in the FTM module based on the register update options selected.

If software PWM synchronization is used, the below function can be used to initiate a software trigger. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ftm

9.4 Typical use case

9.4.1 PWM output

Output a PWM signal on two FTM channels with different duty cycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ftm

Data Structures

- struct ftm_chnl_pwm_signal_param_t
  Options to configure a FTM channel's PWM signal. More...
- struct ftm_chnl_pwm_config_param_t
  Options to configure a FTM channel using precise setting. More...
- struct ftm_dual_edge_capture_param_t
  FlexTimer dual edge capture parameters. More...
- struct ftm_phase_params_t
  FlexTimer quadrature decode phase parameters. More...
- struct ftm_fault_param_t
  Structure is used to hold the parameters to configure a FTM fault. More...
- struct ftm_config_t
  FTM configuration structure. More...

Enumerations

- enum ftm_chnl_t {
  kFTM_Chnl_0 = 0U,
  kFTM_Chnl_1,
  kFTM_Chnl_2,
  kFTM_Chnl_3,
  kFTM_Chnl_4,
  kFTM_Chnl_5,
  kFTM_Chnl_6,
  kFTM_Chnl_7 }
  List of FTM channels.
- enum ftm_fault_input_t {
  kFTM_Fault_0 = 0U,
  kFTM_Fault_1,
  kFTM_Fault_2,
  kFTM_Fault_3 }
  List of FTM faults.
- enum ftm_pwm_mode_t {
  kFTM_EdgeAlignedPwm = 0U,
  kFTM_CenterAlignedPwm,
  kFTM_EdgeAlignedCombinedPwm,
  kFTM_CenterAlignedCombinedPwm,
  kFTM_AsymmetricalCombinedPwm }
  FTM PWM operation modes.
- enum ftm_pwm_level_select_t {

kFTM_NoPwmSignal = 0U,
kFTM_LowTrue,
kFTM_HighTrue }
FTM PWM output pulse mode: high-true, low-true or no output.
- enum ftm_output_compare_mode_t {
kFTM_NoOutputSignal = (1U << FTM_CnSC_MSA_SHIFT),
kFTM_ToggleOnMatch = ((1U << FTM_CnSC_MSA_SHIFT) | (1U << FTM_CnSC_ELSA_S-HIFT)),
kFTM_ClearOnMatch = ((1U << FTM_CnSC_MSA_SHIFT) | (2U << FTM_CnSC_ELSA_SH-IFT)),
kFTM_SetOnMatch = ((1U << FTM_CnSC_MSA_SHIFT) | (3U << FTM_CnSC_ELSA_SHIF-T)) }
FlexTimer output compare mode.
- enum ftm_input_capture_edge_t {
kFTM_RisingEdge = (1U << FTM_CnSC_ELSA_SHIFT),
kFTM_FallingEdge = (2U << FTM_CnSC_ELSA_SHIFT),
kFTM_RiseAndFallEdge = (3U << FTM_CnSC_ELSA_SHIFT) }
FlexTimer input capture edge.
- enum ftm_dual_edge_capture_mode_t {
kFTM_OneShot = 0U,
kFTM_Continuous = (1U << FTM_CnSC_MSA_SHIFT) }
FlexTimer dual edge capture modes.
- enum ftm_quad_decode_mode_t {
kFTM_QuadPhaseEncode = 0U,
kFTM_QuadCountAndDir }
FlexTimer quadrature decode modes.
- enum ftm_phase_polarity_t {
kFTM_QuadPhaseNormal = 0U,
kFTM_QuadPhaseInvert }
FlexTimer quadrature phase polarities.
- enum ftm_deadtime_prescale_t {
kFTM_Deadtime_Prescale_1 = 1U,
kFTM_Deadtime_Prescale_4,
kFTM_Deadtime_Prescale_16 }
FlexTimer pre-scaler factor for the dead time insertion.
- enum ftm_clock_source_t {
kFTM_SystemClock = 1U,
kFTM_FixedClock,
kFTM_ExternalClock }
FlexTimer clock source selection.
- enum ftm_clock_prescale_t {

kFTM_Prescale_Divide_1 = 0U,
kFTM_Prescale_Divide_2,
kFTM_Prescale_Divide_4,
kFTM_Prescale_Divide_8,
kFTM_Prescale_Divide_16,
kFTM_Prescale_Divide_32,
kFTM_Prescale_Divide_64,
kFTM_Prescale_Divide_128 }
FlexTimer pre-scaler factor selection for the clock source.
- enum ftm_bdm_mode_t {
kFTM_BdmMode_0 = 0U,
kFTM_BdmMode_1,
kFTM_BdmMode_2,
kFTM_BdmMode_3 }
Options for the FlexTimer behaviour in BDM Mode.
- enum ftm_fault_mode_t {
kFTM_Fault_Disable = 0U,
kFTM_Fault_EvenChnls,
kFTM_Fault_AllChnlsMan,
kFTM_Fault_AllChnlsAuto }
Options for the FTM fault control mode.
- enum ftm_external_trigger_t {
kFTM_Chnl0Trigger = (1U << 4),
kFTM_Chnl1Trigger = (1U << 5),
kFTM_Chnl2Trigger = (1U << 0),
kFTM_Chnl3Trigger = (1U << 1),
kFTM_Chnl4Trigger = (1U << 2),
kFTM_Chnl5Trigger = (1U << 3),
kFTM_InitTrigger = (1U << 6),
kFTM_ReloadInitTrigger = (1U << 7) }
FTM external trigger options.
- enum ftm_pwm_sync_method_t {
kFTM_SoftwareTrigger = FTM_SYNC_SWSYNC_MASK,
kFTM_HardwareTrigger_0 = FTM_SYNC_TRIG0_MASK,
kFTM_HardwareTrigger_1 = FTM_SYNC_TRIG1_MASK,
kFTM_HardwareTrigger_2 = FTM_SYNC_TRIG2_MASK }
FlexTimer PWM sync options to update registers with buffer.
- enum ftm_reload_point_t {

kFTM_Chnl0Match = (1U << 0),
kFTM_Chnl1Match = (1U << 1),
kFTM_Chnl2Match = (1U << 2),
kFTM_Chnl3Match = (1U << 3),
kFTM_Chnl4Match = (1U << 4),
kFTM_Chnl5Match = (1U << 5),
kFTM_Chnl6Match = (1U << 6),
kFTM_Chnl7Match = (1U << 7),
kFTM_CntMax = (1U << 8),
kFTM_CntMin = (1U << 9),
kFTM_HalfCycMatch = (1U << 10) }
FTM options available as loading point for register reload.

- enum ftm_interrupt_enable_t {
kFTM_Chnl0InterruptEnable = (1U << 0),
kFTM_Chnl1InterruptEnable = (1U << 1),
kFTM_Chnl2InterruptEnable = (1U << 2),
kFTM_Chnl3InterruptEnable = (1U << 3),
kFTM_Chnl4InterruptEnable = (1U << 4),
kFTM_Chnl5InterruptEnable = (1U << 5),
kFTM_Chnl6InterruptEnable = (1U << 6),
kFTM_Chnl7InterruptEnable = (1U << 7),
kFTM_FaultInterruptEnable = (1U << 8),
kFTM_TimeOverflowInterruptEnable = (1U << 9),
kFTM_ReloadInterruptEnable = (1U << 10) }
List of FTM interrupts.
- enum ftm_status_flags_t {
kFTM_Chnl0Flag = (1U << 0),
kFTM_Chnl1Flag = (1U << 1),
kFTM_Chnl2Flag = (1U << 2),
kFTM_Chnl3Flag = (1U << 3),
kFTM_Chnl4Flag = (1U << 4),
kFTM_Chnl5Flag = (1U << 5),
kFTM_Chnl6Flag = (1U << 6),
kFTM_Chnl7Flag = (1U << 7),
kFTM_FaultFlag = (1U << 8),
kFTM_TimeOverflowFlag = (1U << 9),
kFTM_ChnlTriggerFlag = (1U << 10),
kFTM_ReloadFlag = (1U << 11) }
List of FTM flags.
- enum {
kFTM_QuadDecoderCountingIncreaseFlag = FTM_QDCTRL_QUADIR_MASK,
kFTM_QuadDecoderCountingOverflowOnTopFlag = FTM_QDCTRL_TOFDIR_MASK }
List of FTM Quad Decoder flags.

Functions

- void FTM_SetupFaultInput (FTM_Type *base, ftm_fault_input_t faultNumber, const ftm_fault_param_t *faultParams)
  Sets up the working of the FTM fault inputs protection.
- static void FTM_SetGlobalTimeBaseOutputEnable (FTM_Type *base, bool enable)
  Enables or disables the FTM global time base signal generation to other FTMs.
- static void FTM_SetOutputMask (FTM_Type *base, ftm_chnl_t chnlNumber, bool mask)
  Sets the FTM peripheral timer channel output mask.
- static void FTM_SetPwmOutputEnable (FTM_Type *base, ftm_chnl_t chnlNumber, bool value)
  Allows users to enable an output on an FTM channel.
- static void FTM_SetSoftwareTrigger (FTM_Type *base, bool enable)
  Enables or disables the FTM software trigger for PWM synchronization.
- static void FTM_SetWriteProtection (FTM_Type *base, bool enable)
  Enables or disables the FTM write protection.

Driver version

- #define FSL_FTM_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))
  FTM driver version 2.5.0.

Initialization and deinitialization

- status_t FTM_Init (FTM_Type *base, const ftm_config_t *config)
  Ungates the FTM clock and configures the peripheral for basic operation.
- void FTM_Deinit (FTM_Type *base)
  Gates the FTM clock.
- void FTM_GetDefaultConfig (ftm_config_t *config)
  Fills in the FTM configuration structure with the default settings.
- static ftm_clock_prescale_t FTM_CalculateCounterClkDiv (FTM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)
  brief Calculates the counter clock prescaler.

Channel mode operations

- status_t FTM_SetupPwm (FTM_Type *base, const ftm_chnl_pwm_signal_param_t *chnlParams, uint8_t numOfChnls, ftm_pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)
  Configures the PWM signal parameters.
- status_t FTM_UpdatePwmDutycycle (FTM_Type *base, ftm_chnl_t chnlNumber, ftm_pwm_mode_t currentPwmMode, uint8_t dutyCyclePercent)
  Updates the duty cycle of an active PWM signal.
- void FTM_UpdateChnlEdgeLevelSelect (FTM_Type *base, ftm_chnl_t chnlNumber, uint8_t level)
  Updates the edge level selection for a channel.
- status_t FTM_SetupPwmMode (FTM_Type *base, const ftm_chnl_pwm_config_param_t *chnlParams, uint8_t numOfChnls, ftm_pwm_mode_t mode)
  Configures the PWM mode parameters.
- void FTM_SetupInputCapture (FTM_Type *base, ftm_chnl_t chnlNumber, ftm_input_capture_edge_t captureMode, uint32_t filterValue)
  Enables capturing an input signal on the channel using the function parameters.
- void FTM_SetupOutputCompare (FTM_Type *base, ftm_chnl_t chnlNumber, ftm_output_compare_mode_t compareMode, uint32_t compareValue)

Configures the FTM to generate timed pulses.
- void FTM_SetupDualEdgeCapture (FTM_Type ∗base, ftm_chnl_t chnlPairNumber, const ftm_dual_edge_capture_param_t ∗edgeParam, uint32_t filterValue)
  Configures the dual edge capture mode of the FTM.

Interrupt Interface

- void FTM_EnableInterrupts (FTM_Type ∗base, uint32_t mask)
  Enables the selected FTM interrupts.
- void FTM_DisableInterrupts (FTM_Type ∗base, uint32_t mask)
  Disables the selected FTM interrupts.
- uint32_t FTM_GetEnabledInterrupts (FTM_Type ∗base)
  Gets the enabled FTM interrupts.

Status Interface

- uint32_t FTM_GetStatusFlags (FTM_Type ∗base)
  Gets the FTM status flags.
- void FTM_ClearStatusFlags (FTM_Type ∗base, uint32_t mask)
  Clears the FTM status flags.

Read and write the timer period

- static void FTM_SetTimerPeriod (FTM_Type ∗base, uint32_t ticks)
  Sets the timer period in units of ticks.
- static uint32_t FTM_GetCurrentTimerCount (FTM_Type ∗base)
  Reads the current timer counting value.
- static uint32_t FTM_GetInputCaptureValue (FTM_Type ∗base, ftm_chnl_t chnlNumber)
  Reads the captured value.

Timer Start and Stop

- static void FTM_StartTimer (FTM_Type ∗base, ftm_clock_source_t clockSource)
  Starts the FTM counter.
- static void FTM_StopTimer (FTM_Type ∗base)
  Stops the FTM counter.

Software output control

- static void FTM_SetSoftwareCtrlEnable (FTM_Type ∗base, ftm_chnl_t chnlNumber, bool value)
  Enables or disables the channel software output control.
- static void FTM_SetSoftwareCtrlVal (FTM_Type ∗base, ftm_chnl_t chnlNumber, bool value)
  Sets the channel software output control value.

Channel pair operations

- static void FTM_SetFaultControlEnable (FTM_Type ∗base, ftm_chnl_t chnlPairNumber, bool value)
  This function enables/disables the fault control in a channel pair.
- static void FTM_SetDeadTimeEnable (FTM_Type ∗base, ftm_chnl_t chnlPairNumber, bool value)

This function enables/disables the dead time insertion in a channel pair.
- static void FTM_SetComplementaryEnable (FTM_Type ∗base, ftm_chnl_t chnlPairNumber, bool value)
  This function enables/disables complementary mode in a channel pair.
- static void FTM_SetInvertEnable (FTM_Type ∗base, ftm_chnl_t chnlPairNumber, bool value)
  This function enables/disables inverting control in a channel pair.

Quad Decoder

- void FTM_SetupQuadDecode (FTM_Type ∗base, const ftm_phase_params_t ∗phaseAParams, const ftm_phase_params_t ∗phaseBParams, ftm_quad_decode_mode_t quadMode)
  Configures the parameters and activates the quadrature decoder mode.
- static uint32_t FTM_GetQuadDecoderFlags (FTM_Type ∗base)
  Gets the FTM Quad Decoder flags.
- static void FTM_SetQuadDecoderModuloValue (FTM_Type ∗base, uint32_t startValue, uint32_t overValue)
  Sets the modulo values for Quad Decoder.
- static uint32_t FTM_GetQuadDecoderCounterValue (FTM_Type ∗base)
  Gets the current Quad Decoder counter value.
- static void FTM_ClearQuadDecoderCounterValue (FTM_Type ∗base)
  Clears the current Quad Decoder counter value.

9.5 Data Structure Documentation

9.5.1 struct ftm_chnl_pwm_signal_param_t

Data Fields

- ftm_chnl_t chnlNumber
  The channel/channel pair number.
- ftm_pwm_level_select_t level
  PWM output active level select.
- uint8_t dutyCyclePercent
  PWM pulse width, value should be between 0 to 100 0 = inactive signal(0% duty cycle)...
- uint8_t firstEdgeDelayPercent
  Used only in kFTM_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM.
- bool enableComplementary
  Used only in combined PWM mode.
- bool enableDeadtime
  Used only in combined PWM mode with enable complementary.

Field Documentation

(1) ftm_chnl_t ftm_chnl_pwm_signal_param_t::chnlNumber

In combined mode, this represents the channel pair number.

(2) ftm_pwm_level_select_t ftm_chnl_pwm_signal_param_t::level

(3) uint8_t ftm_chnl_pwm_signal_param_t::dutyCyclePercent

100 = always active signal (100% duty cycle).

(4) uint8_t ftm_chnl_pwm_signal_param_t::firstEdgeDelayPercent

Specifies the delay to the first edge in a PWM period. If unsure leave as 0; Should be specified as a percentage of the PWM period

(5) bool ftm_chnl_pwm_signal_param_t::enableComplementary

true: The combined channels output complementary signals; false: The combined channels output same signals;

(6) bool ftm_chnl_pwm_signal_param_t::enableDeadtime

true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

9.5.2 struct ftm_chnl_pwm_config_param_t

Data Fields

- ftm_chnl_t chnlNumber
  The channel/channel pair number.
- ftm_pwm_level_select_t level
  PWM output active level select.
- uint16_t dutyValue
  PWM pulse width, the uint of this value is timer ticks.
- uint16_t firstEdgeValue
  Used only in kFTM_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM.
- bool enableComplementary
  Used only in combined PWM mode.
- bool enableDeadtime
  Used only in combined PWM mode with enable complementary.

Field Documentation

(1) ftm_chnl_t ftm_chnl_pwm_config_param_t::chnlNumber

In combined mode, this represents the channel pair number.

(2) ftm_pwm_level_select_t ftm_chnl_pwm_config_param_t::level

(3) uint16_t ftm_chnl_pwm_config_param_t::dutyValue

(4) uint16_t ftm_chnl_pwm_config_param_t::firstEdgeValue

Specifies the delay to the first edge in a PWM period. If unsure leave as 0, uint of this value is timer ticks.

(5) bool ftm_chnl_pwm_config_param_t::enableComplementary

true: The combined channels output complementary signals; false: The combined channels output same signals;

(6) bool ftm_chnl_pwm_config_param_t::enableDeadtime

true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

9.5.3 struct ftm_dual_edge_capture_param_t

Data Fields

- ftm_dual_edge_capture_mode_t mode
  Dual Edge Capture mode.
- ftm_input_capture_edge_t currChanEdgeMode
  Input capture edge select for channel n.
- ftm_input_capture_edge_t nextChanEdgeMode
  Input capture edge select for channel n+1.

9.5.4 struct ftm_phase_params_t

Data Fields

- bool enablePhaseFilter
  True: enable phase filter; false: disable filter.
- uint32_t phaseFilterVal
  Filter value, used only if phase filter is enabled.
- ftm_phase_polarity_t phasePolarity
  Phase polarity.

9.5.5 struct ftm_fault_param_t

Data Fields

- bool enableFaultInput
  True: Fault input is enabled; false: Fault input is disabled.
- bool faultLevel
  True: Fault polarity is active low; in other words, '0' indicates a fault; False: Fault polarity is active high.
- bool useFaultFilter
  True: Use the filtered fault signal; False: Use the direct path from fault input.

9.5.6 struct ftm_config_t

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

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

Data Fields

- ftm_clock_prescale_t prescale
  FTM clock prescale value.
- ftm_bdm_mode_t bdmMode
  FTM behavior in BDM mode.
- uint32_t pwmSyncMode
  Synchronization methods to use to update buffered registers; Multiple update modes can be used by providing an OR'ed list of options available in enumeration ftm_pwm_sync_method_t.
- uint32_t reloadPoints
  FTM reload points; When using this, the PWM synchronization is not required.
- ftm_fault_mode_t faultMode
  FTM fault control mode.
- uint8_t faultFilterValue
  Fault input filter value.
- ftm_deadtime_prescale_t deadTimePrescale
  The dead time prescalar value.
- uint32_t deadTimeValue
  The dead time value deadTimeValue's available range is 0-1023 when register has DTVALEX, otherwise its available range is 0-63.
- uint32_t extTriggers
  External triggers to enable.
- uint8_t chnlInitState
  Defines the initialization value of the channels in OUTINT register.
- uint8_t chnlPolarity
  Defines the output polarity of the channels in POL register.
- bool useGlobalTimeBase
  True: Use of an external global time base is enabled; False: disabled.

Field Documentation

(1) uint32_t ftm_config_t::pwmSyncMode

(2) uint32_t ftm_config_t::reloadPoints

Multiple reload points can be used by providing an OR'ed list of options available in enumeration ftm_reload_point_t.

(3) uint32_t ftm_config_t::deadTimeValue

(4) uint32_t ftm_config_t::extTriggers

Multiple trigger sources can be enabled by providing an OR'ed list of options available in enumeration ftm_external_trigger_t.

9.6 Macro Definition Documentation

9.6.1 #define FSL_FTM_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))

9.7 Enumeration Type Documentation

9.7.1 enum ftm_chnl_t

Note

Actual number of available channels is SoC dependent

Enumerator

kFTM_Chnl_0 FTM channel number 0.
kFTM_Chnl_1 FTM channel number 1.
kFTM_Chnl_2 FTM channel number 2.
kFTM_Chnl_3 FTM channel number 3.
kFTM_Chnl_4 FTM channel number 4.
kFTM_Chnl_5 FTM channel number 5.
kFTM_Chnl_6 FTM channel number 6.
kFTM_Chnl_7 FTM channel number 7.

9.7.2 enum ftm_fault_input_t

Enumerator

kFTM_Fault_0 FTM fault 0 input pin.
kFTM_Fault_1 FTM fault 1 input pin.
kFTM_Fault_2 FTM fault 2 input pin.
kFTM_Fault_3 FTM fault 3 input pin.

9.7.3 enum ftm_pwm_mode_t

Enumerator

kFTM_EdgeAlignedPwm Edge-aligned PWM.
kFTM_CenterAlignedPwm Center-aligned PWM.
kFTM_EdgeAlignedCombinedPwm Edge-aligned combined PWM.

kFTM_CenterAlignedCombinedPwm Center-aligned combined PWM.
kFTM_AsymmetricalCombinedPwm Asymmetrical combined PWM.

9.7.4 enum ftm_pwm_level_select_t

Enumerator

kFTM_NoPwmSignal No PWM output on pin.
kFTM_LowTrue Low true pulses.
kFTM_HighTrue High true pulses.

9.7.5 enum ftm_output_compare_mode_t

Enumerator

kFTM_NoOutputSignal No channel output when counter reaches CnV.
kFTM_ToggleOnMatch Toggle output.
kFTM_ClearOnMatch Clear output.
kFTM_SetOnMatch Set output.

9.7.6 enum ftm_input_capture_edge_t

Enumerator

kFTM_RisingEdge Capture on rising edge only.
kFTM_FallingEdge Capture on falling edge only.
kFTM_RiseAndFallEdge Capture on rising or falling edge.

9.7.7 enum ftm_dual_edge_capture_mode_t

Enumerator

kFTM_OneShot One-shot capture mode.
kFTM_Continuous Continuous capture mode.

9.7.8 enum ftm_quad_decode_mode_t

Enumerator

kFTM_QuadPhaseEncode Phase A and Phase B encoding mode.
kFTM_QuadCountAndDir Count and direction encoding mode.

9.7.9 enum ftm_phase_polarity_t

Enumerator

kFTM_QuadPhaseNormal Phase input signal is not inverted.
kFTM_QuadPhaseInvert Phase input signal is inverted.

9.7.10 enum ftm_deadtime_prescale_t

Enumerator

kFTM_Deadtime_Prescale_1 Divide by 1.
kFTM_Deadtime_Prescale_4 Divide by 4.
kFTM_Deadtime_Prescale_16 Divide by 16.

9.7.11 enum ftm_clock_source_t

Enumerator

kFTM_SystemClock System clock selected.
kFTM_FixedClock Fixed frequency clock.
kFTM_ExternalClock External clock.

9.7.12 enum ftm_clock_prescale_t

Enumerator

kFTM_Prescale_Divide_1 Divide by 1.
kFTM_Prescale_Divide_2 Divide by 2.
kFTM_Prescale_Divide_4 Divide by 4.
kFTM_Prescale_Divide_8 Divide by 8.
kFTM_Prescale_Divide_16 Divide by 16.
kFTM_Prescale_Divide_32 Divide by 32.
kFTM_Prescale_Divide_64 Divide by 64.
kFTM_Prescale_Divide_128 Divide by 128.

9.7.13 enum ftm_bdm_mode_t

Enumerator

kFTM_BdmMode_0 FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.

kFTM_BdmMode_1 FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers.
kFTM_BdmMode_2 FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.
kFTM_BdmMode_3 FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode.

9.7.14 enum ftm_fault_mode_t

Enumerator

kFTM_Fault_Disable Fault control is disabled for all channels.
kFTM_Fault_EvenChnls Enabled for even channels only(0,2,4,6) with manual fault clearing.
kFTM_Fault_AllChnlsMan Enabled for all channels with manual fault clearing.
kFTM_Fault_AllChnlsAuto Enabled for all channels with automatic fault clearing.

9.7.15 enum ftm_external_trigger_t

Note

Actual available external trigger sources are SoC-specific

Enumerator

kFTM_Chnl0Trigger Generate trigger when counter equals chnl 0 CnV reg.
kFTM_Chnl1Trigger Generate trigger when counter equals chnl 1 CnV reg.
kFTM_Chnl2Trigger Generate trigger when counter equals chnl 2 CnV reg.
kFTM_Chnl3Trigger Generate trigger when counter equals chnl 3 CnV reg.
kFTM_Chnl4Trigger Generate trigger when counter equals chnl 4 CnV reg.
kFTM_Chnl5Trigger Generate trigger when counter equals chnl 5 CnV reg.
kFTM_InitTrigger Generate Trigger when counter is updated with CNTIN.
kFTM_ReloadInitTrigger Available on certain SoC's, trigger on reload point.

9.7.16 enum ftm_pwm_sync_method_t

Enumerator

kFTM_SoftwareTrigger Software triggers PWM sync.
kFTM_HardwareTrigger_0 Hardware trigger 0 causes PWM sync.
kFTM_HardwareTrigger_1 Hardware trigger 1 causes PWM sync.
kFTM_HardwareTrigger_2 Hardware trigger 2 causes PWM sync.

9.7.17 enum ftm_reload_point_t

Note

Actual available reload points are SoC-specific

Enumerator

kFTM_Chnl0Match Channel 0 match included as a reload point.
kFTM_Chnl1Match Channel 1 match included as a reload point.
kFTM_Chnl2Match Channel 2 match included as a reload point.
kFTM_Chnl3Match Channel 3 match included as a reload point.
kFTM_Chnl4Match Channel 4 match included as a reload point.
kFTM_Chnl5Match Channel 5 match included as a reload point.
kFTM_Chnl6Match Channel 6 match included as a reload point.
kFTM_Chnl7Match Channel 7 match included as a reload point.
kFTM_CntMax Use in up-down count mode only, reload when counter reaches the maximum value.

kFTM_CntMin Use in up-down count mode only, reload when counter reaches the minimum value.

kFTM_HalfCycMatch Available on certain SoC's, half cycle match reload point.

9.7.18 enum ftm_interrupt_enable_t

Note

Actual available interrupts are SoC-specific

Enumerator

kFTM_Chnl0InterruptEnable Channel 0 interrupt.
kFTM_Chnl1InterruptEnable Channel 1 interrupt.
kFTM_Chnl2InterruptEnable Channel 2 interrupt.
kFTM_Chnl3InterruptEnable Channel 3 interrupt.
kFTM_Chnl4InterruptEnable Channel 4 interrupt.
kFTM_Chnl5InterruptEnable Channel 5 interrupt.
kFTM_Chnl6InterruptEnable Channel 6 interrupt.
kFTM_Chnl7InterruptEnable Channel 7 interrupt.
kFTM_FaultInterruptEnable Fault interrupt.
kFTM_TimeOverflowInterruptEnable Time overflow interrupt.
kFTM_ReloadInterruptEnable Reload interrupt; Available only on certain SoC's.

9.7.19 enum ftm_status_flags_t

Note

Actual available flags are SoC-specific

Enumerator

kFTM_Chnl0Flag Channel 0 Flag.
kFTM_Chnl1Flag Channel 1 Flag.
kFTM_Chnl2Flag Channel 2 Flag.
kFTM_Chnl3Flag Channel 3 Flag.
kFTM_Chnl4Flag Channel 4 Flag.
kFTM_Chnl5Flag Channel 5 Flag.
kFTM_Chnl6Flag Channel 6 Flag.
kFTM_Chnl7Flag Channel 7 Flag.
kFTM_FaultFlag Fault Flag.
kFTM_TimeOverflowFlag Time overflow Flag.
kFTM_ChnlTriggerFlag Channel trigger Flag.
kFTM_ReloadFlag Reload Flag; Available only on certain SoC's.

9.7.20 anonymous enum

Enumerator

kFTM_QuadDecoderCountingIncreaseFlag Counting direction is increasing (FTM counter increment), or the direction is decreasing.
kFTM_QuadDecoderCountingOverflowOnTopFlag Indicates if the TOF bit was set on the top or the bottom of counting.

9.8 Function Documentation

9.8.1 status_t FTM_Init (FTM_Type ∗ *base*, const ftm_config_t ∗ *config*)

Note

This API should be called at the beginning of the application which is using the FTM driver. If the FTM instance has only TPM features, please use the TPM driver.

Parameters

base	FTM peripheral base address

config	Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

9.8.2 void FTM_Deinit (FTM_Type * *base*)

Parameters

base	FTM peripheral base address

9.8.3 void FTM_GetDefaultConfig (ftm_config_t * *config*)

The default values are:

```
*   config->prescale = kFTM_Prescale_Divide_1;
*   config->bdmMode = kFTM_BdmMode_0;
*   config->pwmSyncMode = kFTM_SoftwareTrigger;
*   config->reloadPoints = 0;
*   config->faultMode = kFTM_Fault_Disable;
*   config->faultFilterValue = 0;
*   config->deadTimePrescale = kFTM_Deadtime_Prescale_1;
*   config->deadTimeValue =  0;
*   config->extTriggers = 0;
*   config->chnlInitState = 0;
*   config->chnlPolarity = 0;
*   config->useGlobalTimeBase = false;
*
```

Parameters

config	Pointer to the user configuration structure.

9.8.4 static ftm_clock_prescale_t FTM_CalculateCounterClkDiv (FTM_Type * *base*, uint32_t *counterPeriod_Hz*, uint32_t *srcClock_Hz*) [inline], [static]

This function calculates the values for SC[PS] bit.

param base FTM peripheral base address param counterPeriod_Hz The desired frequency in Hz which corresponding to the time when the counter reaches the mod value param srcClock_Hz FTM counter clock in Hz

return Calculated clock prescaler value, see ftm_clock_prescale_t.

9.8.5 status_t FTM_SetupPwm (FTM_Type ∗ *base*, const ftm_chnl_pwm_signal_param_t ∗ *chnlParams*, uint8_t *numOfChnls*, ftm_pwm_mode_t *mode*, uint32_t *pwmFreq_Hz*, uint32_t *srcClock_Hz*)

Call this function to configure the PWM signal period, mode, duty cycle, and edge. Use this function to configure all FTM channels that are used to output a PWM signal.

Parameters

base	FTM peripheral base address
chnlParams	Array of PWM channel parameters to configure the channel(s)
numOfChnls	Number of channels to configure; This should be the size of the array passed in
mode	PWM operation mode, options available in enumeration ftm_pwm_mode_t
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	FTM counter clock in Hz

Returns

kStatus_Success if the PWM setup was successful kStatus_Error on failure

9.8.6 status_t FTM_UpdatePwmDutycycle (FTM_Type ∗ *base*, ftm_chnl_t *chnlNumber*, ftm_pwm_mode_t *currentPwmMode*, uint8_t *dutyCyclePercent*)

Parameters

base	FTM peripheral base address
chnlNumber	The channel/channel pair number. In combined mode, this represents the channel pair number
currentPwm-Mode	The current PWM mode set during PWM setup
dutyCycle-Percent	New PWM pulse width; The value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns

kStatus_Success if the PWM update was successful kStatus_Error on failure

9.8.7 void FTM_UpdateChnlEdgeLevelSelect (FTM_Type ∗ *base*, ftm_chnl_t *chnlNumber*, uint8_t *level*)

Parameters

base	FTM peripheral base address
chnlNumber	The channel number
level	The level to be set to the ELSnB:ELSnA field; Valid values are 00, 01, 10, 11. See the Kinetis SoC reference manual for details about this field.

9.8.8 status_t FTM_SetupPwmMode (FTM_Type ∗ *base*, const ftm_chnl_pwm_-config_param_t ∗ *chnlParams*, uint8_t *numOfChnls*, ftm_pwm_mode_t *mode*)

Call this function to configure the PWM signal mode, duty cycle in ticks, and edge. Use this function to configure all FTM channels that are used to output a PWM signal. Please note that: This API is similar with FTM_SetupPwm() API, but will not set the timer period, and this API will set channel match value in timer ticks, not period percent.

Parameters

base	FTM peripheral base address
chnlParams	Array of PWM channel parameters to configure the channel(s)
numOfChnls	Number of channels to configure; This should be the size of the array passed in
mode	PWM operation mode, options available in enumeration ftm_pwm_mode_t

Returns

kStatus_Success if the PWM setup was successful kStatus_Error on failure

9.8.9 void FTM_SetupInputCapture (FTM_Type ∗ *base*, ftm_chnl_t *chnlNumber*, ftm_input_capture_edge_t *captureMode*, uint32_t *filterValue*)

When the edge specified in the captureMode argument occurs on the channel, the FTM counter is captured into the CnV register. The user has to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only for channels 0, 1, 2, 3.

Parameters

base	FTM peripheral base address
chnlNumber	The channel number
captureMode	Specifies which edge to capture
filterValue	Filter value, specify 0 to disable filter. Available only for channels 0-3.

9.8.10 void FTM_SetupOutputCompare (FTM_Type * *base*, ftm_chnl_t *chnlNumber*, ftm_output_compare_mode_t *compareMode*, uint32_t *compareValue*)

When the FTM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

base	FTM peripheral base address
chnlNumber	The channel number
compareMode	Action to take on the channel output when the compare condition is met
compareValue	Value to be programmed in the CnV register.

9.8.11 void FTM_SetupDualEdgeCapture (FTM_Type * *base*, ftm_chnl_t *chnlPairNumber*, const ftm_dual_edge_capture_param_t * *edgeParam*, uint32_t *filterValue*)

This function sets up the dual edge capture mode on a channel pair. The capture edge for the channel pair and the capture mode (one-shot or continuous) is specified in the parameter argument. The filter function is disabled if the filterVal argument passed is zero. The filter function is available only on channels 0 and 2. The user has to read the channel CnV registers separately to get the capture values.

Parameters

base	FTM peripheral base address
chnlPair-Number	The FTM channel pair number; options are 0, 1, 2, 3

edgeParam	Sets up the dual edge capture function
filterValue	Filter value, specify 0 to disable filter. Available only for channel pair 0 and 1.

9.8.12 void FTM_SetupFaultInput (FTM_Type ∗ *base*, ftm_fault_input_t *faultNumber*, const ftm_fault_param_t ∗ *faultParams*)

FTM can have up to 4 fault inputs. This function sets up fault parameters, fault level, and input filter.

Parameters

base	FTM peripheral base address
faultNumber	FTM fault to configure.
faultParams	Parameters passed in to set up the fault

9.8.13 void FTM_EnableInterrupts (FTM_Type ∗ *base*, uint32_t *mask*)

Parameters

base	FTM peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration ftm_-interrupt_enable_t

9.8.14 void FTM_DisableInterrupts (FTM_Type ∗ *base*, uint32_t *mask*)

Parameters

base	FTM peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration ftm_-interrupt_enable_t

9.8.15 uint32_t FTM_GetEnabledInterrupts (FTM_Type ∗ *base*)

Parameters

base	FTM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration ftm_interrupt_enable-_t

9.8.16 uint32_t FTM_GetStatusFlags (FTM_Type * *base*)

Parameters

base	FTM peripheral base address

Returns

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

9.8.17 void FTM_ClearStatusFlags (FTM_Type * *base*, uint32_t *mask*)

Parameters

base	FTM peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration ftm_-status_flags_t

9.8.18 static void FTM_SetTimerPeriod (FTM_Type * *base*, uint32_t *ticks*) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note

1. This API allows the user to use the FTM module as a timer. Do not mix usage of this API with FTM's PWM setup API's.
2. Call the utility macros provided in the fsl_common.h to convert usec or msec to ticks.

Parameters

base	FTM peripheral base address
ticks	A timer period in units of ticks, which should be equal or greater than 1.

9.8.19 static uint32_t FTM_GetCurrentTimerCount (FTM_Type ∗ *base*) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

base	FTM peripheral base address

Returns

The current counter value in ticks

9.8.20 static uint32_t FTM_GetInputCaptureValue (FTM_Type ∗ *base,* ftm_chnl_t *chnlNumber*) [inline], [static]

This function returns the captured value of a FTM channel configured in input capture or dual edge capture mode.

Note

Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

base	FTM peripheral base address

chnlNumber	Channel to be read

Returns

The captured FTM counter value of the input modes.

9.8.21 static void FTM_StartTimer (FTM_Type * *base*, ftm_clock_source_t *clockSource*) [inline], [static]

Parameters

base	FTM peripheral base address
clockSource	FTM clock source; After the clock source is set, the counter starts running.

9.8.22 static void FTM_StopTimer (FTM_Type * *base*) [inline], [static]

Parameters

base	FTM peripheral base address

9.8.23 static void FTM_SetSoftwareCtrlEnable (FTM_Type * *base*, ftm_chnl_t *chnlNumber*, bool *value*) [inline], [static]

Parameters

base	FTM peripheral base address
chnlNumber	Channel to be enabled or disabled
value	true: channel output is affected by software output control false: channel output is unaffected by software output control

9.8.24 static void FTM_SetSoftwareCtrlVal (FTM_Type * *base*, ftm_chnl_t *chnlNumber*, bool *value*) [inline], [static]

Parameters

base	FTM peripheral base address.
chnlNumber	Channel to be configured
value	true to set 1, false to set 0

9.8.25 static void FTM_SetGlobalTimeBaseOutputEnable (FTM_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	FTM peripheral base address
enable	true to enable, false to disable

9.8.26 static void FTM_SetOutputMask (FTM_Type * *base*, ftm_chnl_t *chnlNumber*, bool *mask*) [inline], [static]

Parameters

base	FTM peripheral base address
chnlNumber	Channel to be configured
mask	true: masked, channel is forced to its inactive state; false: unmasked

9.8.27 static void FTM_SetPwmOutputEnable (FTM_Type * *base*, ftm_chnl_t *chnlNumber*, bool *value*) [inline], [static]

To enable the PWM channel output call this function with val=true. For input mode, call this function with val=false.

Parameters

base	FTM peripheral base address

chnlNumber	Channel to be configured
value	true: enable output; false: output is disabled, used in input mode

9.8.28 static void FTM_SetFaultControlEnable (FTM_Type ∗ *base*, ftm_chnl_t *chnlPairNumber*, bool *value*) [inline], [static]

Parameters

base	FTM peripheral base address
chnlPair-Number	The FTM channel pair number; options are 0, 1, 2, 3
value	true: Enable fault control for this channel pair; false: No fault control

9.8.29 static void FTM_SetDeadTimeEnable (FTM_Type ∗ *base*, ftm_chnl_t *chnlPairNumber*, bool *value*) [inline], [static]

Parameters

base	FTM peripheral base address
chnlPair-Number	The FTM channel pair number; options are 0, 1, 2, 3
value	true: Insert dead time in this channel pair; false: No dead time inserted

9.8.30 static void FTM_SetComplementaryEnable (FTM_Type ∗ *base*, ftm_chnl_t *chnlPairNumber*, bool *value*) [inline], [static]

Parameters

base	FTM peripheral base address
chnlPair-Number	The FTM channel pair number; options are 0, 1, 2, 3

value	true: enable complementary mode; false: disable complementary mode

9.8.31 static void FTM_SetInvertEnable (FTM_Type * *base,* ftm_chnl_t *chnlPairNumber,* bool *value*) [inline], [static]

Parameters

base	FTM peripheral base address
chnlPair-Number	The FTM channel pair number; options are 0, 1, 2, 3
value	true: enable inverting; false: disable inverting

9.8.32 void FTM_SetupQuadDecode (FTM_Type * *base,* const ftm_phase_-params_t * *phaseAParams,* const ftm_phase_params_t * *phaseBParams,* ftm_quad_decode_mode_t *quadMode*)

Parameters

base	FTM peripheral base address
phaseAParams	Phase A configuration parameters
phaseBParams	Phase B configuration parameters
quadMode	Selects encoding mode used in quadrature decoder mode

9.8.33 static uint32_t FTM_GetQuadDecoderFlags (FTM_Type * *base*) [inline], [static]

Parameters

base	FTM peripheral base address.

Returns

Flag mask of FTM Quad Decoder, see _ftm_quad_decoder_flags.

9.8.34 static void FTM_SetQuadDecoderModuloValue (FTM_Type * *base,* uint32_t *startValue,* uint32_t *overValue*) [inline], [static]

The modulo values configure the minimum and maximum values that the Quad decoder counter can reach. After the counter goes over, the counter value goes to the other side and decrease/increase again.

Parameters

base	FTM peripheral base address.
startValue	The low limit value for Quad Decoder counter.
overValue	The high limit value for Quad Decoder counter.

9.8.35 static uint32_t FTM_GetQuadDecoderCounterValue (FTM_Type * *base*) [inline], [static]

Parameters

base	FTM peripheral base address.

Returns

Current quad Decoder counter value.

9.8.36 static void FTM_ClearQuadDecoderCounterValue (FTM_Type * *base*) [inline], [static]

The counter is set as the initial value.

Parameters

base	FTM peripheral base address.

9.8.37 static void FTM_SetSoftwareTrigger (FTM_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	FTM peripheral base address
enable	true: software trigger is selected, false: software trigger is not selected

9.8.38 static void FTM_SetWriteProtection (FTM_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	FTM peripheral base address
enable	true: Write-protection is enabled, false: Write-protection is disabled

Chapter 10
GPIO: General-Purpose Input/Output Driver

10.1 Overview

Modules

- FGPIO Driver
- GPIO Driver

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 }
  GPIO direction definition.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  GPIO driver version.

10.2 Data Structure Documentation

10.2.1 struct gpio_pin_config_t

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Data Fields

- gpio_pin_direction_t pinDirection
  GPIO direction, input or output.
- uint8_t outputLogic
  Set a default output logic, which has no use in input.

10.3 Macro Definition Documentation

10.3.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

10.4 Enumeration Type Documentation

10.4.1 enum gpio_pin_direction_t

Enumerator

kGPIO_DigitalInput Set current pin as digital input.
kGPIO_DigitalOutput Set current pin as digital output.

10.5 GPIO Driver

10.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

10.5.2 Typical use case

10.5.2.1 Output Operation

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

10.5.2.2 Input Operation

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

GPIO Configuration

- void GPIO_PinInit (GPIO_Type ∗base, 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 pin, uint8_t output)
  Sets the output level of the multiple GPIO pins to the logic 1 or 0.
- static void GPIO_PortSet (GPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 1.
- static void GPIO_PortClear (GPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 0.
- static void GPIO_PortToggle (GPIO_Type ∗base, uint32_t mask)
  Reverses the current output logic of the multiple GPIO pins.

GPIO Input Operations

- static uint32_t GPIO_PinRead (GPIO_Type ∗base, uint32_t pin)
  Reads the current input value of the GPIO port.

GPIO Interrupt

- uint32_t GPIO_PortGetInterruptFlags (GPIO_Type ∗base)
  Reads the GPIO port interrupt status flag.

- void GPIO_PortClearInterruptFlags (GPIO_Type *base, uint32_t mask)
  Clears multiple GPIO pin interrupt status flags.

10.5.3 Function Documentation

10.5.3.1 void GPIO_PinInit (GPIO_Type * *base*, uint32_t *pin*, const gpio_pin_config_t * *config*)

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
* Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO port pin number
config	GPIO pin configuration pointer

10.5.3.2 static void GPIO_PinWrite (GPIO_Type * *base*, uint32_t *pin*, uint8_t *output*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO pin number

output	GPIO pin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

10.5.3.3 static void GPIO_PortSet (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

10.5.3.4 static void GPIO_PortClear (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

10.5.3.5 static void GPIO_PortToggle (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

10.5.3.6 static uint32_t GPIO_PinRead (GPIO_Type * *base*, uint32_t *pin*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO pin number

Return values

GPIO	port input value • 0: corresponding pin input low-logic level. • 1: corresponding pin input high-logic level.

10.5.3.7 uint32_t GPIO_PortGetInterruptFlags (GPIO_Type ∗ *base*)

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

Return values

The	current GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

10.5.3.8 void GPIO_PortClearInterruptFlags (GPIO_Type ∗ *base,* uint32_t *mask*)

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

10.6 FGPIO Driver

10.6.1 Overview

This section describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the GPIO application.

Note

> FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and GPIO share the same peripheral but use different registers. FGPIO is closer to the core than the regular GPIO and it's faster to read and write.

10.6.2 Typical use case

10.6.2.1 Output Operation

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

10.6.2.2 Input Operation

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

FGPIO Configuration

- void FGPIO_PinInit (FGPIO_Type ∗base, uint32_t pin, const gpio_pin_config_t ∗config)
  Initializes a FGPIO pin used by the board.

FGPIO Output Operations

- static void FGPIO_PinWrite (FGPIO_Type ∗base, uint32_t pin, uint8_t output)
  Sets the output level of the multiple FGPIO pins to the logic 1 or 0.
- static void FGPIO_PortSet (FGPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple FGPIO pins to the logic 1.
- static void FGPIO_PortClear (FGPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple FGPIO pins to the logic 0.
- static void FGPIO_PortToggle (FGPIO_Type ∗base, uint32_t mask)
  Reverses the current output logic of the multiple FGPIO pins.

FGPIO Input Operations

- static uint32_t FGPIO_PinRead (FGPIO_Type ∗base, uint32_t pin)
  Reads the current input value of the FGPIO port.

FGPIO Interrupt

- uint32_t FGPIO_PortGetInterruptFlags (FGPIO_Type *base)
  Reads the FGPIO port interrupt status flag.
- void FGPIO_PortClearInterruptFlags (FGPIO_Type *base, uint32_t mask)
  Clears the multiple FGPIO pin interrupt status flag.

10.6.3 Function Documentation

10.6.3.1 void FGPIO_PinInit (FGPIO_Type * *base*, uint32_t *pin*, const gpio_pin_config_t * *config*)

To initialize the FGPIO driver, define a pin configuration, as either input or output, in the user file. Then, call the FGPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration:

```
* Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
pin	FGPIO port pin number
config	FGPIO pin configuration pointer

10.6.3.2 static void FGPIO_PinWrite (FGPIO_Type * *base*, uint32_t *pin*, uint8_t *output*) [inline], [static]

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
pin	FGPIO pin number
output	FGPIOpin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

10.6.3.3 static void FGPIO_PortSet (FGPIO_Type ∗ *base,* uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

10.6.3.4 static void FGPIO_PortClear (FGPIO_Type ∗ *base,* uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

10.6.3.5 static void FGPIO_PortToggle (FGPIO_Type ∗ *base,* uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

10.6.3.6 static uint32_t FGPIO_PinRead (FGPIO_Type ∗ *base,* uint32_t *pin*) `[inline]`, `[static]`

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
pin	FGPIO pin number

Return values

FGPIO	port input value • 0: corresponding pin input low-logic level. • 1: corresponding pin input high-logic level.

10.6.3.7 uint32_t FGPIO_PortGetInterruptFlags (FGPIO_Type ∗ *base*)

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level-sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)

Return values

The	current FGPIO port interrupt status flags, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

10.6.3.8 void FGPIO_PortClearInterruptFlags (FGPIO_Type ∗ *base,* uint32_t *mask*)

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

Chapter 11
LPI2C: Low Power Inter-Integrated Circuit Driver

11.1 Overview

Modules

- LPI2C CMSIS Driver
- LPI2C FreeRTOS Driver
- LPI2C Master DMA Driver
- LPI2C Master Driver
- LPI2C Slave Driver

Macros

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

Enumerations

- enum {
  kStatus_LPI2C_Busy = MAKE_STATUS(kStatusGroup_LPI2C, 0),
  kStatus_LPI2C_Idle = MAKE_STATUS(kStatusGroup_LPI2C, 1),
  kStatus_LPI2C_Nak = MAKE_STATUS(kStatusGroup_LPI2C, 2),
  kStatus_LPI2C_FifoError = MAKE_STATUS(kStatusGroup_LPI2C, 3),
  kStatus_LPI2C_BitError = MAKE_STATUS(kStatusGroup_LPI2C, 4),
  kStatus_LPI2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_LPI2C, 5),
  kStatus_LPI2C_PinLowTimeout,
  kStatus_LPI2C_NoTransferInProgress,
  kStatus_LPI2C_DmaRequestFail = MAKE_STATUS(kStatusGroup_LPI2C, 8),
  kStatus_LPI2C_Timeout = MAKE_STATUS(kStatusGroup_LPI2C, 9) }
  LPI2C status return codes.

Driver version

- #define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))
  LPI2C driver version.

11.2 Macro Definition Documentation

11.2.1 #define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))

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

11.3 Enumeration Type Documentation

11.3.1 anonymous enum

Enumerator

kStatus_LPI2C_Busy The master is already performing a transfer.
kStatus_LPI2C_Idle The slave driver is idle.
kStatus_LPI2C_Nak The slave device sent a NAK in response to a byte.
kStatus_LPI2C_FifoError FIFO under run or overrun.
kStatus_LPI2C_BitError Transferred bit was not seen on the bus.
kStatus_LPI2C_ArbitrationLost Arbitration lost error.
kStatus_LPI2C_PinLowTimeout SCL or SDA were held low longer than the timeout.
kStatus_LPI2C_NoTransferInProgress Attempt to abort a transfer when one is not in progress.
kStatus_LPI2C_DmaRequestFail DMA request failed.
kStatus_LPI2C_Timeout Timeout polling status flags.

11.4 LPI2C Master Driver

11.4.1 Overview

Data Structures

- struct lpi2c_master_config_t
  Structure with settings to initialize the LPI2C master module. More...
- struct lpi2c_data_match_config_t
  LPI2C master data match configuration structure. More...
- struct lpi2c_master_transfer_t
  Non-blocking transfer descriptor structure. More...
- struct lpi2c_master_handle_t
  Driver handle for master non-blocking APIs. More...

Typedefs

- typedef void(∗ lpi2c_master_transfer_callback_t)(LPI2C_Type ∗base, lpi2c_master_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Master completion callback function pointer type.
- typedef void(∗ lpi2c_master_isr_t)(LPI2C_Type ∗base, void ∗handle)
  Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

Enumerations

- enum _lpi2c_master_flags {
  kLPI2C_MasterTxReadyFlag = LPI2C_MSR_TDF_MASK,
  kLPI2C_MasterRxReadyFlag = LPI2C_MSR_RDF_MASK,
  kLPI2C_MasterEndOfPacketFlag = LPI2C_MSR_EPF_MASK,
  kLPI2C_MasterStopDetectFlag = LPI2C_MSR_SDF_MASK,
  kLPI2C_MasterNackDetectFlag = LPI2C_MSR_NDF_MASK,
  kLPI2C_MasterArbitrationLostFlag = LPI2C_MSR_ALF_MASK,
  kLPI2C_MasterFifoErrFlag = LPI2C_MSR_FEF_MASK,
  kLPI2C_MasterPinLowTimeoutFlag = LPI2C_MSR_PLTF_MASK,
  kLPI2C_MasterDataMatchFlag = LPI2C_MSR_DMF_MASK,
  kLPI2C_MasterBusyFlag = LPI2C_MSR_MBF_MASK,
  kLPI2C_MasterBusBusyFlag = LPI2C_MSR_BBF_MASK,
  kLPI2C_MasterClearFlags,
  kLPI2C_MasterIrqFlags,
  kLPI2C_MasterErrorFlags }
  LPI2C master peripheral flags.
- enum lpi2c_direction_t {
  kLPI2C_Write = 0U,
  kLPI2C_Read = 1U }

Direction of master and slave transfers.
- enum lpi2c_master_pin_config_t {
kLPI2C_2PinOpenDrain = 0x0U,
kLPI2C_2PinOutputOnly = 0x1U,
kLPI2C_2PinPushPull = 0x2U,
kLPI2C_4PinPushPull = 0x3U,
kLPI2C_2PinOpenDrainWithSeparateSlave,
kLPI2C_2PinOutputOnlyWithSeparateSlave,
kLPI2C_2PinPushPullWithSeparateSlave,
kLPI2C_4PinPushPullWithInvertedOutput = 0x7U }
LPI2C pin configuration.
- enum lpi2c_host_request_source_t {
kLPI2C_HostRequestExternalPin = 0x0U,
kLPI2C_HostRequestInputTrigger = 0x1U }
LPI2C master host request selection.
- enum lpi2c_host_request_polarity_t {
kLPI2C_HostRequestPinActiveLow = 0x0U,
kLPI2C_HostRequestPinActiveHigh = 0x1U }
LPI2C master host request pin polarity configuration.
- enum lpi2c_data_match_config_mode_t {
kLPI2C_MatchDisabled = 0x0U,
kLPI2C_1stWordEqualsM0OrM1 = 0x2U,
kLPI2C_AnyWordEqualsM0OrM1 = 0x3U,
kLPI2C_1stWordEqualsM0And2ndWordEqualsM1,
kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1,
kLPI2C_1stWordAndM1EqualsM0AndM1,
kLPI2C_AnyWordAndM1EqualsM0AndM1 }
LPI2C master data match configuration modes.
- enum _lpi2c_master_transfer_flags {
kLPI2C_TransferDefaultFlag = 0x00U,
kLPI2C_TransferNoStartFlag = 0x01U,
kLPI2C_TransferRepeatedStartFlag = 0x02U,
kLPI2C_TransferNoStopFlag = 0x04U }
Transfer option flags.

Initialization and deinitialization

- void LPI2C_MasterGetDefaultConfig (lpi2c_master_config_t ∗masterConfig)
Provides a default configuration for the LPI2C master peripheral.
- void LPI2C_MasterInit (LPI2C_Type ∗base, const lpi2c_master_config_t ∗masterConfig, uint32_t sourceClock_Hz)
Initializes the LPI2C master peripheral.
- void LPI2C_MasterDeinit (LPI2C_Type ∗base)
Deinitializes the LPI2C master peripheral.
- void LPI2C_MasterConfigureDataMatch (LPI2C_Type ∗base, const lpi2c_data_match_config_t ∗matchConfig)

Configures LPI2C master data match feature.
- status_t **LPI2C_MasterCheckAndClearError** (LPI2C_Type ∗base, uint32_t status)
- status_t **LPI2C_CheckForBusyBus** (LPI2C_Type ∗base)
- static void LPI2C_MasterReset (LPI2C_Type ∗base)
  Performs a software reset.
- static void LPI2C_MasterEnable (LPI2C_Type ∗base, bool enable)
  Enables or disables the LPI2C module as master.

Status

- static uint32_t LPI2C_MasterGetStatusFlags (LPI2C_Type ∗base)
  Gets the LPI2C master status flags.
- static void LPI2C_MasterClearStatusFlags (LPI2C_Type ∗base, uint32_t statusMask)
  Clears the LPI2C master status flag state.

Interrupts

- static void LPI2C_MasterEnableInterrupts (LPI2C_Type ∗base, uint32_t interruptMask)
  Enables the LPI2C master interrupt requests.
- static void LPI2C_MasterDisableInterrupts (LPI2C_Type ∗base, uint32_t interruptMask)
  Disables the LPI2C master interrupt requests.
- static uint32_t LPI2C_MasterGetEnabledInterrupts (LPI2C_Type ∗base)
  Returns the set of currently enabled LPI2C master interrupt requests.

DMA control

- static void LPI2C_MasterEnableDMA (LPI2C_Type ∗base, bool enableTx, bool enableRx)
  Enables or disables LPI2C master DMA requests.
- static uint32_t LPI2C_MasterGetTxFifoAddress (LPI2C_Type ∗base)
  Gets LPI2C master transmit data register address for DMA transfer.
- static uint32_t LPI2C_MasterGetRxFifoAddress (LPI2C_Type ∗base)
  Gets LPI2C master receive data register address for DMA transfer.

FIFO control

- static void LPI2C_MasterSetWatermarks (LPI2C_Type ∗base, size_t txWords, size_t rxWords)
  Sets the watermarks for LPI2C master FIFOs.
- static void LPI2C_MasterGetFifoCounts (LPI2C_Type ∗base, size_t ∗rxCount, size_t ∗txCount)
  Gets the current number of words in the LPI2C master FIFOs.

Bus operations

- void LPI2C_MasterSetBaudRate (LPI2C_Type ∗base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.
- static bool LPI2C_MasterGetBusIdleState (LPI2C_Type ∗base)
  Returns whether the bus is idle.
- status_t LPI2C_MasterStart (LPI2C_Type ∗base, uint8_t address, lpi2c_direction_t dir)
  Sends a START signal and slave address on the I2C bus.
- static status_t LPI2C_MasterRepeatedStart (LPI2C_Type ∗base, uint8_t address, lpi2c_direction_t dir)
  Sends a repeated START signal and slave address on the I2C bus.
- status_t LPI2C_MasterSend (LPI2C_Type ∗base, void ∗txBuff, size_t txSize)
  Performs a polling send transfer on the I2C bus.
- status_t LPI2C_MasterReceive (LPI2C_Type ∗base, void ∗rxBuff, size_t rxSize)
  Performs a polling receive transfer on the I2C bus.
- status_t LPI2C_MasterStop (LPI2C_Type ∗base)
  Sends a STOP signal on the I2C bus.
- status_t LPI2C_MasterTransferBlocking (LPI2C_Type ∗base, lpi2c_master_transfer_t ∗transfer)
  Performs a master polling transfer on the I2C bus.

Non-blocking

- void LPI2C_MasterTransferCreateHandle (LPI2C_Type ∗base, lpi2c_master_handle_t ∗handle, lpi2c_master_transfer_callback_t callback, void ∗userData)
  Creates a new handle for the LPI2C master non-blocking APIs.
- status_t LPI2C_MasterTransferNonBlocking (LPI2C_Type ∗base, lpi2c_master_handle_t ∗handle, lpi2c_master_transfer_t ∗transfer)
  Performs a non-blocking transaction on the I2C bus.
- status_t LPI2C_MasterTransferGetCount (LPI2C_Type ∗base, lpi2c_master_handle_t ∗handle, size_t ∗count)
  Returns number of bytes transferred so far.
- void LPI2C_MasterTransferAbort (LPI2C_Type ∗base, lpi2c_master_handle_t ∗handle)
  Terminates a non-blocking LPI2C master transmission early.

IRQ handler

- void LPI2C_MasterTransferHandleIRQ (LPI2C_Type ∗base, void ∗lpi2cMasterHandle)
  Reusable routine to handle master interrupts.

11.4.2 Data Structure Documentation

11.4.2.1 struct lpi2c_master_config_t

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Data Fields

- bool enableMaster
  Whether to enable master mode.
- bool enableDoze
  Whether master is enabled in doze mode.
- bool debugEnable
  Enable transfers to continue when halted in debug mode.
- bool ignoreAck
  Whether to ignore ACK/NACK.
- lpi2c_master_pin_config_t pinConfig
  The pin configuration option.
- uint32_t baudRate_Hz
  Desired baud rate in Hertz.
- uint32_t busIdleTimeout_ns
  Bus idle timeout in nanoseconds.
- uint32_t pinLowTimeout_ns
  Pin low timeout in nanoseconds.
- uint8_t sdaGlitchFilterWidth_ns
  Width in nanoseconds of glitch filter on SDA pin.
- uint8_t sclGlitchFilterWidth_ns
  Width in nanoseconds of glitch filter on SCL pin.
- struct {
    bool enable
      Enable host request.
    lpi2c_host_request_source_t source
      Host request source.
    lpi2c_host_request_polarity_t polarity
      Host request pin polarity.
  } hostRequest

  Host request options.

Field Documentation

(1) bool lpi2c_master_config_t::enableMaster

(2) bool lpi2c_master_config_t::enableDoze

(3) bool lpi2c_master_config_t::debugEnable

(4) bool lpi2c_master_config_t::ignoreAck

(5) lpi2c_master_pin_config_t lpi2c_master_config_t::pinConfig

(6) uint32_t lpi2c_master_config_t::baudRate_Hz

(7) uint32_t lpi2c_master_config_t::busIdleTimeout_ns

Set to 0 to disable.

(8) uint32_t lpi2c_master_config_t::pinLowTimeout_ns

Set to 0 to disable.

(9) uint8_t lpi2c_master_config_t::sdaGlitchFilterWidth_ns

Set to 0 to disable.

(10) uint8_t lpi2c_master_config_t::sclGlitchFilterWidth_ns

Set to 0 to disable.

(11) bool lpi2c_master_config_t::enable

(12) lpi2c_host_request_source_t lpi2c_master_config_t::source

(13) lpi2c_host_request_polarity_t lpi2c_master_config_t::polarity

(14) struct { ... } lpi2c_master_config_t::hostRequest

11.4.2.2 struct lpi2c_data_match_config_t

Data Fields

- lpi2c_data_match_config_mode_t matchMode
  Data match configuration setting.
- bool rxDataMatchOnly
  When set to true, received data is ignored until a successful match.
- uint32_t match0
  Match value 0.
- uint32_t match1
  Match value 1.

Field Documentation

(1) lpi2c_data_match_config_mode_t lpi2c_data_match_config_t::matchMode

(2) bool lpi2c_data_match_config_t::rxDataMatchOnly

(3) uint32_t lpi2c_data_match_config_t::match0

(4) uint32_t lpi2c_data_match_config_t::match1

11.4.2.3 struct _lpi2c_master_transfer

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Data Fields

- uint32_t flags

Bit mask of options for the transfer.
- uint16_t slaveAddress
  The 7-bit slave address.
- lpi2c_direction_t direction
  Either kLPI2C_Read or kLPI2C_Write.
- uint32_t subaddress
  Sub address.
- size_t subaddressSize
  Length of sub address to send in bytes.
- void * data
  Pointer to data to transfer.
- size_t dataSize
  Number of bytes to transfer.

Field Documentation

(1) uint32_t lpi2c_master_transfer_t::flags

See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefault-Flag for normal transfers.

(2) uint16_t lpi2c_master_transfer_t::slaveAddress

(3) lpi2c_direction_t lpi2c_master_transfer_t::direction

(4) uint32_t lpi2c_master_transfer_t::subaddress

Transferred MSB first.

(5) size_t lpi2c_master_transfer_t::subaddressSize

Maximum size is 4 bytes.

(6) void* lpi2c_master_transfer_t::data

(7) size_t lpi2c_master_transfer_t::dataSize

11.4.2.4 struct _lpi2c_master_handle

Note

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

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint16_t remainingBytes
  Remaining byte count in current state.
- uint8_t * buf

Buffer pointer for current state.
- uint16_t commandBuffer [6]
  LPI2C command sequence.
- lpi2c_master_transfer_t transfer
  Copy of the current transfer info.
- lpi2c_master_transfer_callback_t completionCallback
  Callback function pointer.
- void ∗ userData
  Application data passed to callback.

Field Documentation

(1) uint8_t lpi2c_master_handle_t::state

(2) uint16_t lpi2c_master_handle_t::remainingBytes

(3) uint8_t∗ lpi2c_master_handle_t::buf

(4) uint16_t lpi2c_master_handle_t::commandBuffer[6]

When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

(5) lpi2c_master_transfer_t lpi2c_master_handle_t::transfer

(6) lpi2c_master_transfer_callback_t lpi2c_master_handle_t::completionCallback

(7) void∗ lpi2c_master_handle_t::userData

11.4.3 Typedef Documentation

11.4.3.1 typedef void(∗ lpi2c_master_transfer_callback_t)(LPI2C_Type ∗base, lpi2c_master_handle_t ∗handle, status_t completionStatus, void ∗userData)

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Parameters

base	The LPI2C peripheral base address.
completion-Status	Either kStatus_Success or an error code describing how the transfer completed.

userData	Arbitrary pointer-sized value passed from the application.

11.4.4 Enumeration Type Documentation

11.4.4.1 enum _lpi2c_master_flags

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note

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

Enumerator

kLPI2C_MasterTxReadyFlag Transmit data flag.
kLPI2C_MasterRxReadyFlag Receive data flag.
kLPI2C_MasterEndOfPacketFlag End Packet flag.
kLPI2C_MasterStopDetectFlag Stop detect flag.
kLPI2C_MasterNackDetectFlag NACK detect flag.
kLPI2C_MasterArbitrationLostFlag Arbitration lost flag.
kLPI2C_MasterFifoErrFlag FIFO error flag.
kLPI2C_MasterPinLowTimeoutFlag Pin low timeout flag.
kLPI2C_MasterDataMatchFlag Data match flag.
kLPI2C_MasterBusyFlag Master busy flag.
kLPI2C_MasterBusBusyFlag Bus busy flag.
kLPI2C_MasterClearFlags All flags which are cleared by the driver upon starting a transfer.
kLPI2C_MasterIrqFlags IRQ sources enabled by the non-blocking transactional API.
kLPI2C_MasterErrorFlags Errors to check for.

11.4.4.2 enum lpi2c_direction_t

Enumerator

kLPI2C_Write Master transmit.
kLPI2C_Read Master receive.

11.4.4.3 enum lpi2c_master_pin_config_t

Enumerator

kLPI2C_2PinOpenDrain LPI2C Configured for 2-pin open drain mode.
kLPI2C_2PinOutputOnly LPI2C Configured for 2-pin output only mode (ultra-fast mode)
kLPI2C_2PinPushPull LPI2C Configured for 2-pin push-pull mode.
kLPI2C_4PinPushPull LPI2C Configured for 4-pin push-pull mode.
kLPI2C_2PinOpenDrainWithSeparateSlave LPI2C Configured for 2-pin open drain mode with separate LPI2C slave.
kLPI2C_2PinOutputOnlyWithSeparateSlave LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave.
kLPI2C_2PinPushPullWithSeparateSlave LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave.
kLPI2C_4PinPushPullWithInvertedOutput LPI2C Configured for 4-pin push-pull mode(inverted outputs)

11.4.4.4 enum lpi2c_host_request_source_t

Enumerator

kLPI2C_HostRequestExternalPin Select the LPI2C_HREQ pin as the host request input.
kLPI2C_HostRequestInputTrigger Select the input trigger as the host request input.

11.4.4.5 enum lpi2c_host_request_polarity_t

Enumerator

kLPI2C_HostRequestPinActiveLow Configure the LPI2C_HREQ pin active low.
kLPI2C_HostRequestPinActiveHigh Configure the LPI2C_HREQ pin active high.

11.4.4.6 enum lpi2c_data_match_config_mode_t

Enumerator

kLPI2C_MatchDisabled LPI2C Match Disabled.
kLPI2C_1stWordEqualsM0OrM1 LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1.
kLPI2C_AnyWordEqualsM0OrM1 LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1.
kLPI2C_1stWordEqualsM0And2ndWordEqualsM1 LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1.
kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1 LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1.

kLPI2C_1stWordAndM1EqualsM0AndM1 LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1.
kLPI2C_AnyWordAndM1EqualsM0AndM1 LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1.

11.4.4.7 enum _lpi2c_master_transfer_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Enumerator

kLPI2C_TransferDefaultFlag Transfer starts with a start signal, stops with a stop signal.
kLPI2C_TransferNoStartFlag Don't send a start condition, address, and sub address.
kLPI2C_TransferRepeatedStartFlag Send a repeated start condition.
kLPI2C_TransferNoStopFlag Don't send a stop condition.

11.4.5 Function Documentation

11.4.5.1 void LPI2C_MasterGetDefaultConfig (lpi2c_master_config_t * *masterConfig*)

This function provides the following default configuration for the LPI2C master peripheral:

```
*  masterConfig->enableMaster            = true;
*  masterConfig->debugEnable             = false;
*  masterConfig->ignoreAck               = false;
*  masterConfig->pinConfig               = kLPI2C_2PinOpenDrain;
*  masterConfig->baudRate_Hz             = 100000U;
*  masterConfig->busIdleTimeout_ns       = 0;
*  masterConfig->pinLowTimeout_ns        = 0;
*  masterConfig->sdaGlitchFilterWidth_ns = 0;
*  masterConfig->sclGlitchFilterWidth_ns = 0;
*  masterConfig->hostRequest.enable      = false;
*  masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
*  masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

out	*masterConfig*	User provided configuration structure for default values. Refer to lpi2c_master_config_t.

11.4.5.2 void LPI2C_MasterInit (LPI2C_Type * *base*, const lpi2c_master_config_t * *masterConfig*, uint32_t *sourceClock_Hz*)

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

base	The LPI2C peripheral base address.
masterConfig	User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
sourceClock_Hz	Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

11.4.5.3 void LPI2C_MasterDeinit (LPI2C_Type * *base*)

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The LPI2C peripheral base address.

11.4.5.4 void LPI2C_MasterConfigureDataMatch (LPI2C_Type * *base*, const lpi2c_data_match_config_t * *matchConfig*)

Parameters

base	The LPI2C peripheral base address.
matchConfig	Settings for the data match feature.

11.4.5.5 static void LPI2C_MasterReset (LPI2C_Type * *base*) [inline], [static]

Restores the LPI2C master peripheral to reset conditions.

Parameters

base	The LPI2C peripheral base address.

11.4.5.6 static void LPI2C_MasterEnable (LPI2C_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.
enable	Pass true to enable or false to disable the specified LPI2C as master.

11.4.5.7 static uint32_t LPI2C_MasterGetStatusFlags (LPI2C_Type ∗ *base*) [inline], [static]

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The LPI2C peripheral base address.

Returns

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

See Also

_lpi2c_master_flags

11.4.5.8 static void LPI2C_MasterClearStatusFlags (LPI2C_Type ∗ *base*, uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag

- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

Parameters

base	The LPI2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_-master_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags().

See Also

_lpi2c_master_flags.

11.4.5.9 static void LPI2C_MasterEnableInterrupts (LPI2C_Type ∗ *base*, uint32_t *interruptMask*) [inline], [static]

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

base	The LPI2C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

11.4.5.10 static void LPI2C_MasterDisableInterrupts (LPI2C_Type ∗ *base*, uint32_t *interruptMask*) [inline], [static]

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

base	The LPI2C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

11.4.5.11 static uint32_t LPI2C_MasterGetEnabledInterrupts (LPI2C_Type ∗ *base*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

11.4.5.12 static void LPI2C_MasterEnableDMA (LPI2C_Type ∗ *base,* bool *enableTx,* bool *enableRx*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.
enableTx	Enable flag for transmit DMA request. Pass true for enable, false for disable.
enableRx	Enable flag for receive DMA request. Pass true for enable, false for disable.

11.4.5.13 static uint32_t LPI2C_MasterGetTxFifoAddress (LPI2C_Type ∗ *base*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

11.4.5.14 static uint32_t LPI2C_MasterGetRxFifoAddress (LPI2C_Type ∗ *base*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

11.4.5.15 static void LPI2C_MasterSetWatermarks (LPI2C_Type * *base*, size_t *txWords*, size_t *rxWords*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.
txWords	Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
rxWords	Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

11.4.5.16 static void LPI2C_MasterGetFifoCounts (LPI2C_Type * *base*, size_t * *rxCount*, size_t * *txCount*) [inline], [static]

Parameters

	base	The LPI2C peripheral base address.
out	*txCount*	Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
out	*rxCount*	Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

11.4.5.17 void LPI2C_MasterSetBaudRate (LPI2C_Type * *base*, uint32_t *sourceClock_Hz*, uint32_t *baudRate_Hz*)

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note

Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

base	The LPI2C peripheral base address.
sourceClock_-Hz	LPI2C functional clock frequency in Hertz.
baudRate_Hz	Requested bus frequency in Hertz.

11.4.5.18 static bool LPI2C_MasterGetBusIdleState (LPI2C_Type * *base*) [inline], [static]

Requires the master mode to be enabled.

Parameters

base	The LPI2C peripheral base address.

Return values

true	Bus is busy.
false	Bus is idle.

11.4.5.19 status_t LPI2C_MasterStart (LPI2C_Type * *base*, uint8_t *address*, lpi2c_direction_t *dir*)

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

base	The LPI2C peripheral base address.
address	7-bit slave device address, in bits [6:0].
dir	Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success	START signal and address were successfully enqueued in the transmit FIFO.
kStatus_LPI2C_Busy	Another master is currently utilizing the bus.

11.4.5.20 static status_t LPI2C_MasterRepeatedStart (LPI2C_Type * *base*, uint8_t *address*, lpi2c_direction_t *dir*) [inline], [static]

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note

> This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

base	The LPI2C peripheral base address.
address	7-bit slave device address, in bits [6:0].
dir	Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success	Repeated START signal and address were successfully enqueued in the transmit FIFO.
kStatus_LPI2C_Busy	Another master is currently utilizing the bus.

11.4.5.21 status_t LPI2C_MasterSend (LPI2C_Type * *base*, void * *txBuff*, size_t *txSize*)

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_LPI2C_Nak.

Parameters

base	The LPI2C peripheral base address.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.

Return values

kStatus_Success	Data was sent successfully.
kStatus_LPI2C_Busy	Another master is currently utilizing the bus.
kStatus_LPI2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_LPI2C_FifoError	FIFO under run or over run.
kStatus_LPI2C_-ArbitrationLost	Arbitration lost error.
kStatus_LPI2C_PinLow-Timeout	SCL or SDA were held low longer than the timeout.

11.4.5.22 status_t LPI2C_MasterReceive (LPI2C_Type ∗ *base*, void ∗ *rxBuff*, size_t *rxSize*)

Parameters

base	The LPI2C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.

Return values

kStatus_Success	Data was received successfully.
kStatus_LPI2C_Busy	Another master is currently utilizing the bus.
kStatus_LPI2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_LPI2C_FifoError	FIFO under run or overrun.
kStatus_LPI2C_-ArbitrationLost	Arbitration lost error.
kStatus_LPI2C_PinLow-Timeout	SCL or SDA were held low longer than the timeout.

11.4.5.23 status_t LPI2C_MasterStop (LPI2C_Type ∗ *base*)

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

base	The LPI2C peripheral base address.

Return values

kStatus_Success	The STOP signal was successfully sent on the bus and the transaction terminated.
kStatus_LPI2C_Busy	Another master is currently utilizing the bus.
kStatus_LPI2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_LPI2C_FifoError	FIFO under run or overrun.
kStatus_LPI2C_-ArbitrationLost	Arbitration lost error.
kStatus_LPI2C_PinLow-Timeout	SCL or SDA were held low longer than the timeout.

11.4.5.24 status_t LPI2C_MasterTransferBlocking (LPI2C_Type ∗ *base*, lpi2c_master_transfer_t ∗ *transfer*)

Note

The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

base	The LPI2C peripheral base address.
transfer	Pointer to the transfer structure.

Return values

kStatus_Success	Data was received successfully.
kStatus_LPI2C_Busy	Another master is currently utilizing the bus.
kStatus_LPI2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_LPI2C_FifoError	FIFO under run or overrun.
kStatus_LPI2C_-ArbitrationLost	Arbitration lost error.
kStatus_LPI2C_PinLow-Timeout	SCL or SDA were held low longer than the timeout.

11.4.5.25 void LPI2C_MasterTransferCreateHandle (LPI2C_Type * *base*, lpi2c_master_handle_t * *handle*, lpi2c_master_transfer_callback_t *callback*, void * *userData*)

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note

> The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

	base	The LPI2C peripheral base address.
out	*handle*	Pointer to the LPI2C master driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

11.4.5.26 status_t LPI2C_MasterTransferNonBlocking (LPI2C_Type * *base*, lpi2c_master_handle_t * *handle*, lpi2c_master_transfer_t * *transfer*)

Parameters

base	The LPI2C peripheral base address.
handle	Pointer to the LPI2C master driver handle.
transfer	The pointer to the transfer descriptor.

Return values

kStatus_Success	The transaction was started successfully.
kStatus_LPI2C_Busy	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

11.4.5.27 status_t LPI2C_MasterTransferGetCount (LPI2C_Type * *base*, lpi2c_master_handle_t * *handle*, size_t * *count*)

Parameters

	base	The LPI2C peripheral base address.
	handle	Pointer to the LPI2C master driver handle.
out	*count*	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

11.4.5.28 void LPI2C_MasterTransferAbort (LPI2C_Type * *base*, lpi2c_master_handle_t * *handle*)

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

base	The LPI2C peripheral base address.
handle	Pointer to the LPI2C master driver handle.

Return values

kStatus_Success	A transaction was successfully aborted.
kStatus_LPI2C_Idle	There is not a non-blocking transaction currently in progress.

11.4.5.29 void LPI2C_MasterTransferHandleIRQ (LPI2C_Type * *base*, void * *lpi2cMasterHandle*)

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

base	The LPI2C peripheral base address.
lpi2cMaster-Handle	Pointer to the LPI2C master driver handle.

11.5 LPI2C Slave Driver

11.5.1 Overview

Data Structures

- struct lpi2c_slave_config_t
  Structure with settings to initialize the LPI2C slave module. More...
- struct lpi2c_slave_transfer_t
  LPI2C slave transfer structure. More...
- struct lpi2c_slave_handle_t
  LPI2C slave handle structure. More...

Typedefs

- typedef void(∗ lpi2c_slave_transfer_callback_t)(LPI2C_Type ∗base, lpi2c_slave_transfer_t ∗transfer, void ∗userData)
  Slave event callback function pointer type.

Enumerations

- enum _lpi2c_slave_flags {
  kLPI2C_SlaveTxReadyFlag = LPI2C_SSR_TDF_MASK,
  kLPI2C_SlaveRxReadyFlag = LPI2C_SSR_RDF_MASK,
  kLPI2C_SlaveAddressValidFlag = LPI2C_SSR_AVF_MASK,
  kLPI2C_SlaveTransmitAckFlag = LPI2C_SSR_TAF_MASK,
  kLPI2C_SlaveRepeatedStartDetectFlag = LPI2C_SSR_RSF_MASK,
  kLPI2C_SlaveStopDetectFlag = LPI2C_SSR_SDF_MASK,
  kLPI2C_SlaveBitErrFlag = LPI2C_SSR_BEF_MASK,
  kLPI2C_SlaveFifoErrFlag = LPI2C_SSR_FEF_MASK,
  kLPI2C_SlaveAddressMatch0Flag = LPI2C_SSR_AM0F_MASK,
  kLPI2C_SlaveAddressMatch1Flag = LPI2C_SSR_AM1F_MASK,
  kLPI2C_SlaveGeneralCallFlag = LPI2C_SSR_GCF_MASK,
  kLPI2C_SlaveBusyFlag = LPI2C_SSR_SBF_MASK,
  kLPI2C_SlaveBusBusyFlag = LPI2C_SSR_BBF_MASK,
  kLPI2C_SlaveClearFlags,
  kLPI2C_SlaveIrqFlags,
  kLPI2C_SlaveErrorFlags = kLPI2C_SlaveFifoErrFlag | kLPI2C_SlaveBitErrFlag }
  LPI2C slave peripheral flags.
- enum lpi2c_slave_address_match_t {
  kLPI2C_MatchAddress0 = 0U,
  kLPI2C_MatchAddress0OrAddress1 = 2U,
  kLPI2C_MatchAddress0ThroughAddress1 = 6U }
  LPI2C slave address match options.

- enum lpi2c_slave_transfer_event_t {
kLPI2C_SlaveAddressMatchEvent = 0x01U,
kLPI2C_SlaveTransmitEvent = 0x02U,
kLPI2C_SlaveReceiveEvent = 0x04U,
kLPI2C_SlaveTransmitAckEvent = 0x08U,
kLPI2C_SlaveRepeatedStartEvent = 0x10U,
kLPI2C_SlaveCompletionEvent = 0x20U,
kLPI2C_SlaveAllEvents }
Set of events sent to the callback for non blocking slave transfers.

Slave initialization and deinitialization

- void LPI2C_SlaveGetDefaultConfig (lpi2c_slave_config_t *slaveConfig)
Provides a default configuration for the LPI2C slave peripheral.
- void LPI2C_SlaveInit (LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t sourceClock_Hz)
Initializes the LPI2C slave peripheral.
- void LPI2C_SlaveDeinit (LPI2C_Type *base)
Deinitializes the LPI2C slave peripheral.
- static void LPI2C_SlaveReset (LPI2C_Type *base)
Performs a software reset of the LPI2C slave peripheral.
- static void LPI2C_SlaveEnable (LPI2C_Type *base, bool enable)
Enables or disables the LPI2C module as slave.

Slave status

- static uint32_t LPI2C_SlaveGetStatusFlags (LPI2C_Type *base)
Gets the LPI2C slave status flags.
- static void LPI2C_SlaveClearStatusFlags (LPI2C_Type *base, uint32_t statusMask)
Clears the LPI2C status flag state.

Slave interrupts

- static void LPI2C_SlaveEnableInterrupts (LPI2C_Type *base, uint32_t interruptMask)
Enables the LPI2C slave interrupt requests.
- static void LPI2C_SlaveDisableInterrupts (LPI2C_Type *base, uint32_t interruptMask)
Disables the LPI2C slave interrupt requests.
- static uint32_t LPI2C_SlaveGetEnabledInterrupts (LPI2C_Type *base)
Returns the set of currently enabled LPI2C slave interrupt requests.

Slave DMA control

- static void LPI2C_SlaveEnableDMA (LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Slave bus operations

- static bool LPI2C_SlaveGetBusIdleState (LPI2C_Type *base)
  Returns whether the bus is idle.
- static void LPI2C_SlaveTransmitAck (LPI2C_Type *base, bool ackOrNack)
  Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.
- static uint32_t LPI2C_SlaveGetReceivedAddress (LPI2C_Type *base)
  Returns the slave address sent by the I2C master.
- status_t LPI2C_SlaveSend (LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)
  Performs a polling send transfer on the I2C bus.
- status_t LPI2C_SlaveReceive (LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)
  Performs a polling receive transfer on the I2C bus.

Slave non-blocking

- void LPI2C_SlaveTransferCreateHandle (LPI2C_Type *base, lpi2c_slave_handle_t *handle, lpi2c_slave_transfer_callback_t callback, void *userData)
  Creates a new handle for the LPI2C slave non-blocking APIs.
- status_t LPI2C_SlaveTransferNonBlocking (LPI2C_Type *base, lpi2c_slave_handle_t *handle, uint32_t eventMask)
  Starts accepting slave transfers.
- status_t LPI2C_SlaveTransferGetCount (LPI2C_Type *base, lpi2c_slave_handle_t *handle, size_t *count)
  Gets the slave transfer status during a non-blocking transfer.
- void LPI2C_SlaveTransferAbort (LPI2C_Type *base, lpi2c_slave_handle_t *handle)
  Aborts the slave non-blocking transfers.

Slave IRQ handler

- void LPI2C_SlaveTransferHandleIRQ (LPI2C_Type *base, lpi2c_slave_handle_t *handle)
  Reusable routine to handle slave interrupts.

11.5.2 Data Structure Documentation

11.5.2.1 struct lpi2c_slave_config_t

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Data Fields

- bool enableSlave
  Enable slave mode.
- uint8_t address0
  Slave's 7-bit address.
- uint8_t address1
  Alternate slave 7-bit address.
- lpi2c_slave_address_match_t addressMatchMode
  Address matching options.
- bool filterDozeEnable
  Enable digital glitch filter in doze mode.
- bool filterEnable
  Enable digital glitch filter.
- bool enableGeneralCall
  Enable general call address matching.
- bool ignoreAck
  Continue transfers after a NACK is detected.
- bool enableReceivedAddressRead
  Enable reading the address received address as the first byte of data.
- uint32_t sdaGlitchFilterWidth_ns
  Width in nanoseconds of the digital filter on the SDA signal.
- uint32_t sclGlitchFilterWidth_ns
  Width in nanoseconds of the digital filter on the SCL signal.
- uint32_t dataValidDelay_ns
  Width in nanoseconds of the data valid delay.
- uint32_t clockHoldTime_ns
  Width in nanoseconds of the clock hold time.
- bool enableAck
  Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted.
- bool enableTx
  Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.
- bool enableRx
  Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.
- bool enableAddress
  Enables SCL clock stretching when the address valid flag is asserted.

Field Documentation

(1) bool lpi2c_slave_config_t::enableSlave

(2) uint8_t lpi2c_slave_config_t::address0

(3) uint8_t lpi2c_slave_config_t::address1

(4) lpi2c_slave_address_match_t lpi2c_slave_config_t::addressMatchMode

(5) bool lpi2c_slave_config_t::filterDozeEnable

(6) bool lpi2c_slave_config_t::filterEnable

(7) bool lpi2c_slave_config_t::enableGeneralCall

(8) bool lpi2c_slave_config_t::enableAck

Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

(9) bool lpi2c_slave_config_t::enableTx

(10) bool lpi2c_slave_config_t::enableRx

(11) bool lpi2c_slave_config_t::enableAddress

(12) bool lpi2c_slave_config_t::ignoreAck

(13) bool lpi2c_slave_config_t::enableReceivedAddressRead

(14) uint32_t lpi2c_slave_config_t::sdaGlitchFilterWidth_ns

Set to 0 to disable.

(15) uint32_t lpi2c_slave_config_t::sclGlitchFilterWidth_ns

Set to 0 to disable.

(16) uint32_t lpi2c_slave_config_t::dataValidDelay_ns

(17) uint32_t lpi2c_slave_config_t::clockHoldTime_ns

11.5.2.2 struct lpi2c_slave_transfer_t

Data Fields

- lpi2c_slave_transfer_event_t event
  Reason the callback is being invoked.
- uint8_t receivedAddress
  Matching address send by master.
- uint8_t * data
  Transfer buffer.
- size_t dataSize
  Transfer size.
- status_t completionStatus
  Success or error code describing how the transfer completed.
- size_t transferredCount
  Number of bytes actually transferred since start or last repeated start.

Field Documentation

(1) lpi2c_slave_transfer_event_t lpi2c_slave_transfer_t::event

(2) uint8_t lpi2c_slave_transfer_t::receivedAddress

(3) status_t lpi2c_slave_transfer_t::completionStatus

Only applies for kLPI2C_SlaveCompletionEvent.

(4) size_t lpi2c_slave_transfer_t::transferredCount

11.5.2.3 struct _lpi2c_slave_handle

Note

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

Data Fields

- lpi2c_slave_transfer_t transfer
  LPI2C slave transfer copy.
- bool isBusy
  Whether transfer is busy.
- bool wasTransmit
  Whether the last transfer was a transmit.
- uint32_t eventMask
  Mask of enabled events.
- uint32_t transferredCount
  Count of bytes transferred.
- lpi2c_slave_transfer_callback_t callback
  Callback function called at transfer event.
- void ∗ userData
  Callback parameter passed to callback.

Field Documentation

(1) lpi2c_slave_transfer_t lpi2c_slave_handle_t::transfer

(2) bool lpi2c_slave_handle_t::isBusy

(3) bool lpi2c_slave_handle_t::wasTransmit

(4) uint32_t lpi2c_slave_handle_t::eventMask

(5) uint32_t lpi2c_slave_handle_t::transferredCount

(6) lpi2c_slave_transfer_callback_t lpi2c_slave_handle_t::callback

(7) void∗ lpi2c_slave_handle_t::userData

11.5.3 Typedef Documentation

11.5.3.1 typedef void(∗ lpi2c_slave_transfer_callback_t)(LPI2C_Type ∗base, lpi2c_slave_transfer_t ∗transfer, void ∗userData)

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_-SlaveSetCallback() function after you have created a handle.

Parameters

base	Base address for the LPI2C instance on which the event occurred.
transfer	Pointer to transfer descriptor containing values passed to and/or from the callback.
userData	Arbitrary pointer-sized value passed from the application.

11.5.4 Enumeration Type Documentation

11.5.4.1 enum _lpi2c_slave_flags

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Note

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

Enumerator

kLPI2C_SlaveTxReadyFlag Transmit data flag.
kLPI2C_SlaveRxReadyFlag Receive data flag.
kLPI2C_SlaveAddressValidFlag Address valid flag.
kLPI2C_SlaveTransmitAckFlag Transmit ACK flag.
kLPI2C_SlaveRepeatedStartDetectFlag Repeated start detect flag.
kLPI2C_SlaveStopDetectFlag Stop detect flag.
kLPI2C_SlaveBitErrFlag Bit error flag.
kLPI2C_SlaveFifoErrFlag FIFO error flag.
kLPI2C_SlaveAddressMatch0Flag Address match 0 flag.
kLPI2C_SlaveAddressMatch1Flag Address match 1 flag.
kLPI2C_SlaveGeneralCallFlag General call flag.
kLPI2C_SlaveBusyFlag Master busy flag.
kLPI2C_SlaveBusBusyFlag Bus busy flag.
kLPI2C_SlaveClearFlags All flags which are cleared by the driver upon starting a transfer.
kLPI2C_SlaveIrqFlags IRQ sources enabled by the non-blocking transactional API.
kLPI2C_SlaveErrorFlags Errors to check for.

11.5.4.2 enum lpi2c_slave_address_match_t

Enumerator

kLPI2C_MatchAddress0 Match only address 0.
kLPI2C_MatchAddress0OrAddress1 Match either address 0 or address 1.
kLPI2C_MatchAddress0ThroughAddress1 Match a range of slave addresses from address 0 through address 1.

11.5.4.3 enum lpi2c_slave_transfer_event_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

kLPI2C_SlaveAddressMatchEvent Received the slave address after a start or repeated start.
kLPI2C_SlaveTransmitEvent Callback is requested to provide data to transmit (slave-transmitter role).
kLPI2C_SlaveReceiveEvent Callback is requested to provide a buffer in which to place received data (slave-receiver role).
kLPI2C_SlaveTransmitAckEvent Callback needs to either transmit an ACK or NACK.
kLPI2C_SlaveRepeatedStartEvent A repeated start was detected.
kLPI2C_SlaveCompletionEvent A stop was detected, completing the transfer.
kLPI2C_SlaveAllEvents Bit mask of all available events.

11.5.5 Function Documentation

11.5.5.1 void LPI2C_SlaveGetDefaultConfig (lpi2c_slave_config_t * *slaveConfig*)

This function provides the following default configuration for the LPI2C slave peripheral:

```
*  slaveConfig->enableSlave               = true;
*  slaveConfig->address0                  = 0U;
*  slaveConfig->address1                  = 0U;
*  slaveConfig->addressMatchMode          = kLPI2C_MatchAddress0;
*  slaveConfig->filterDozeEnable          = true;
*  slaveConfig->filterEnable              = true;
*  slaveConfig->enableGeneralCall         = false;
*  slaveConfig->sclStall.enableAck        = false;
*  slaveConfig->sclStall.enableTx         = true;
*  slaveConfig->sclStall.enableRx         = true;
*  slaveConfig->sclStall.enableAddress    = true;
```

```
*   slaveConfig->ignoreAck                     = false;
*   slaveConfig->enableReceivedAddressRead = false;
*   slaveConfig->sdaGlitchFilterWidth_ns    = 0;
*   slaveConfig->sclGlitchFilterWidth_ns    = 0;
*   slaveConfig->dataValidDelay_ns          = 0;
*   slaveConfig->clockHoldTime_ns           = 0;
*
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

out	*slaveConfig*	User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t.

11.5.5.2 void LPI2C_SlaveInit (LPI2C_Type ∗ *base*, const lpi2c_slave_config_t ∗ *slaveConfig*, uint32_t *sourceClock_Hz*)

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

base	The LPI2C peripheral base address.
slaveConfig	User provided peripheral configuration. Use LPI2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
sourceClock_-Hz	Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

11.5.5.3 void LPI2C_SlaveDeinit (LPI2C_Type ∗ *base*)

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The LPI2C peripheral base address.

11.5.5.4 static void LPI2C_SlaveReset (LPI2C_Type ∗ *base*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.

11.5.5.5 static void LPI2C_SlaveEnable (LPI2C_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.
enable	Pass true to enable or false to disable the specified LPI2C as slave.

11.5.5.6 static uint32_t LPI2C_SlaveGetStatusFlags (LPI2C_Type ∗ *base*) [inline], [static]

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The LPI2C peripheral base address.

Returns

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

See Also

_lpi2c_slave_flags

11.5.5.7 static void LPI2C_SlaveClearStatusFlags (LPI2C_Type ∗ *base*, uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

Parameters

base	The LPI2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_-slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags().

See Also

_lpi2c_slave_flags.

11.5.5.8 static void LPI2C_SlaveEnableInterrupts (LPI2C_Type * *base,* uint32_t *interruptMask*) [inline], [static]

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

base	The LPI2C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

11.5.5.9 static void LPI2C_SlaveDisableInterrupts (LPI2C_Type * *base,* uint32_t *interruptMask*) [inline], [static]

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

base	The LPI2C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

11.5.5.10 static uint32_t LPI2C_SlaveGetEnabledInterrupts (LPI2C_Type * *base*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

11.5.5.11 static void LPI2C_SlaveEnableDMA (LPI2C_Type * *base*, bool *enableAddressValid*, bool *enableRx*, bool *enableTx*) [inline], [static]

Parameters

base	The LPI2C peripheral base address.
enableAddress-Valid	Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
enableRx	Enable flag for the receive data DMA request. Pass true for enable, false for disable.
enableTx	Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

11.5.5.12 static bool LPI2C_SlaveGetBusIdleState (LPI2C_Type * *base*) [inline], [static]

Requires the slave mode to be enabled.

Parameters

base	The LPI2C peripheral base address.

Return values

true	Bus is busy.
false	Bus is idle.

11.5.5.13 static void LPI2C_SlaveTransmitAck (LPI2C_Type * *base*, bool *ackOrNack*) [inline], [static]

Use this function to send an ACK or NAK when the kLPI2C_SlaveTransmitAckFlag is asserted. This only happens if you enable the sclStall.enableAck field of the lpi2c_slave_config_t configuration structure used to initialize the slave peripheral.

Parameters

base	The LPI2C peripheral base address.
ackOrNack	Pass true for an ACK or false for a NAK.

11.5.5.14 static uint32_t LPI2C_SlaveGetReceivedAddress (LPI2C_Type * *base*) [inline], [static]

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

base	The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

11.5.5.15 status_t LPI2C_SlaveSend (LPI2C_Type * *base,* void * *txBuff,* size_t *txSize,* size_t * *actualTxSize*)

Parameters

	base	The LPI2C peripheral base address.
	txBuff	The pointer to the data to be transferred.
	txSize	The length in bytes of the data to be transferred.
out	*actualTxSize*	

Returns

Error or success status returned by API.

11.5.5.16 status_t LPI2C_SlaveReceive (LPI2C_Type * *base,* void * *rxBuff,* size_t *rxSize,* size_t * *actualRxSize*)

Parameters

	base	The LPI2C peripheral base address.
	rxBuff	The pointer to the data to be transferred.
	rxSize	The length in bytes of the data to be transferred.
out	*actualRxSize*	

Returns

Error or success status returned by API.

11.5.5.17 void LPI2C_SlaveTransferCreateHandle (LPI2C_Type * *base*, lpi2c_slave_handle_t * *handle*, lpi2c_slave_transfer_callback_t *callback*, void * *userData*)

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

	base	The LPI2C peripheral base address.
out	*handle*	Pointer to the LPI2C slave driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

11.5.5.18 status_t LPI2C_SlaveTransferNonBlocking (LPI2C_Type * *base*, lpi2c_slave_handle_t * *handle*, uint32_t *eventMask*)

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of lpi2c_slave_transfer_event_t enumerators for the events you wish to receive. The kLPI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kLPI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The LPI2C peripheral base address.
handle	Pointer to lpi2c_slave_handle_t structure which stores the transfer state.
eventMask	Bit mask formed by OR'ing together lpi2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kLPI2C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_LPI2C_Busy	Slave transfers have already been started on this handle.

11.5.5.19 status_t LPI2C_SlaveTransferGetCount (LPI2C_Type ∗ *base*, lpi2c_slave_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

	base	The LPI2C peripheral base address.
	handle	Pointer to i2c_slave_handle_t structure.
out	*count*	Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

kStatus_Success	
kStatus_NoTransferInProgress	

11.5.5.20 void LPI2C_SlaveTransferAbort (LPI2C_Type ∗ *base*, lpi2c_slave_handle_t ∗ *handle*)

Note

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

Parameters

base	The LPI2C peripheral base address.
handle	Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

Return values

kStatus_Success	
kStatus_LPI2C_Idle	

11.5.5.21 void LPI2C_SlaveTransferHandleIRQ (LPI2C_Type ∗ *base*, lpi2c_slave_handle_t ∗ *handle*)

Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

base	The LPI2C peripheral base address.
handle	Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

11.6 LPI2C Master DMA Driver

11.7 LPI2C FreeRTOS Driver

11.7.1 Overview

Driver version

- #define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  LPI2C FreeRTOS driver version.

LPI2C RTOS Operation

- status_t LPI2C_RTOS_Init (lpi2c_rtos_handle_t ∗handle, LPI2C_Type ∗base, const lpi2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes LPI2C.
- status_t LPI2C_RTOS_Deinit (lpi2c_rtos_handle_t ∗handle)
  Deinitializes the LPI2C.
- status_t LPI2C_RTOS_Transfer (lpi2c_rtos_handle_t ∗handle, lpi2c_master_transfer_t ∗transfer)
  Performs I2C transfer.

11.7.2 Macro Definition Documentation

11.7.2.1 #define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

11.7.3 Function Documentation

11.7.3.1 status_t LPI2C_RTOS_Init (lpi2c_rtos_handle_t ∗ *handle*, LPI2C_Type ∗ *base*, const lpi2c_master_config_t ∗ *masterConfig*, uint32_t *srcClock_Hz*)

This function initializes the LPI2C module and related RTOS context.

Parameters

handle	The RTOS LPI2C handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the LPI2C instance to initialize.
masterConfig	Configuration structure to set-up LPI2C in master mode.
srcClock_Hz	Frequency of input clock of the LPI2C module.

Returns

status of the operation.

11.7.3.2 status_t LPI2C_RTOS_Deinit (lpi2c_rtos_handle_t * *handle*)

This function deinitializes the LPI2C module and related RTOS context.

Parameters

handle	The RTOS LPI2C handle.

11.7.3.3 status_t LPI2C_RTOS_Transfer (lpi2c_rtos_handle_t ∗ *handle*, lpi2c_master_transfer_t ∗ *transfer*)

This function performs an I2C transfer using LPI2C module according to data given in the transfer structure.

Parameters

handle	The RTOS LPI2C handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

11.8 LPI2C CMSIS Driver

Chapter 12
LPIT: Low-Power Interrupt Timer

12.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Interrupt Timer (LPIT) of MCUXpresso SDK devices.

12.2 Function groups

The LPIT driver supports operating the module as a time counter.

12.2.1 Initialization and deinitialization

The function LPIT_Init() initializes the LPIT with specified configurations. The function LPIT_GetDefaultConfig() gets the default configurations. The initialization function configures the LPIT operation in doze mode and debug mode.

The function LPIT_SetupChannel() configures the operation of each LPIT channel.

The function LPIT_Deinit() disables the LPIT module and disables the module clock.

12.2.2 Timer period Operations

The function LPITR_SetTimerPeriod() sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function LPIT_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. User can call the utility macros provided in fsl_common.h to convert to microseconds or milliseconds

12.2.3 Start and Stop timer operations

The function LPIT_StartTimer() starts the timer counting. After calling this function, the timer loads the period value set earlier via the LPIT_SetPeriod() function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function LPIT_StopTimer() stops the timer counting.

12.2.4 Status

Provides functions to get and clear the LPIT status.

12.2.5 Interrupt

Provides functions to enable/disable LPIT interrupts and get current enabled interrupts.

12.3 Typical use case

12.3.1 LPIT tick example

Updates the LPIT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lpit

Data Structures

- struct lpit_chnl_params_t
  Structure to configure the channel timer. More...
- struct lpit_config_t
  LPIT configuration structure. More...

Functions

- static void LPIT_Reset (LPIT_Type ∗base)
  Performs a software reset on the LPIT module.

Driver version

- enum lpit_chnl_t {
  kLPIT_Chnl_0 = 0U,
  kLPIT_Chnl_1,
  kLPIT_Chnl_2,
  kLPIT_Chnl_3 }
  List of LPIT channels.
- enum lpit_timer_modes_t {
  kLPIT_PeriodicCounter = 0U,
  kLPIT_DualPeriodicCounter,
  kLPIT_TriggerAccumulator,
  kLPIT_InputCapture }
  Mode options available for the LPIT timer.
- enum lpit_trigger_select_t {

kLPIT_Trigger_TimerChn0 = 0U,
kLPIT_Trigger_TimerChn1,
kLPIT_Trigger_TimerChn2,
kLPIT_Trigger_TimerChn3,
kLPIT_Trigger_TimerChn4,
kLPIT_Trigger_TimerChn5,
kLPIT_Trigger_TimerChn6,
kLPIT_Trigger_TimerChn7,
kLPIT_Trigger_TimerChn8,
kLPIT_Trigger_TimerChn9,
kLPIT_Trigger_TimerChn10,
kLPIT_Trigger_TimerChn11,
kLPIT_Trigger_TimerChn12,
kLPIT_Trigger_TimerChn13,
kLPIT_Trigger_TimerChn14,
kLPIT_Trigger_TimerChn15 }
Trigger options available.
- enum lpit_trigger_source_t {
kLPIT_TriggerSource_External = 0U,
kLPIT_TriggerSource_Internal }
Trigger source options available.
- enum lpit_interrupt_enable_t {
kLPIT_Channel0TimerInterruptEnable = (1U << 0),
kLPIT_Channel1TimerInterruptEnable = (1U << 1),
kLPIT_Channel2TimerInterruptEnable = (1U << 2),
kLPIT_Channel3TimerInterruptEnable = (1U << 3) }
List of LPIT interrupts.
- enum lpit_status_flags_t {
kLPIT_Channel0TimerFlag = (1U << 0),
kLPIT_Channel1TimerFlag = (1U << 1),
kLPIT_Channel2TimerFlag = (1U << 2),
kLPIT_Channel3TimerFlag = (1U << 3) }
List of LPIT status flags.
- #define FSL_LPIT_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
Version 2.0.2.

Initialization and deinitialization

- void LPIT_Init (LPIT_Type ∗base, const lpit_config_t ∗config)
Ungates the LPIT clock and configures the peripheral for a basic operation.
- void LPIT_Deinit (LPIT_Type ∗base)
Disables the module and gates the LPIT clock.
- void LPIT_GetDefaultConfig (lpit_config_t ∗config)
Fills in the LPIT configuration structure with default settings.
- status_t LPIT_SetupChannel (LPIT_Type ∗base, lpit_chnl_t channel, const lpit_chnl_params_t ∗chnlSetup)
Sets up an LPIT channel based on the user's preference.

Interrupt Interface

- static void LPIT_EnableInterrupts (LPIT_Type ∗base, uint32_t mask)
  Enables the selected PIT interrupts.
- static void LPIT_DisableInterrupts (LPIT_Type ∗base, uint32_t mask)
  Disables the selected PIT interrupts.
- static uint32_t LPIT_GetEnabledInterrupts (LPIT_Type ∗base)
  Gets the enabled LPIT interrupts.

Status Interface

- static uint32_t LPIT_GetStatusFlags (LPIT_Type ∗base)
  Gets the LPIT status flags.
- static void LPIT_ClearStatusFlags (LPIT_Type ∗base, uint32_t mask)
  Clears the LPIT status flags.

Read and Write the timer period

- static void LPIT_SetTimerPeriod (LPIT_Type ∗base, lpit_chnl_t channel, uint32_t ticks)
  Sets the timer period in units of count.
- static uint32_t LPIT_GetCurrentTimerCount (LPIT_Type ∗base, lpit_chnl_t channel)
  Reads the current timer counting value.

Timer Start and Stop

- static void LPIT_StartTimer (LPIT_Type ∗base, lpit_chnl_t channel)
  Starts the timer counting.
- static void LPIT_StopTimer (LPIT_Type ∗base, lpit_chnl_t channel)
  Stops the timer counting.

12.4 Data Structure Documentation

12.4.1 struct lpit_chnl_params_t

Data Fields

- bool chainChannel
  true: Timer chained to previous timer; false: Timer not chained
- lpit_timer_modes_t timerMode
  Timers mode of operation.
- lpit_trigger_select_t triggerSelect
  Trigger selection for the timer.
- lpit_trigger_source_t triggerSource
  Decides if we use external or internal trigger.
- bool enableReloadOnTrigger
  true: Timer reloads when a trigger is detected; false: No effect
- bool enableStopOnTimeout
  true: Timer will stop after timeout; false: does not stop after timeout
- bool enableStartOnTrigger
  true: Timer starts when a trigger is detected; false: decrement immediately

Field Documentation

(1) lpit_timer_modes_t lpit_chnl_params_t::timerMode

(2) lpit_trigger_source_t lpit_chnl_params_t::triggerSource

12.4.2 struct lpit_config_t

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

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

Data Fields

- bool enableRunInDebug
  true: Timers run in debug mode; false: Timers stop in debug mode
- bool enableRunInDoze
  true: Timers run in doze mode; false: Timers stop in doze mode

12.5 Enumeration Type Documentation

12.5.1 enum lpit_chnl_t

Note

Actual number of available channels is SoC-dependent

Enumerator

kLPIT_Chnl_0 LPIT channel number 0.
kLPIT_Chnl_1 LPIT channel number 1.
kLPIT_Chnl_2 LPIT channel number 2.
kLPIT_Chnl_3 LPIT channel number 3.

12.5.2 enum lpit_timer_modes_t

Enumerator

kLPIT_PeriodicCounter Use the all 32-bits, counter loads and decrements to zero.
kLPIT_DualPeriodicCounter Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement.
kLPIT_TriggerAccumulator Counter loads on first trigger and decrements on each trigger.
kLPIT_InputCapture Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

12.5.3 enum lpit_trigger_select_t

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Enumerator

kLPIT_Trigger_TimerChn0 Channel 0 is selected as a trigger source.
kLPIT_Trigger_TimerChn1 Channel 1 is selected as a trigger source.
kLPIT_Trigger_TimerChn2 Channel 2 is selected as a trigger source.
kLPIT_Trigger_TimerChn3 Channel 3 is selected as a trigger source.
kLPIT_Trigger_TimerChn4 Channel 4 is selected as a trigger source.
kLPIT_Trigger_TimerChn5 Channel 5 is selected as a trigger source.
kLPIT_Trigger_TimerChn6 Channel 6 is selected as a trigger source.
kLPIT_Trigger_TimerChn7 Channel 7 is selected as a trigger source.
kLPIT_Trigger_TimerChn8 Channel 8 is selected as a trigger source.
kLPIT_Trigger_TimerChn9 Channel 9 is selected as a trigger source.
kLPIT_Trigger_TimerChn10 Channel 10 is selected as a trigger source.
kLPIT_Trigger_TimerChn11 Channel 11 is selected as a trigger source.
kLPIT_Trigger_TimerChn12 Channel 12 is selected as a trigger source.
kLPIT_Trigger_TimerChn13 Channel 13 is selected as a trigger source.
kLPIT_Trigger_TimerChn14 Channel 14 is selected as a trigger source.
kLPIT_Trigger_TimerChn15 Channel 15 is selected as a trigger source.

12.5.4 enum lpit_trigger_source_t

Enumerator

kLPIT_TriggerSource_External Use external trigger input.
kLPIT_TriggerSource_Internal Use internal trigger.

12.5.5 enum lpit_interrupt_enable_t

Note

Number of timer channels are SoC-specific. See the SoC Reference Manual.

Enumerator

kLPIT_Channel0TimerInterruptEnable Channel 0 Timer interrupt.
kLPIT_Channel1TimerInterruptEnable Channel 1 Timer interrupt.
kLPIT_Channel2TimerInterruptEnable Channel 2 Timer interrupt.
kLPIT_Channel3TimerInterruptEnable Channel 3 Timer interrupt.

12.5.6 enum lpit_status_flags_t

Note

Number of timer channels are SoC-specific. See the SoC Reference Manual.

Enumerator

kLPIT_Channel0TimerFlag Channel 0 Timer interrupt flag.
kLPIT_Channel1TimerFlag Channel 1 Timer interrupt flag.
kLPIT_Channel2TimerFlag Channel 2 Timer interrupt flag.
kLPIT_Channel3TimerFlag Channel 3 Timer interrupt flag.

12.6 Function Documentation

12.6.1 void LPIT_Init (LPIT_Type ∗ *base*, const lpit_config_t ∗ *config*)

This function issues a software reset to reset all channels and registers except the Module Control register.

Note

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

Parameters

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

12.6.2 void LPIT_Deinit (LPIT_Type ∗ *base*)

Parameters

base	LPIT peripheral base address.

12.6.3 void LPIT_GetDefaultConfig (lpit_config_t ∗ *config*)

The default values are:

```
*    config->enableRunInDebug = false;
*    config->enableRunInDoze = false;
*
```

Parameters

config	Pointer to the user configuration structure.

12.6.4 status_t LPIT_SetupChannel (LPIT_Type ∗ *base*, lpit_chnl_t *channel*, const lpit_chnl_params_t ∗ *chnlSetup*)

This function sets up the operation mode to one of the options available in the enumeration lpit_timer_-modes_t. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

base	LPIT peripheral base address.
channel	Channel that is being configured.
chnlSetup	Configuration parameters.

12.6.5 static void LPIT_EnableInterrupts (LPIT_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LPIT peripheral base address.
mask	The interrupts to enable. This is a logical OR of members of the enumeration lpit_-interrupt_enable_t

12.6.6 static void LPIT_DisableInterrupts (LPIT_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LPIT peripheral base address.
mask	The interrupts to enable. This is a logical OR of members of the enumeration lpit_-interrupt_enable_t

12.6.7 static uint32_t LPIT_GetEnabledInterrupts (LPIT_Type * base) [inline], [static]

Parameters

base	LPIT peripheral base address.

Returns

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

12.6.8 static uint32_t LPIT_GetStatusFlags (LPIT_Type * *base*) [inline], [static]

Parameters

base	LPIT peripheral base address.

Returns

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

12.6.9 static void LPIT_ClearStatusFlags (LPIT_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

12.6.10 static void LPIT_SetTimerPeriod (LPIT_Type * *base*, lpit_chnl_t *channel*, uint32_t *ticks*) [inline], [static]

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note

User can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

base	LPIT peripheral base address.
channel	Timer channel number.
ticks	Timer period in units of ticks.

12.6.11 static uint32_t LPIT_GetCurrentTimerCount (LPIT_Type * *base*, lpit_chnl_t *channel*) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in fsl_common.h to convert ticks to microseconds or milliseconds.

Parameters

base	LPIT peripheral base address.
channel	Timer channel number.

Returns

Current timer counting value in ticks.

12.6.12 static void LPIT_StartTimer (LPIT_Type * *base*, lpit_chnl_t *channel*) [inline], [static]

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

base	LPIT peripheral base address.
channel	Timer channel number.

12.6.13 static void LPIT_StopTimer (LPIT_Type * *base*, lpit_chnl_t *channel*) [inline], [static]

Parameters

base	LPIT peripheral base address.
channel	Timer channel number.

12.6.14 static void LPIT_Reset (LPIT_Type * *base*) [inline], [static]

This resets all channels and registers except the Module Control Register.

Parameters

base	LPIT peripheral base address.

Chapter 13
LPSPI: Low Power Serial Peripheral Interface

13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power Serial Peripheral Interface (LPSPI) module of MCUXpresso SDK devices.

Modules

- LPSPI Peripheral driver

13.2 LPSPI Peripheral driver

13.2.1 Overview

This section describes the programming interface of the LPSPI Peripheral driver. The LPSPI driver configures LPSPI module, provides the functional and transactional interfaces to build the LPSPI application.

13.2.2 Function groups

13.2.2.1 LPSPI Initialization and De-initialization

This function group initializes the default configuration structure for master and slave, initializes the LPSPI master with a master configuration, initializes the LPSPI slave with a slave configuration, and de-initializes the LPSPI module.

13.2.2.2 LPSPI Basic Operation

This function group enables/disables the LPSPI module both interrupt and DMA, gets the data register address for the DMA transfer, sets master and slave, starts and stops the transfer, and so on.

13.2.2.3 LPSPI Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

13.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

13.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

13.2.3 Typical use case

13.2.3.1 Master Operation

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

13.2.3.2 Slave Operation

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

Data Structures

- struct lpspi_master_config_t
  LPSPI master configuration structure. More...
- struct lpspi_slave_config_t
  LPSPI slave configuration structure. More...
- struct lpspi_transfer_t
  LPSPI master/slave transfer structure. More...
- struct lpspi_master_handle_t
  LPSPI master transfer handle structure used for transactional API. More...
- struct lpspi_slave_handle_t
  LPSPI slave transfer handle structure used for transactional API. More...

Macros

- #define LPSPI_DUMMY_DATA (0x00U)
  LPSPI dummy data if no Tx data.
- #define SPI_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.
- #define LPSPI_MASTER_PCS_SHIFT (4U)
  LPSPI master PCS shift macro , internal used.
- #define LPSPI_MASTER_PCS_MASK (0xF0U)
  LPSPI master PCS shift macro , internal used.
- #define LPSPI_SLAVE_PCS_SHIFT (4U)
  LPSPI slave PCS shift macro , internal used.
- #define LPSPI_SLAVE_PCS_MASK (0xF0U)
  LPSPI slave PCS shift macro , internal used.

Typedefs

- typedef void(∗ lpspi_master_transfer_callback_t)(LPSPI_Type ∗base, lpspi_master_handle_t ∗handle, status_t status, void ∗userData)
  Master completion callback function pointer type.
- typedef void(∗ lpspi_slave_transfer_callback_t)(LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle, status_t status, void ∗userData)
  Slave completion callback function pointer type.

Enumerations

- enum {
  kStatus_LPSPI_Busy = MAKE_STATUS(kStatusGroup_LPSPI, 0),
  kStatus_LPSPI_Error = MAKE_STATUS(kStatusGroup_LPSPI, 1),
  kStatus_LPSPI_Idle = MAKE_STATUS(kStatusGroup_LPSPI, 2),
  kStatus_LPSPI_OutOfRange = MAKE_STATUS(kStatusGroup_LPSPI, 3),
  kStatus_LPSPI_Timeout = MAKE_STATUS(kStatusGroup_LPSPI, 4) }
  Status for the LPSPI driver.
- enum _lpspi_flags {
  kLPSPI_TxDataRequestFlag = LPSPI_SR_TDF_MASK,
  kLPSPI_RxDataReadyFlag = LPSPI_SR_RDF_MASK,
  kLPSPI_WordCompleteFlag = LPSPI_SR_WCF_MASK,
  kLPSPI_FrameCompleteFlag = LPSPI_SR_FCF_MASK,
  kLPSPI_TransferCompleteFlag = LPSPI_SR_TCF_MASK,
  kLPSPI_TransmitErrorFlag = LPSPI_SR_TEF_MASK,
  kLPSPI_ReceiveErrorFlag = LPSPI_SR_REF_MASK,
  kLPSPI_DataMatchFlag = LPSPI_SR_DMF_MASK,
  kLPSPI_ModuleBusyFlag = LPSPI_SR_MBF_MASK,
  kLPSPI_AllStatusFlag }
  LPSPI status flags in SPIx_SR register.
- enum _lpspi_interrupt_enable {
  kLPSPI_TxInterruptEnable = LPSPI_IER_TDIE_MASK,
  kLPSPI_RxInterruptEnable = LPSPI_IER_RDIE_MASK,
  kLPSPI_WordCompleteInterruptEnable = LPSPI_IER_WCIE_MASK,
  kLPSPI_FrameCompleteInterruptEnable = LPSPI_IER_FCIE_MASK,
  kLPSPI_TransferCompleteInterruptEnable = LPSPI_IER_TCIE_MASK,
  kLPSPI_TransmitErrorInterruptEnable = LPSPI_IER_TEIE_MASK,
  kLPSPI_ReceiveErrorInterruptEnable = LPSPI_IER_REIE_MASK,
  kLPSPI_DataMatchInterruptEnable = LPSPI_IER_DMIE_MASK,
  kLPSPI_AllInterruptEnable }
  LPSPI interrupt source.
- enum _lpspi_dma_enable {
  kLPSPI_TxDmaEnable = LPSPI_DER_TDDE_MASK,
  kLPSPI_RxDmaEnable = LPSPI_DER_RDDE_MASK }
  LPSPI DMA source.
- enum lpspi_master_slave_mode_t {
  kLPSPI_Master = 1U,
  kLPSPI_Slave = 0U }
  LPSPI master or slave mode configuration.
- enum lpspi_which_pcs_t {
  kLPSPI_Pcs0 = 0U,
  kLPSPI_Pcs1 = 1U,
  kLPSPI_Pcs2 = 2U,
  kLPSPI_Pcs3 = 3U }
  LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

- enum lpspi_pcs_polarity_config_t {
  kLPSPI_PcsActiveHigh = 1U,
  kLPSPI_PcsActiveLow = 0U }
  LPSPI Peripheral Chip Select (PCS) Polarity configuration.
- enum _lpspi_pcs_polarity {
  kLPSPI_Pcs0ActiveLow = 1U << 0,
  kLPSPI_Pcs1ActiveLow = 1U << 1,
  kLPSPI_Pcs2ActiveLow = 1U << 2,
  kLPSPI_Pcs3ActiveLow = 1U << 3,
  kLPSPI_PcsAllActiveLow = 0xFU }
  LPSPI Peripheral Chip Select (PCS) Polarity.
- enum lpspi_clock_polarity_t {
  kLPSPI_ClockPolarityActiveHigh = 0U,
  kLPSPI_ClockPolarityActiveLow = 1U }
  LPSPI clock polarity configuration.
- enum lpspi_clock_phase_t {
  kLPSPI_ClockPhaseFirstEdge = 0U,
  kLPSPI_ClockPhaseSecondEdge = 1U }
  LPSPI clock phase configuration.
- enum lpspi_shift_direction_t {
  kLPSPI_MsbFirst = 0U,
  kLPSPI_LsbFirst = 1U }
  LPSPI data shifter direction options.
- enum lpspi_host_request_select_t {
  kLPSPI_HostReqExtPin = 0U,
  kLPSPI_HostReqInternalTrigger = 1U }
  LPSPI Host Request select configuration.
- enum lpspi_match_config_t {
  kLPSI_MatchDisabled = 0x0U,
  kLPSI_1stWordEqualsM0orM1 = 0x2U,
  kLPSI_AnyWordEqualsM0orM1 = 0x3U,
  kLPSI_1stWordEqualsM0and2ndWordEqualsM1 = 0x4U,
  kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 = 0x5U,
  kLPSI_1stWordAndM1EqualsM0andM1 = 0x6U,
  kLPSI_AnyWordAndM1EqualsM0andM1 = 0x7U }
  LPSPI Match configuration options.
- enum lpspi_pin_config_t {
  kLPSPI_SdiInSdoOut = 0U,
  kLPSPI_SdiInSdiOut = 1U,
  kLPSPI_SdoInSdoOut = 2U,
  kLPSPI_SdoInSdiOut = 3U }
  LPSPI pin (SDO and SDI) configuration.
- enum lpspi_data_out_config_t {
  kLpspiDataOutRetained = 0U,
  kLpspiDataOutTristate = 1U }
  LPSPI data output configuration.
- enum lpspi_transfer_width_t {

kLPSPI_SingleBitXfer = 0U,
kLPSPI_TwoBitXfer = 1U,
kLPSPI_FourBitXfer = 2U }
LPSPI transfer width configuration.
- enum lpspi_delay_type_t {
kLPSPI_PcsToSck = 1U,
kLPSPI_LastSckToPcs,
kLPSPI_BetweenTransfer }
LPSPI delay type selection.
- enum _lpspi_transfer_config_flag_for_master {
kLPSPI_MasterPcs0 = 0U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs1 = 1U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs2 = 2U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs3 = 3U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcsContinuous = 1U << 20,
kLPSPI_MasterByteSwap }
Use this enumeration for LPSPI master transfer configFlags.
- enum _lpspi_transfer_config_flag_for_slave {
kLPSPI_SlavePcs0 = 0U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs1 = 1U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs2 = 2U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs3 = 3U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlaveByteSwap }
Use this enumeration for LPSPI slave transfer configFlags.
- enum _lpspi_transfer_state {
kLPSPI_Idle = 0x0U,
kLPSPI_Busy,
kLPSPI_Error }
LPSPI transfer state, which is used for LPSPI transactional API state machine.

Variables

- volatile uint8_t g_lpspiDummyData []
Global variable for dummy data value setting.

Driver version

- #define FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
LPSPI driver version.

Initialization and deinitialization

- void LPSPI_MasterInit (LPSPI_Type ∗base, const lpspi_master_config_t ∗masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.
- void LPSPI_MasterGetDefaultConfig (lpspi_master_config_t *masterConfig)
  Sets the lpspi_master_config_t structure to default values.
- void LPSPI_SlaveInit (LPSPI_Type *base, const lpspi_slave_config_t *slaveConfig)
  LPSPI slave configuration.
- void LPSPI_SlaveGetDefaultConfig (lpspi_slave_config_t *slaveConfig)
  Sets the lpspi_slave_config_t structure to default values.
- void LPSPI_Deinit (LPSPI_Type *base)
  De-initializes the LPSPI peripheral.
- void LPSPI_Reset (LPSPI_Type *base)
  Restores the LPSPI peripheral to reset state.
- uint32_t LPSPI_GetInstance (LPSPI_Type *base)
  Get the LPSPI instance from peripheral base address.
- static void LPSPI_Enable (LPSPI_Type *base, bool enable)
  Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Status

- static uint32_t LPSPI_GetStatusFlags (LPSPI_Type *base)
  Gets the LPSPI status flag state.
- static uint8_t LPSPI_GetTxFifoSize (LPSPI_Type *base)
  Gets the LPSPI Tx FIFO size.
- static uint8_t LPSPI_GetRxFifoSize (LPSPI_Type *base)
  Gets the LPSPI Rx FIFO size.
- static uint32_t LPSPI_GetTxFifoCount (LPSPI_Type *base)
  Gets the LPSPI Tx FIFO count.
- static uint32_t LPSPI_GetRxFifoCount (LPSPI_Type *base)
  Gets the LPSPI Rx FIFO count.
- static void LPSPI_ClearStatusFlags (LPSPI_Type *base, uint32_t statusFlags)
  Clears the LPSPI status flag.

Interrupts

- static void LPSPI_EnableInterrupts (LPSPI_Type *base, uint32_t mask)
  Enables the LPSPI interrupts.
- static void LPSPI_DisableInterrupts (LPSPI_Type *base, uint32_t mask)
  Disables the LPSPI interrupts.

DMA Control

- static void LPSPI_EnableDMA (LPSPI_Type *base, uint32_t mask)
  Enables the LPSPI DMA request.
- static void LPSPI_DisableDMA (LPSPI_Type *base, uint32_t mask)
  Disables the LPSPI DMA request.
- static uint32_t LPSPI_GetTxRegisterAddress (LPSPI_Type *base)
  Gets the LPSPI Transmit Data Register address for a DMA operation.
- static uint32_t LPSPI_GetRxRegisterAddress (LPSPI_Type *base)

Gets the LPSPI Receive Data Register address for a DMA operation.

Bus Operations

- bool LPSPI_CheckTransferArgument (LPSPI_Type ∗base, lpspi_transfer_t ∗transfer, bool isEdma)
  Check the argument for transfer .
- static void LPSPI_SetMasterSlaveMode (LPSPI_Type ∗base, lpspi_master_slave_mode_t mode)
  Configures the LPSPI for either master or slave.
- static void LPSPI_SelectTransferPCS (LPSPI_Type ∗base, lpspi_which_pcs_t select)
  Configures the peripheral chip select used for the transfer.
- static void LPSPI_SetPCSContinous (LPSPI_Type ∗base, bool IsContinous)
  Set the PCS signal to continuous or uncontinuous mode.
- static bool LPSPI_IsMaster (LPSPI_Type ∗base)
  Returns whether the LPSPI module is in master mode.
- static void LPSPI_FlushFifo (LPSPI_Type ∗base, bool flushTxFifo, bool flushRxFifo)
  Flushes the LPSPI FIFOs.
- static void LPSPI_SetFifoWatermarks (LPSPI_Type ∗base, uint32_t txWater, uint32_t rxWater)
  Sets the transmit and receive FIFO watermark values.
- static void LPSPI_SetAllPcsPolarity (LPSPI_Type ∗base, uint32_t mask)
  Configures all LPSPI peripheral chip select polarities simultaneously.
- static void LPSPI_SetFrameSize (LPSPI_Type ∗base, uint32_t frameSize)
  Configures the frame size.
- uint32_t LPSPI_MasterSetBaudRate (LPSPI_Type ∗base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t ∗tcrPrescaleValue)
  Sets the LPSPI baud rate in bits per second.
- void LPSPI_MasterSetDelayScaler (LPSPI_Type ∗base, uint32_t scaler, lpspi_delay_type_t whichDelay)
  Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).
- uint32_t LPSPI_MasterSetDelayTimes (LPSPI_Type ∗base, uint32_t delayTimeInNanoSec, lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)
  Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).
- static void LPSPI_WriteData (LPSPI_Type ∗base, uint32_t data)
  Writes data into the transmit data buffer.
- static uint32_t LPSPI_ReadData (LPSPI_Type ∗base)
  Reads data from the data buffer.
- void LPSPI_SetDummyData (LPSPI_Type ∗base, uint8_t dummyData)
  Set up the dummy data.

Transactional

- void LPSPI_MasterTransferCreateHandle (LPSPI_Type ∗base, lpspi_master_handle_t ∗handle, lpspi_master_transfer_callback_t callback, void ∗userData)
  Initializes the LPSPI master handle.
- status_t LPSPI_MasterTransferBlocking (LPSPI_Type ∗base, lpspi_transfer_t ∗transfer)
  LPSPI master transfer data using a polling method.

- status_t LPSPI_MasterTransferNonBlocking (LPSPI_Type ∗base, lpspi_master_handle_t ∗handle, lpspi_transfer_t ∗transfer)
  LPSPI master transfer data using an interrupt method.
- status_t LPSPI_MasterTransferGetCount (LPSPI_Type ∗base, lpspi_master_handle_t ∗handle, size_t ∗count)
  Gets the master transfer remaining bytes.
- void LPSPI_MasterTransferAbort (LPSPI_Type ∗base, lpspi_master_handle_t ∗handle)
  LPSPI master abort transfer which uses an interrupt method.
- void LPSPI_MasterTransferHandleIRQ (LPSPI_Type ∗base, lpspi_master_handle_t ∗handle)
  LPSPI Master IRQ handler function.
- void LPSPI_SlaveTransferCreateHandle (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle, lpspi_slave_transfer_callback_t callback, void ∗userData)
  Initializes the LPSPI slave handle.
- status_t LPSPI_SlaveTransferNonBlocking (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle, lpspi_transfer_t ∗transfer)
  LPSPI slave transfer data using an interrupt method.
- status_t LPSPI_SlaveTransferGetCount (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle, size_t ∗count)
  Gets the slave transfer remaining bytes.
- void LPSPI_SlaveTransferAbort (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle)
  LPSPI slave aborts a transfer which uses an interrupt method.
- void LPSPI_SlaveTransferHandleIRQ (LPSPI_Type ∗base, lpspi_slave_handle_t ∗handle)
  LPSPI Slave IRQ handler function.

13.2.4 Data Structure Documentation

13.2.4.1 struct lpspi_master_config_t

Data Fields

- uint32_t baudRate
  Baud Rate for LPSPI.
- uint32_t bitsPerFrame
  Bits per frame, minimum 8, maximum 4096.
- lpspi_clock_polarity_t cpol
  Clock polarity.
- lpspi_clock_phase_t cpha
  Clock phase.
- lpspi_shift_direction_t direction
  MSB or LSB data shift direction.
- uint32_t pcsToSckDelayInNanoSec
  PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay.
- uint32_t lastSckToPcsDelayInNanoSec
  Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay.
- uint32_t betweenTransferDelayInNanoSec
  `After the SCK delay time with nanoseconds, setting to 0 sets the` *minimum delay.*
- lpspi_which_pcs_t whichPcs
  Desired Peripheral Chip Select (PCS).

- lpspi_pcs_polarity_config_t pcsActiveHighOrLow
  Desired PCS active high or low.
- lpspi_pin_config_t pinCfg
  Configures which pins are used for input and output data during single bit transfers.
- lpspi_data_out_config_t dataOutConfig
  Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

Field Documentation

(1) uint32_t lpspi_master_config_t::baudRate

(2) uint32_t lpspi_master_config_t::bitsPerFrame

(3) lpspi_clock_polarity_t lpspi_master_config_t::cpol

(4) lpspi_clock_phase_t lpspi_master_config_t::cpha

(5) lpspi_shift_direction_t lpspi_master_config_t::direction

(6) uint32_t lpspi_master_config_t::pcsToSckDelayInNanoSec

It sets the boundary value if out of range.

(7) uint32_t lpspi_master_config_t::lastSckToPcsDelayInNanoSec

It sets the boundary value if out of range.

(8) uint32_t lpspi_master_config_t::betweenTransferDelayInNanoSec

It sets the boundary value if out of range.

(9) lpspi_which_pcs_t lpspi_master_config_t::whichPcs

(10) lpspi_pin_config_t lpspi_master_config_t::pinCfg

(11) lpspi_data_out_config_t lpspi_master_config_t::dataOutConfig

13.2.4.2 struct lpspi_slave_config_t

Data Fields

- uint32_t bitsPerFrame
  Bits per frame, minimum 8, maximum 4096.
- lpspi_clock_polarity_t cpol
  Clock polarity.
- lpspi_clock_phase_t cpha
  Clock phase.
- lpspi_shift_direction_t direction
  MSB or LSB data shift direction.
- lpspi_which_pcs_t whichPcs
  Desired Peripheral Chip Select (pcs)

- lpspi_pcs_polarity_config_t pcsActiveHighOrLow
  Desired PCS active high or low.
- lpspi_pin_config_t pinCfg
  Configures which pins are used for input and output data during single bit transfers.
- lpspi_data_out_config_t dataOutConfig
  Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

Field Documentation

(1) uint32_t lpspi_slave_config_t::bitsPerFrame

(2) lpspi_clock_polarity_t lpspi_slave_config_t::cpol

(3) lpspi_clock_phase_t lpspi_slave_config_t::cpha

(4) lpspi_shift_direction_t lpspi_slave_config_t::direction

(5) lpspi_pin_config_t lpspi_slave_config_t::pinCfg

(6) lpspi_data_out_config_t lpspi_slave_config_t::dataOutConfig

13.2.4.3 struct lpspi_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- volatile size_t dataSize
  Transfer bytes.
- uint32_t configFlags
  Transfer transfer configuration flags.

Field Documentation

(1) uint8_t* lpspi_transfer_t::txData

(2) uint8_t* lpspi_transfer_t::rxData

(3) volatile size_t lpspi_transfer_t::dataSize

(4) uint32_t lpspi_transfer_t::configFlags

Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_-config_flag_for_slave enumeration if the transfer is used for slave.

13.2.4.4 struct _lpspi_master_handle

Forward declaration of the _lpspi_master_handle typedefs.

Data Fields

- volatile bool isPcsContinuous
  Is PCS continuous in transfer.
- volatile bool writeTcrInIsr
  A flag that whether should write TCR in ISR.
- volatile bool isByteSwap
  A flag that whether should byte swap.
- volatile bool isTxMask
  A flag that whether TCR[TXMSK] is set.
- volatile uint16_t bytesPerFrame
  Number of bytes in each frame.
- volatile uint8_t fifoSize
  FIFO dataSize.
- volatile uint8_t rxWatermark
  Rx watermark.
- volatile uint8_t bytesEachWrite
  Bytes for each write TDR.
- volatile uint8_t bytesEachRead
  Bytes for each read RDR.
- uint8_t ∗volatile txData
  Send buffer.
- uint8_t ∗volatile rxData
  Receive buffer.
- volatile size_t txRemainingByteCount
  Number of bytes remaining to send.
- volatile size_t rxRemainingByteCount
  Number of bytes remaining to receive.
- volatile uint32_t writeRegRemainingTimes
  Write TDR register remaining times.
- volatile uint32_t readRegRemainingTimes
  Read RDR register remaining times.
- uint32_t totalByteCount
  Number of transfer bytes.
- uint32_t txBuffIfNull
  Used if the txData is NULL.
- volatile uint8_t state
  LPSPI transfer state , _lpspi_transfer_state.
- lpspi_master_transfer_callback_t callback
  Completion callback.
- void ∗ userData
  Callback user data.

Field Documentation

(1) volatile bool lpspi_master_handle_t::isPcsContinuous

(2) volatile bool lpspi_master_handle_t::writeTcrInIsr

(3) volatile bool lpspi_master_handle_t::isByteSwap

(4) volatile bool lpspi_master_handle_t::isTxMask

(5) volatile uint8_t lpspi_master_handle_t::fifoSize

(6) volatile uint8_t lpspi_master_handle_t::rxWatermark

(7) volatile uint8_t lpspi_master_handle_t::bytesEachWrite

(8) volatile uint8_t lpspi_master_handle_t::bytesEachRead

(9) uint8_t* volatile lpspi_master_handle_t::txData

(10) uint8_t* volatile lpspi_master_handle_t::rxData

(11) volatile size_t lpspi_master_handle_t::txRemainingByteCount

(12) volatile size_t lpspi_master_handle_t::rxRemainingByteCount

(13) volatile uint32_t lpspi_master_handle_t::writeRegRemainingTimes

(14) volatile uint32_t lpspi_master_handle_t::readRegRemainingTimes

(15) uint32_t lpspi_master_handle_t::txBuffIfNull

(16) volatile uint8_t lpspi_master_handle_t::state

(17) lpspi_master_transfer_callback_t lpspi_master_handle_t::callback

(18) void* lpspi_master_handle_t::userData

13.2.4.5 struct _lpspi_slave_handle

Forward declaration of the _lpspi_slave_handle typedefs.

Data Fields

- volatile bool isByteSwap
  A flag that whether should byte swap.
- volatile uint8_t fifoSize
  FIFO dataSize.
- volatile uint8_t rxWatermark
  Rx watermark.
- volatile uint8_t bytesEachWrite
  Bytes for each write TDR.
- volatile uint8_t bytesEachRead
  Bytes for each read RDR.
- uint8_t *volatile txData
  Send buffer.
- uint8_t *volatile rxData
  Receive buffer.

- volatile size_t txRemainingByteCount
  Number of bytes remaining to send.
- volatile size_t rxRemainingByteCount
  Number of bytes remaining to receive.
- volatile uint32_t writeRegRemainingTimes
  Write TDR register remaining times.
- volatile uint32_t readRegRemainingTimes
  Read RDR register remaining times.
- uint32_t totalByteCount
  Number of transfer bytes.
- volatile uint8_t state
  LPSPI transfer state , _lpspi_transfer_state.
- volatile uint32_t errorCount
  Error count for slave transfer.
- lpspi_slave_transfer_callback_t callback
  Completion callback.
- void ∗ userData
  Callback user data.

Field Documentation

(1) volatile bool lpspi_slave_handle_t::isByteSwap

(2) volatile uint8_t lpspi_slave_handle_t::fifoSize

(3) volatile uint8_t lpspi_slave_handle_t::rxWatermark

(4) volatile uint8_t lpspi_slave_handle_t::bytesEachWrite

(5) volatile uint8_t lpspi_slave_handle_t::bytesEachRead

(6) uint8_t∗ volatile lpspi_slave_handle_t::txData

(7) uint8_t∗ volatile lpspi_slave_handle_t::rxData

(8) volatile size_t lpspi_slave_handle_t::txRemainingByteCount

(9) volatile size_t lpspi_slave_handle_t::rxRemainingByteCount

(10) volatile uint32_t lpspi_slave_handle_t::writeRegRemainingTimes

(11) volatile uint32_t lpspi_slave_handle_t::readRegRemainingTimes

(12) volatile uint8_t lpspi_slave_handle_t::state

(13) volatile uint32_t lpspi_slave_handle_t::errorCount

(14) lpspi_slave_transfer_callback_t lpspi_slave_handle_t::callback

(15) void∗ lpspi_slave_handle_t::userData

13.2.5 Macro Definition Documentation

13.2.5.1 #define FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

13.2.5.2 #define LPSPI_DUMMY_DATA (0x00U)

Dummy data used for tx if there is not txData.

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

13.2.5.4 #define LPSPI_MASTER_PCS_SHIFT (4U)

13.2.5.5 #define LPSPI_MASTER_PCS_MASK (0xF0U)

13.2.5.6 #define LPSPI_SLAVE_PCS_SHIFT (4U)

13.2.5.7 #define LPSPI_SLAVE_PCS_MASK (0xF0U)

13.2.6 Typedef Documentation

13.2.6.1 typedef void(* lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)

Parameters

base	LPSPI peripheral address.
handle	Pointer to the handle for the LPSPI master.
status	Success or error code describing whether the transfer is completed.
userData	Arbitrary pointer-dataSized value passed from the application.

13.2.6.2 typedef void(* lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle, status_t status, void *userData)

Parameters

base	LPSPI peripheral address.
handle	Pointer to the handle for the LPSPI slave.
status	Success or error code describing whether the transfer is completed.
userData	Arbitrary pointer-dataSized value passed from the application.

13.2.7 Enumeration Type Documentation

13.2.7.1 anonymous enum

Enumerator

kStatus_LPSPI_Busy LPSPI transfer is busy.
kStatus_LPSPI_Error LPSPI driver error.
kStatus_LPSPI_Idle LPSPI is idle.
kStatus_LPSPI_OutOfRange LPSPI transfer out Of range.
kStatus_LPSPI_Timeout LPSPI timeout polling status flags.

13.2.7.2 enum _lpspi_flags

Enumerator

kLPSPI_TxDataRequestFlag Transmit data flag.
kLPSPI_RxDataReadyFlag Receive data flag.
kLPSPI_WordCompleteFlag Word Complete flag.
kLPSPI_FrameCompleteFlag Frame Complete flag.
kLPSPI_TransferCompleteFlag Transfer Complete flag.
kLPSPI_TransmitErrorFlag Transmit Error flag (FIFO underrun)
kLPSPI_ReceiveErrorFlag Receive Error flag (FIFO overrun)
kLPSPI_DataMatchFlag Data Match flag.
kLPSPI_ModuleBusyFlag Module Busy flag.
kLPSPI_AllStatusFlag Used for clearing all w1c status flags.

13.2.7.3 enum _lpspi_interrupt_enable

Enumerator

kLPSPI_TxInterruptEnable Transmit data interrupt enable.
kLPSPI_RxInterruptEnable Receive data interrupt enable.
kLPSPI_WordCompleteInterruptEnable Word complete interrupt enable.
kLPSPI_FrameCompleteInterruptEnable Frame complete interrupt enable.
kLPSPI_TransferCompleteInterruptEnable Transfer complete interrupt enable.

kLPSPI_TransmitErrorInterruptEnable Transmit error interrupt enable(FIFO underrun)
kLPSPI_ReceiveErrorInterruptEnable Receive Error interrupt enable (FIFO overrun)
kLPSPI_DataMatchInterruptEnable Data Match interrupt enable.
kLPSPI_AllInterruptEnable All above interrupts enable.

13.2.7.4 enum _lpspi_dma_enable

Enumerator

kLPSPI_TxDmaEnable Transmit data DMA enable.
kLPSPI_RxDmaEnable Receive data DMA enable.

13.2.7.5 enum lpspi_master_slave_mode_t

Enumerator

kLPSPI_Master LPSPI peripheral operates in master mode.
kLPSPI_Slave LPSPI peripheral operates in slave mode.

13.2.7.6 enum lpspi_which_pcs_t

Enumerator

kLPSPI_Pcs0 PCS[0].
kLPSPI_Pcs1 PCS[1].
kLPSPI_Pcs2 PCS[2].
kLPSPI_Pcs3 PCS[3].

13.2.7.7 enum lpspi_pcs_polarity_config_t

Enumerator

kLPSPI_PcsActiveHigh PCS Active High (idles low)
kLPSPI_PcsActiveLow PCS Active Low (idles high)

13.2.7.8 enum _lpspi_pcs_polarity

Enumerator

kLPSPI_Pcs0ActiveLow Pcs0 Active Low (idles high).
kLPSPI_Pcs1ActiveLow Pcs1 Active Low (idles high).

kLPSPI_Pcs2ActiveLow Pcs2 Active Low (idles high).
kLPSPI_Pcs3ActiveLow Pcs3 Active Low (idles high).
kLPSPI_PcsAllActiveLow Pcs0 to Pcs5 Active Low (idles high).

13.2.7.9 enum lpspi_clock_polarity_t

Enumerator

kLPSPI_ClockPolarityActiveHigh CPOL=0. Active-high LPSPI clock (idles low)
kLPSPI_ClockPolarityActiveLow CPOL=1. Active-low LPSPI clock (idles high)

13.2.7.10 enum lpspi_clock_phase_t

Enumerator

kLPSPI_ClockPhaseFirstEdge CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.
kLPSPI_ClockPhaseSecondEdge CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

13.2.7.11 enum lpspi_shift_direction_t

Enumerator

kLPSPI_MsbFirst Data transfers start with most significant bit.
kLPSPI_LsbFirst Data transfers start with least significant bit.

13.2.7.12 enum lpspi_host_request_select_t

Enumerator

kLPSPI_HostReqExtPin Host Request is an ext pin.
kLPSPI_HostReqInternalTrigger Host Request is an internal trigger.

13.2.7.13 enum lpspi_match_config_t

Enumerator

kLPSI_MatchDisabled LPSPI Match Disabled.
kLPSI_1stWordEqualsM0orM1 LPSPI Match Enabled.
kLPSI_AnyWordEqualsM0orM1 LPSPI Match Enabled.

kLPSI_1stWordEqualsM0and2ndWordEqualsM1 LPSPI Match Enabled.
kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 LPSPI Match Enabled.
kLPSI_1stWordAndM1EqualsM0andM1 LPSPI Match Enabled.
kLPSI_AnyWordAndM1EqualsM0andM1 LPSPI Match Enabled.

13.2.7.14 enum lpspi_pin_config_t

Enumerator

kLPSPI_SdiInSdoOut LPSPI SDI input, SDO output.
kLPSPI_SdiInSdiOut LPSPI SDI input, SDI output.
kLPSPI_SdoInSdoOut LPSPI SDO input, SDO output.
kLPSPI_SdoInSdiOut LPSPI SDO input, SDI output.

13.2.7.15 enum lpspi_data_out_config_t

Enumerator

kLpspiDataOutRetained Data out retains last value when chip select is de-asserted.
kLpspiDataOutTristate Data out is tristated when chip select is de-asserted.

13.2.7.16 enum lpspi_transfer_width_t

Enumerator

kLPSPI_SingleBitXfer 1-bit shift at a time, data out on SDO, in on SDI (normal mode)
kLPSPI_TwoBitXfer 2-bits shift out on SDO/SDI and in on SDO/SDI
kLPSPI_FourBitXfer 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

13.2.7.17 enum lpspi_delay_type_t

Enumerator

kLPSPI_PcsToSck PCS-to-SCK delay.
kLPSPI_LastSckToPcs Last SCK edge to PCS delay.
kLPSPI_BetweenTransfer Delay between transfers.

13.2.7.18 enum _lpspi_transfer_config_flag_for_master

Enumerator

kLPSPI_MasterPcs0 LPSPI master transfer use PCS0 signal.
kLPSPI_MasterPcs1 LPSPI master transfer use PCS1 signal.
kLPSPI_MasterPcs2 LPSPI master transfer use PCS2 signal.
kLPSPI_MasterPcs3 LPSPI master transfer use PCS3 signal.
kLPSPI_MasterPcsContinuous Is PCS signal continuous.
kLPSPI_MasterByteSwap Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

1. If you set bitPerFrame = 8 , no matter the kLPSPI_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_-MasterByteSwap flag.
3. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_-MasterByteSwap flag.

13.2.7.19 enum _lpspi_transfer_config_flag_for_slave

Enumerator

kLPSPI_SlavePcs0 LPSPI slave transfer use PCS0 signal.
kLPSPI_SlavePcs1 LPSPI slave transfer use PCS1 signal.
kLPSPI_SlavePcs2 LPSPI slave transfer use PCS2 signal.
kLPSPI_SlavePcs3 LPSPI slave transfer use PCS3 signal.
kLPSPI_SlaveByteSwap Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

1. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_-SlaveByteSwap flag.
3. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_-SlaveByteSwap flag.

13.2.7.20 enum _lpspi_transfer_state

Enumerator

kLPSPI_Idle Nothing in the transmitter/receiver.

kLPSPI_Busy Transfer queue is not finished.
kLPSPI_Error Transfer error.

13.2.8 Function Documentation

13.2.8.1 void LPSPI_MasterInit (LPSPI_Type * *base*, const lpspi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

Parameters

base	LPSPI peripheral address.
masterConfig	Pointer to structure lpspi_master_config_t.
srcClock_Hz	Module source input clock in Hertz

13.2.8.2 void LPSPI_MasterGetDefaultConfig (lpspi_master_config_t * *masterConfig*)

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
*  lpspi_master_config_t  masterConfig;
*  LPSPI_MasterGetDefaultConfig(&masterConfig);
*
```

Parameters

masterConfig	pointer to lpspi_master_config_t structure

13.2.8.3 void LPSPI_SlaveInit (LPSPI_Type * *base*, const lpspi_slave_config_t * *slaveConfig*)

Parameters

base	LPSPI peripheral address.
slaveConfig	Pointer to a structure lpspi_slave_config_t.

13.2.8.4 void LPSPI_SlaveGetDefaultConfig (lpspi_slave_config_t * *slaveConfig*)

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
* lpspi_slave_config_t  slaveConfig;
* LPSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

slaveConfig	pointer to lpspi_slave_config_t structure.

13.2.8.5 void LPSPI_Deinit (LPSPI_Type ∗ *base*)

Call this API to disable the LPSPI clock.

Parameters

base	LPSPI peripheral address.

13.2.8.6 void LPSPI_Reset (LPSPI_Type ∗ *base*)

Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

base	LPSPI peripheral address.

13.2.8.7 uint32_t LPSPI_GetInstance (LPSPI_Type ∗ *base*)

Parameters

base	LPSPI peripheral base address.

Returns

LPSPI instance.

13.2.8.8 static void LPSPI_Enable (LPSPI_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	LPSPI peripheral address.
enable	Pass true to enable module, false to disable module.

13.2.8.9 static uint32_t LPSPI_GetStatusFlags (LPSPI_Type * *base*) [inline], [static]

Parameters

base	LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

13.2.8.10 static uint8_t LPSPI_GetTxFifoSize (LPSPI_Type * *base*) [inline], [static]

Parameters

base	LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

13.2.8.11 static uint8_t LPSPI_GetRxFifoSize (LPSPI_Type * *base*) [inline], [static]

Parameters

base	LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

13.2.8.12 static uint32_t LPSPI_GetTxFifoCount (LPSPI_Type * *base*) [inline], [static]

Parameters

base	LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

13.2.8.13 static uint32_t LPSPI_GetRxFifoCount (LPSPI_Type * *base*) [inline], [static]

Parameters

base	LPSPI peripheral address.

Returns

The number of words in the receive FIFO.

13.2.8.14 static void LPSPI_ClearStatusFlags (LPSPI_Type * *base*, uint32_t *statusFlags*) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
*  LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|
     kLPSPI_RxDataReadyFlag);
*
```

Parameters

base	LPSPI peripheral address.
statusFlags	The status flag used from type _lpspi_flags.

$<$ The status flags are cleared by writing 1 (w1c).

13.2.8.15 static void LPSPI_EnableInterrupts (LPSPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
*  LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable |
     kLPSPI_RxInterruptEnable );
*
```

Parameters

base	LPSPI peripheral address.
mask	The interrupt mask; Use the enum _lpspi_interrupt_enable.

13.2.8.16 static void LPSPI_DisableInterrupts (LPSPI_Type ∗ *base*, uint32_t *mask*) [inline], [static]

```
* LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable |
    kLPSPI_RxInterruptEnable );
*
```

Parameters

base	LPSPI peripheral address.
mask	The interrupt mask; Use the enum _lpspi_interrupt_enable.

13.2.8.17 static void LPSPI_EnableDMA (LPSPI_Type ∗ *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
* LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable |
    kLPSPI_RxDmaEnable);
*
```

Parameters

base	LPSPI peripheral address.
mask	The interrupt mask; Use the enum _lpspi_dma_enable.

13.2.8.18 static void LPSPI_DisableDMA (LPSPI_Type ∗ *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
* SPI_DisableDMA(base, kLPSPI_TxDmaEnable |
    kLPSPI_RxDmaEnable);
*
```

Parameters

base	LPSPI peripheral address.
mask	The interrupt mask; Use the enum _lpspi_dma_enable.

13.2.8.19 static uint32_t LPSPI_GetTxRegisterAddress (LPSPI_Type * *base*) [inline], [static]

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

base	LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

13.2.8.20 static uint32_t LPSPI_GetRxRegisterAddress (LPSPI_Type * *base*) [inline], [static]

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

base	LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

13.2.8.21 bool LPSPI_CheckTransferArgument (LPSPI_Type * *base*, lpspi_transfer_t * *transfer*, bool *isEdma*)

Parameters

base	LPSPI peripheral address.
transfer	the transfer struct to be used.
isEdma	True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

13.2.8.22 static void LPSPI_SetMasterSlaveMode (LPSPI_Type ∗ *base*, lpspi_master_slave_mode_t *mode*) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

base	LPSPI peripheral address.
mode	Mode setting (master or slave) of type lpspi_master_slave_mode_t.

13.2.8.23 static void LPSPI_SelectTransferPCS (LPSPI_Type ∗ *base*, lpspi_which_pcs_t *select*) [inline], [static]

Parameters

base	LPSPI peripheral address.
select	LPSPI Peripheral Chip Select (PCS) configuration.

13.2.8.24 static void LPSPI_SetPCSContinous (LPSPI_Type ∗ *base*, bool *IsContinous*) [inline], [static]

Note

In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

base	LPSPI peripheral address.
IsContinous	True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

13.2.8.25 static bool LPSPI_IsMaster (LPSPI_Type ∗ *base*) [inline], [static]

Parameters

base	LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

13.2.8.26 static void LPSPI_FlushFifo (LPSPI_Type ∗ *base*, bool *flushTxFifo*, bool *flushRxFifo*) [inline], [static]

Parameters

base	LPSPI peripheral address.
flushTxFifo	Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
flushRxFifo	Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

13.2.8.27 static void LPSPI_SetFifoWatermarks (LPSPI_Type ∗ *base*, uint32_t *txWater*, uint32_t *rxWater*) [inline], [static]

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

base	LPSPI peripheral address.

txWater	The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
rxWater	The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

13.2.8.28 static void LPSPI_SetAllPcsPolarity (LPSPI_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
*  LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow |
      kLPSPI_Pcs1ActiveLow);
*
```

Parameters

base	LPSPI peripheral address.
mask	The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

13.2.8.29 static void LPSPI_SetFrameSize (LPSPI_Type ∗ *base*, uint32_t *frameSize*) [inline], [static]

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

base	LPSPI peripheral address.
frameSize	The frame size in number of bits.

13.2.8.30 uint32_t LPSPI_MasterSetBaudRate (LPSPI_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*, uint32_t * *tcrPrescaleValue*)

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

base	LPSPI peripheral address.
baudRate_Bps	The desired baud rate in bits per second.
srcClock_Hz	Module source input clock in Hertz.
tcrPrescale-Value	The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

13.2.8.31 void LPSPI_MasterSetDelayScaler (LPSPI_Type * *base*, uint32_t *scaler*, lpspi_delay_type_t *whichDelay*)

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

base	LPSPI peripheral address.
scaler	The 8-bit delay value 0x00 to 0xFF (255).
whichDelay	The desired delay to configure, must be of type lpspi_delay_type_t.

13.2.8.32 uint32_t LPSPI_MasterSetDelayTimes (LPSPI_Type ∗ *base*, uint32_t *delayTimeInNanoSec*, lpspi_delay_type_t *whichDelay*, uint32_t *srcClock_Hz*)

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the delayTime = LPSPI_clockSource / (PRESCALE ∗ Delay_scaler).

Parameters

base	LPSPI peripheral address.
delayTimeIn-NanoSec	The desired delay value in nano-seconds.
whichDelay	The desired delay to configuration, which must be of type lpspi_delay_type_t.
srcClock_Hz	Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

13.2.8.33 static void LPSPI_WriteData (LPSPI_Type ∗ *base*, uint32_t *data*) [inline], [static]

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

base	LPSPI peripheral address.
data	The data word to be sent.

13.2.8.34 static uint32_t LPSPI_ReadData (LPSPI_Type * *base*) [inline], [static]

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

base	LPSPI peripheral address.

Returns

The data read from the data buffer.

13.2.8.35 void LPSPI_SetDummyData (LPSPI_Type ∗ *base,* uint8_t *dummyData*)

Parameters

base	LPSPI peripheral address.
dummyData	Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

13.2.8.36 void LPSPI_MasterTransferCreateHandle (LPSPI_Type ∗ *base,* lpspi_master_handle_t ∗ *handle,* lpspi_master_transfer_callback_t *callback,* void ∗ *userData*)

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

Parameters

base	LPSPI peripheral address.
handle	LPSPI handle pointer to lpspi_master_handle_t.
callback	DSPI callback.
userData	callback function parameter.

13.2.8.37 status_t LPSPI_MasterTransferBlocking (LPSPI_Type ∗ *base,* lpspi_transfer_t ∗ *transfer*)

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

base	LPSPI peripheral address.
transfer	pointer to lpspi_transfer_t structure.

Returns

status of status_t.

13.2.8.38 status_t LPSPI_MasterTransferNonBlocking (LPSPI_Type * *base*, lpspi_master_handle_t * *handle*, lpspi_transfer_t * *transfer*)

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_master_handle_t structure which stores the transfer state.
transfer	pointer to lpspi_transfer_t structure.

Returns

status of status_t.

13.2.8.39 status_t LPSPI_MasterTransferGetCount (LPSPI_Type * *base*, lpspi_master_handle_t * *handle*, size_t * *count*)

This function gets the master transfer remaining bytes.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_master_handle_t structure which stores the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

13.2.8.40 void LPSPI_MasterTransferAbort (LPSPI_Type ∗ *base*, lpspi_master_handle_t ∗ *handle*)

This function aborts a transfer which uses an interrupt method.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_master_handle_t structure which stores the transfer state.

13.2.8.41 void LPSPI_MasterTransferHandleIRQ (LPSPI_Type ∗ *base*, lpspi_master_handle_t ∗ *handle*)

This function processes the LPSPI transmit and receive IRQ.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_master_handle_t structure which stores the transfer state.

13.2.8.42 void LPSPI_SlaveTransferCreateHandle (LPSPI_Type ∗ *base*, lpspi_slave_handle_t ∗ *handle*, lpspi_slave_transfer_callback_t *callback*, void ∗ *userData*)

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

Parameters

base	LPSPI peripheral address.
handle	LPSPI handle pointer to lpspi_slave_handle_t.
callback	DSPI callback.
userData	callback function parameter.

13.2.8.43 status_t LPSPI_SlaveTransferNonBlocking (LPSPI_Type ∗ *base*, lpspi_slave_handle_t ∗ *handle*, lpspi_transfer_t ∗ *transfer*)

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_slave_handle_t structure which stores the transfer state.
transfer	pointer to lpspi_transfer_t structure.

Returns

status of status_t.

13.2.8.44 status_t LPSPI_SlaveTransferGetCount (LPSPI_Type ∗ *base*, lpspi_slave_handle_t ∗ *handle*, size_t ∗ *count*)

This function gets the slave transfer remaining bytes.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_slave_handle_t structure which stores the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

13.2.8.45 void LPSPI_SlaveTransferAbort (LPSPI_Type ∗ *base*, lpspi_slave_handle_t ∗ *handle*)

This function aborts a transfer which uses an interrupt method.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_slave_handle_t structure which stores the transfer state.

13.2.8.46 void LPSPI_SlaveTransferHandleIRQ (LPSPI_Type * *base*, lpspi_slave_handle_t * *handle*)

This function processes the LPSPI transmit and receives an IRQ.

Parameters

base	LPSPI peripheral address.
handle	pointer to lpspi_slave_handle_t structure which stores the transfer state.

13.2.9 Variable Documentation

13.2.9.1 volatile uint8_t g_lpspiDummyData[ ]

Chapter 14
LPTMR: Low-Power Timer

14.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

14.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

14.2.1 Initialization and deinitialization

The function LPTMR_Init() initializes the LPTMR with specified configurations. The function LPTMR_-GetDefaultConfig() gets the default configurations. The initialization function configures the LPTMR for a timer or a pulse counter mode mode. It also sets up the LPTMR's free running mode operation and a clock source.

The function LPTMR_DeInit() disables the LPTMR module and gates the module clock.

14.2.2 Timer period Operations

The function LPTMR_SetTimerPeriod() sets the timer period in units of count. Timers counts from 0 to the count value set here.

The function LPTMR_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value ranging from 0 to a timer period.

The timer period operation function takes the count value in ticks. Call the utility macros provided in the fsl_common.h file to convert to microseconds or milliseconds.

14.2.3 Start and Stop timer operations

The function LPTMR_StartTimer() starts the timer counting. After calling this function, the timer counts up to the counter value set earlier by using the LPTMR_SetPeriod() function. Each time the timer reaches the count value and increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

The function LPTMR_StopTimer() stops the timer counting and resets the timer's counter register.

14.2.4 Status

Provides functions to get and clear the LPTMR status.

14.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

14.3 Typical use case

14.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lptmr

Data Structures

- struct lptmr_config_t
  LPTMR config structure. More...

Enumerations

- enum lptmr_pin_select_t {
  kLPTMR_PinSelectInput_0 = 0x0U,
  kLPTMR_PinSelectInput_1 = 0x1U,
  kLPTMR_PinSelectInput_2 = 0x2U,
  kLPTMR_PinSelectInput_3 = 0x3U }
  LPTMR pin selection used in pulse counter mode.
- enum lptmr_pin_polarity_t {
  kLPTMR_PinPolarityActiveHigh = 0x0U,
  kLPTMR_PinPolarityActiveLow = 0x1U }
  LPTMR pin polarity used in pulse counter mode.
- enum lptmr_timer_mode_t {
  kLPTMR_TimerModeTimeCounter = 0x0U,
  kLPTMR_TimerModePulseCounter = 0x1U }
  LPTMR timer mode selection.
- enum lptmr_prescaler_glitch_value_t {

kLPTMR_Prescale_Glitch_0 = 0x0U,
kLPTMR_Prescale_Glitch_1 = 0x1U,
kLPTMR_Prescale_Glitch_2 = 0x2U,
kLPTMR_Prescale_Glitch_3 = 0x3U,
kLPTMR_Prescale_Glitch_4 = 0x4U,
kLPTMR_Prescale_Glitch_5 = 0x5U,
kLPTMR_Prescale_Glitch_6 = 0x6U,
kLPTMR_Prescale_Glitch_7 = 0x7U,
kLPTMR_Prescale_Glitch_8 = 0x8U,
kLPTMR_Prescale_Glitch_9 = 0x9U,
kLPTMR_Prescale_Glitch_10 = 0xAU,
kLPTMR_Prescale_Glitch_11 = 0xBU,
kLPTMR_Prescale_Glitch_12 = 0xCU,
kLPTMR_Prescale_Glitch_13 = 0xDU,
kLPTMR_Prescale_Glitch_14 = 0xEU,
kLPTMR_Prescale_Glitch_15 = 0xFU }
  LPTMR prescaler/glitch filter values.
- enum lptmr_prescaler_clock_select_t {
kLPTMR_PrescalerClock_0 = 0x0U,
kLPTMR_PrescalerClock_1 = 0x1U,
kLPTMR_PrescalerClock_2 = 0x2U,
kLPTMR_PrescalerClock_3 = 0x3U }
  LPTMR prescaler/glitch filter clock select.
- enum lptmr_interrupt_enable_t { kLPTMR_TimerInterruptEnable = LPTMR_CSR_TIE_MASK }
  List of the LPTMR interrupts.
- enum lptmr_status_flags_t { kLPTMR_TimerCompareFlag = LPTMR_CSR_TCF_MASK }
  List of the LPTMR status flags.

Functions

- static void LPTMR_EnableTimerDMA (LPTMR_Type ∗base, bool enable)
  Enable or disable timer DMA request.

Driver version

- #define FSL_LPTMR_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  Version 2.1.1.

Initialization and deinitialization

- void LPTMR_Init (LPTMR_Type ∗base, const lptmr_config_t ∗config)
  Ungates the LPTMR clock and configures the peripheral for a basic operation.
- void LPTMR_Deinit (LPTMR_Type ∗base)
  Gates the LPTMR clock.
- void LPTMR_GetDefaultConfig (lptmr_config_t ∗config)
  Fills in the LPTMR configuration structure with default settings.

Interrupt Interface

- static void LPTMR_EnableInterrupts (LPTMR_Type *base, uint32_t mask)
  Enables the selected LPTMR interrupts.
- static void LPTMR_DisableInterrupts (LPTMR_Type *base, uint32_t mask)
  Disables the selected LPTMR interrupts.
- static uint32_t LPTMR_GetEnabledInterrupts (LPTMR_Type *base)
  Gets the enabled LPTMR interrupts.

Status Interface

- static uint32_t LPTMR_GetStatusFlags (LPTMR_Type *base)
  Gets the LPTMR status flags.
- static void LPTMR_ClearStatusFlags (LPTMR_Type *base, uint32_t mask)
  Clears the LPTMR status flags.

Read and write the timer period

- static void LPTMR_SetTimerPeriod (LPTMR_Type *base, uint32_t ticks)
  Sets the timer period in units of count.
- static uint32_t LPTMR_GetCurrentTimerCount (LPTMR_Type *base)
  Reads the current timer counting value.

Timer Start and Stop

- static void LPTMR_StartTimer (LPTMR_Type *base)
  Starts the timer.
- static void LPTMR_StopTimer (LPTMR_Type *base)
  Stops the timer.

14.4 Data Structure Documentation

14.4.1 struct lptmr_config_t

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

The configuration struct can be made constant so it resides in flash.

Data Fields

- lptmr_timer_mode_t timerMode
  Time counter mode or pulse counter mode.
- lptmr_pin_select_t pinSelect
  LPTMR pulse input pin select; used only in pulse counter mode.
- lptmr_pin_polarity_t pinPolarity
  LPTMR pulse input pin polarity; used only in pulse counter mode.
- bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set.

- bool bypassPrescaler
  True: bypass prescaler; false: use clock from prescaler.
- lptmr_prescaler_clock_select_t prescalerClockSource
  LPTMR clock source.
- lptmr_prescaler_glitch_value_t value
  Prescaler or glitch filter value.

14.5 Enumeration Type Documentation

14.5.1 enum lptmr_pin_select_t

Enumerator

kLPTMR_PinSelectInput_0 Pulse counter input 0 is selected.
kLPTMR_PinSelectInput_1 Pulse counter input 1 is selected.
kLPTMR_PinSelectInput_2 Pulse counter input 2 is selected.
kLPTMR_PinSelectInput_3 Pulse counter input 3 is selected.

14.5.2 enum lptmr_pin_polarity_t

Enumerator

kLPTMR_PinPolarityActiveHigh Pulse Counter input source is active-high.
kLPTMR_PinPolarityActiveLow Pulse Counter input source is active-low.

14.5.3 enum lptmr_timer_mode_t

Enumerator

kLPTMR_TimerModeTimeCounter Time Counter mode.
kLPTMR_TimerModePulseCounter Pulse Counter mode.

14.5.4 enum lptmr_prescaler_glitch_value_t

Enumerator

kLPTMR_Prescale_Glitch_0 Prescaler divide 2, glitch filter does not support this setting.
kLPTMR_Prescale_Glitch_1 Prescaler divide 4, glitch filter 2.
kLPTMR_Prescale_Glitch_2 Prescaler divide 8, glitch filter 4.
kLPTMR_Prescale_Glitch_3 Prescaler divide 16, glitch filter 8.
kLPTMR_Prescale_Glitch_4 Prescaler divide 32, glitch filter 16.

kLPTMR_Prescale_Glitch_5 Prescaler divide 64, glitch filter 32.
kLPTMR_Prescale_Glitch_6 Prescaler divide 128, glitch filter 64.
kLPTMR_Prescale_Glitch_7 Prescaler divide 256, glitch filter 128.
kLPTMR_Prescale_Glitch_8 Prescaler divide 512, glitch filter 256.
kLPTMR_Prescale_Glitch_9 Prescaler divide 1024, glitch filter 512.
kLPTMR_Prescale_Glitch_10 Prescaler divide 2048 glitch filter 1024.
kLPTMR_Prescale_Glitch_11 Prescaler divide 4096, glitch filter 2048.
kLPTMR_Prescale_Glitch_12 Prescaler divide 8192, glitch filter 4096.
kLPTMR_Prescale_Glitch_13 Prescaler divide 16384, glitch filter 8192.
kLPTMR_Prescale_Glitch_14 Prescaler divide 32768, glitch filter 16384.
kLPTMR_Prescale_Glitch_15 Prescaler divide 65536, glitch filter 32768.

14.5.5 enum lptmr_prescaler_clock_select_t

Note

Clock connections are SoC-specific

Enumerator

kLPTMR_PrescalerClock_0 Prescaler/glitch filter clock 0 selected.
kLPTMR_PrescalerClock_1 Prescaler/glitch filter clock 1 selected.
kLPTMR_PrescalerClock_2 Prescaler/glitch filter clock 2 selected.
kLPTMR_PrescalerClock_3 Prescaler/glitch filter clock 3 selected.

14.5.6 enum lptmr_interrupt_enable_t

Enumerator

kLPTMR_TimerInterruptEnable Timer interrupt enable.

14.5.7 enum lptmr_status_flags_t

Enumerator

kLPTMR_TimerCompareFlag Timer compare flag.

14.6 Function Documentation

14.6.1 void LPTMR_Init (LPTMR_Type ∗ *base,* const lptmr_config_t ∗ *config*)

Note

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

Parameters

base	LPTMR peripheral base address
config	A pointer to the LPTMR configuration structure.

14.6.2 void LPTMR_Deinit (LPTMR_Type * *base*)

Parameters

base	LPTMR peripheral base address

14.6.3 void LPTMR_GetDefaultConfig (lptmr_config_t * *config*)

The default values are as follows.

```
*    config->timerMode = kLPTMR_TimerModeTimeCounter;
*    config->pinSelect = kLPTMR_PinSelectInput_0;
*    config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
*    config->enableFreeRunning = false;
*    config->bypassPrescaler = true;
*    config->prescalerClockSource = kLPTMR_PrescalerClock_1;
*    config->value = kLPTMR_Prescale_Glitch_0;
*
```

Parameters

config	A pointer to the LPTMR configuration structure.

14.6.4 static void LPTMR_EnableInterrupts (LPTMR_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

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

14.6.5 static void LPTMR_DisableInterrupts (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LPTMR peripheral base address
mask	The interrupts to disable. This is a logical OR of members of the enumeration lptmr-_interrupt_enable_t.

14.6.6 static uint32_t LPTMR_GetEnabledInterrupts (LPTMR_Type * *base*) [inline], [static]

Parameters

base	LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration lptmr_interrupt_-enable_t

14.6.7 static void LPTMR_EnableTimerDMA (LPTMR_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	base LPTMR peripheral base address
enable	Switcher of timer DMA feature. "true" means to enable, "false" means to disable.

14.6.8 static uint32_t LPTMR_GetStatusFlags (LPTMR_Type * *base*) [inline], [static]

Parameters

base	LPTMR peripheral base address

Returns

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

14.6.9 static void LPTMR_ClearStatusFlags (LPTMR_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	LPTMR peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration lptmr_-status_flags_t.

14.6.10 static void LPTMR_SetTimerPeriod (LPTMR_Type ∗ *base,* uint32_t *ticks*) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note

1. The TCF flag is set with the CNR equals the count provided here and then increments.
2. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

base	LPTMR peripheral base address
ticks	A timer period in units of ticks, which should be equal or greater than 1.

14.6.11 static uint32_t LPTMR_GetCurrentTimerCount (LPTMR_Type ∗ *base*) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

base	LPTMR peripheral base address

Returns

The current counter value in ticks

14.6.12 static void LPTMR_StartTimer (LPTMR_Type * *base*) [inline], [static]

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

base	LPTMR peripheral base address

14.6.13 static void LPTMR_StopTimer (LPTMR_Type * *base*) [inline], [static]

This function stops the timer and resets the timer's counter register.

Parameters

base	LPTMR peripheral base address

Chapter 15

LPUART: Low Power Universal Asynchronous Receiver/-Transmitter Driver

15.1 Overview

Modules

- LPUART Driver

15.2 LPUART Driver

15.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

15.2.2 Typical use case

15.2.2.1 LPUART Operation

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

Data Structures

- struct lpuart_config_t
  LPUART configuration structure. More...
- struct lpuart_transfer_t
  LPUART transfer structure. More...
- struct lpuart_handle_t
  LPUART handle structure. More...

Macros

- #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef void(∗ lpuart_transfer_callback_t)(LPUART_Type ∗base, lpuart_handle_t ∗handle, status-_t status, void ∗userData)
  LPUART transfer callback function.

Enumerations

- enum {
kStatus_LPUART_TxBusy = MAKE_STATUS(kStatusGroup_LPUART, 0),
kStatus_LPUART_RxBusy = MAKE_STATUS(kStatusGroup_LPUART, 1),
kStatus_LPUART_TxIdle = MAKE_STATUS(kStatusGroup_LPUART, 2),
kStatus_LPUART_RxIdle = MAKE_STATUS(kStatusGroup_LPUART, 3),
kStatus_LPUART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 4),
kStatus_LPUART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 5),
kStatus_LPUART_FlagCannotClearManually = MAKE_STATUS(kStatusGroup_LPUART, 6),
kStatus_LPUART_Error = MAKE_STATUS(kStatusGroup_LPUART, 7),
kStatus_LPUART_RxRingBufferOverrun,
kStatus_LPUART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_LPUART, 9),
kStatus_LPUART_NoiseError = MAKE_STATUS(kStatusGroup_LPUART, 10),
kStatus_LPUART_FramingError = MAKE_STATUS(kStatusGroup_LPUART, 11),
kStatus_LPUART_ParityError = MAKE_STATUS(kStatusGroup_LPUART, 12),
kStatus_LPUART_BaudrateNotSupport,
kStatus_LPUART_IdleLineDetected = MAKE_STATUS(kStatusGroup_LPUART, 14),
kStatus_LPUART_Timeout = MAKE_STATUS(kStatusGroup_LPUART, 15) }
Error codes for the LPUART driver.
- enum lpuart_parity_mode_t {
kLPUART_ParityDisabled = 0x0U,
kLPUART_ParityEven = 0x2U,
kLPUART_ParityOdd = 0x3U }
LPUART parity mode.
- enum lpuart_data_bits_t {
kLPUART_EightDataBits = 0x0U,
kLPUART_SevenDataBits = 0x1U }
LPUART data bits count.
- enum lpuart_stop_bit_count_t {
kLPUART_OneStopBit = 0U,
kLPUART_TwoStopBit = 1U }
LPUART stop bit count.
- enum lpuart_transmit_cts_source_t {
kLPUART_CtsSourcePin = 0U,
kLPUART_CtsSourceMatchResult = 1U }
LPUART transmit CTS source.
- enum lpuart_transmit_cts_config_t {
kLPUART_CtsSampleAtStart = 0U,
kLPUART_CtsSampleAtIdle = 1U }
LPUART transmit CTS configure.
- enum lpuart_idle_type_select_t {
kLPUART_IdleTypeStartBit = 0U,
kLPUART_IdleTypeStopBit = 1U }
LPUART idle flag type defines when the receiver starts counting.
- enum lpuart_idle_config_t {

kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
LPUART idle detected configuration.

- enum _lpuart_interrupt_enable {
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIE_MASK >> 8U),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8U),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK),
kLPUART_Match1InterruptEnable = (LPUART_CTRL_MA1IE_MASK),
kLPUART_Match2InterruptEnable = (LPUART_CTRL_MA2IE_MASK),
kLPUART_TxFifoOverflowInterruptEnable = (LPUART_FIFO_TXOFE_MASK),
kLPUART_RxFifoUnderflowInterruptEnable = (LPUART_FIFO_RXUFE_MASK) }
LPUART interrupt configuration structure, default settings all disabled.
- enum _lpuart_flags {
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag = (LPUART_STAT_RDRF_MASK),
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag,
kLPUART_DataMatch2Flag,
kLPUART_TxFifoEmptyFlag,
kLPUART_RxFifoEmptyFlag,
kLPUART_TxFifoOverflowFlag,
kLPUART_RxFifoUnderflowFlag }
LPUART status flags.

Driver version

- #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 5, 3))
  LPUART driver version.

Software Reset

- static void LPUART_SoftwareReset (LPUART_Type ∗base)
  Resets the LPUART using software.

Initialization and deinitialization

- status_t LPUART_Init (LPUART_Type ∗base, const lpuart_config_t ∗config, uint32_t srcClock_-Hz)
  Initializes an LPUART instance with the user configuration structure and the peripheral clock.
- void LPUART_Deinit (LPUART_Type ∗base)
  Deinitializes a LPUART instance.
- void LPUART_GetDefaultConfig (lpuart_config_t ∗config)
  Gets the default configuration structure.

Module configuration

- status_t LPUART_SetBaudRate (LPUART_Type ∗base, uint32_t baudRate_Bps, uint32_t src-Clock_Hz)
  Sets the LPUART instance baudrate.
- void LPUART_Enable9bitMode (LPUART_Type ∗base, bool enable)
  Enable 9-bit data mode for LPUART.
- static void LPUART_SetMatchAddress (LPUART_Type ∗base, uint16_t address1, uint16_t address2)
  Set the LPUART address.
- static void LPUART_EnableMatchAddress (LPUART_Type ∗base, bool match1, bool match2)
  Enable the LPUART match address feature.
- static void LPUART_SetRxFifoWatermark (LPUART_Type ∗base, uint8_t water)
  Sets the rx FIFO watermark.
- static void LPUART_SetTxFifoWatermark (LPUART_Type ∗base, uint8_t water)
  Sets the tx FIFO watermark.

Status

- uint32_t LPUART_GetStatusFlags (LPUART_Type ∗base)
  Gets LPUART status flags.
- status_t LPUART_ClearStatusFlags (LPUART_Type ∗base, uint32_t mask)
  Clears status flags with a provided mask.

Interrupts

- void LPUART_EnableInterrupts (LPUART_Type ∗base, uint32_t mask)
  Enables LPUART interrupts according to a provided mask.
- void LPUART_DisableInterrupts (LPUART_Type ∗base, uint32_t mask)
  Disables LPUART interrupts according to a provided mask.
- uint32_t LPUART_GetEnabledInterrupts (LPUART_Type ∗base)
  Gets enabled LPUART interrupts.

Bus Operations

- uint32_t LPUART_GetInstance (LPUART_Type ∗base)
  Get the LPUART instance from peripheral base address.
- static void LPUART_EnableTx (LPUART_Type ∗base, bool enable)
  Enables or disables the LPUART transmitter.
- static void LPUART_EnableRx (LPUART_Type ∗base, bool enable)
  Enables or disables the LPUART receiver.
- static void LPUART_WriteByte (LPUART_Type ∗base, uint8_t data)
  Writes to the transmitter register.
- static uint8_t LPUART_ReadByte (LPUART_Type ∗base)
  Reads the receiver register.
- static uint8_t LPUART_GetRxFifoCount (LPUART_Type ∗base)
  Gets the rx FIFO data count.
- static uint8_t LPUART_GetTxFifoCount (LPUART_Type ∗base)
  Gets the tx FIFO data count.
- void LPUART_SendAddress (LPUART_Type ∗base, uint8_t address)
  Transmit an address frame in 9-bit data mode.
- status_t LPUART_WriteBlocking (LPUART_Type ∗base, const uint8_t ∗data, size_t length)
  Writes to the transmitter register using a blocking method.
- status_t LPUART_ReadBlocking (LPUART_Type ∗base, uint8_t ∗data, size_t length)
  Reads the receiver data register using a blocking method.

Transactional

- void LPUART_TransferCreateHandle (LPUART_Type ∗base, lpuart_handle_t ∗handle, lpuart_transfer_callback_t callback, void ∗userData)
  Initializes the LPUART handle.
- status_t LPUART_TransferSendNonBlocking (LPUART_Type ∗base, lpuart_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Transmits a buffer of data using the interrupt method.
- void LPUART_TransferStartRingBuffer (LPUART_Type ∗base, lpuart_handle_t ∗handle, uint8_t ∗ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void LPUART_TransferStopRingBuffer (LPUART_Type ∗base, lpuart_handle_t ∗handle)
  Aborts the background transfer and uninstalls the ring buffer.
- size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type ∗base, lpuart_handle_t ∗handle)
  Get the length of received data in RX ring buffer.

- void LPUART_TransferAbortSend (LPUART_Type ∗base, lpuart_handle_t ∗handle)
  Aborts the interrupt-driven data transmit.
- status_t LPUART_TransferGetSendCount (LPUART_Type ∗base, lpuart_handle_t ∗handle, uint32-_t ∗count)
  Gets the number of bytes that have been sent out to bus.
- status_t LPUART_TransferReceiveNonBlocking (LPUART_Type ∗base, lpuart_handle_t ∗handle, lpuart_transfer_t ∗xfer, size_t ∗receivedBytes)
  Receives a buffer of data using the interrupt method.
- void LPUART_TransferAbortReceive (LPUART_Type ∗base, lpuart_handle_t ∗handle)
  Aborts the interrupt-driven data receiving.
- status_t LPUART_TransferGetReceiveCount (LPUART_Type ∗base, lpuart_handle_t ∗handle, uint32_t ∗count)
  Gets the number of bytes that have been received.
- void LPUART_TransferHandleIRQ (LPUART_Type ∗base, void ∗irqHandle)
  LPUART IRQ handle function.
- void LPUART_TransferHandleErrorIRQ (LPUART_Type ∗base, void ∗irqHandle)
  LPUART Error IRQ handle function.

15.2.3 Data Structure Documentation

15.2.3.1 struct lpuart_config_t

Data Fields

- uint32_t baudRate_Bps
  LPUART baud rate.
- lpuart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- lpuart_data_bits_t dataBitsCount
  Data bits count, eight (default), seven.
- bool isMsb
  Data bits order, LSB (default), MSB.
- lpuart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- uint8_t txFifoWatermark
  TX FIFO watermark.
- uint8_t rxFifoWatermark
  RX FIFO watermark.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- lpuart_transmit_cts_source_t txCtsSource
  TX CTS source.
- lpuart_transmit_cts_config_t txCtsConfig
  TX CTS configure.
- lpuart_idle_type_select_t rxIdleType
  RX IDLE type.
- lpuart_idle_config_t rxIdleConfig

RX IDLE configuration.
- bool enableTx
  Enable TX.
- bool enableRx
  Enable RX.

Field Documentation

(1) lpuart_idle_type_select_t lpuart_config_t::rxIdleType

(2) lpuart_idle_config_t lpuart_config_t::rxIdleConfig

15.2.3.2 struct lpuart_transfer_t

Data Fields

- size_t dataSize
  The byte count to be transfer.
- uint8_t * data
  The buffer of data to be transfer.
- uint8_t * rxData
  The buffer to receive data.
- const uint8_t * txData
  The buffer of data to be sent.

Field Documentation

(1) uint8_t* lpuart_transfer_t::data

(2) uint8_t* lpuart_transfer_t::rxData

(3) const uint8_t* lpuart_transfer_t::txData

(4) size_t lpuart_transfer_t::dataSize

15.2.3.3 struct _lpuart_handle

Data Fields

- const uint8_t *volatile txData
  Address of remaining data to send.
- volatile size_t txDataSize
  Size of the remaining data to send.
- size_t txDataSizeAll
  Size of the data to send out.
- uint8_t *volatile rxData
  Address of remaining data to receive.
- volatile size_t rxDataSize
  Size of the remaining data to receive.
- size_t rxDataSizeAll
  Size of the data to receive.

- uint8_t ∗ rxRingBuffer
  Start address of the receiver ring buffer.
- size_t rxRingBufferSize
  Size of the ring buffer.
- volatile uint16_t rxRingBufferHead
  Index for the driver to store received data into ring buffer.
- volatile uint16_t rxRingBufferTail
  Index for the user to get data from the ring buffer.
- lpuart_transfer_callback_t callback
  Callback function.
- void ∗ userData
  LPUART callback function parameter.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.
- bool isSevenDataBits
  Seven data bits flag.

Field Documentation

(1) const uint8_t∗ volatile lpuart_handle_t::txData

(2) volatile size_t lpuart_handle_t::txDataSize

(3) size_t lpuart_handle_t::txDataSizeAll

(4) uint8_t∗ volatile lpuart_handle_t::rxData

(5) volatile size_t lpuart_handle_t::rxDataSize

(6) size_t lpuart_handle_t::rxDataSizeAll

(7) uint8_t∗ lpuart_handle_t::rxRingBuffer

(8) size_t lpuart_handle_t::rxRingBufferSize

(9) volatile uint16_t lpuart_handle_t::rxRingBufferHead

(10) volatile uint16_t lpuart_handle_t::rxRingBufferTail

(11) lpuart_transfer_callback_t lpuart_handle_t::callback

(12) void∗ lpuart_handle_t::userData

(13) volatile uint8_t lpuart_handle_t::txState

(14) volatile uint8_t lpuart_handle_t::rxState

(15) bool lpuart_handle_t::isSevenDataBits

15.2.4 Macro Definition Documentation

15.2.4.1 #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 5, 3))

15.2.4.2 #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/

15.2.5 Typedef Documentation

15.2.5.1 typedef void(∗ lpuart_transfer_callback_t)(LPUART_Type ∗base, lpuart_handle_t ∗handle, status_t status, void ∗userData)

15.2.6 Enumeration Type Documentation

15.2.6.1 anonymous enum

Enumerator

kStatus_LPUART_TxBusy TX busy.
kStatus_LPUART_RxBusy RX busy.
kStatus_LPUART_TxIdle LPUART transmitter is idle.
kStatus_LPUART_RxIdle LPUART receiver is idle.
kStatus_LPUART_TxWatermarkTooLarge TX FIFO watermark too large.
kStatus_LPUART_RxWatermarkTooLarge RX FIFO watermark too large.
kStatus_LPUART_FlagCannotClearManually Some flag can't manually clear.
kStatus_LPUART_Error Error happens on LPUART.
kStatus_LPUART_RxRingBufferOverrun LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverrun LPUART RX receiver overrun.
kStatus_LPUART_NoiseError LPUART noise error.
kStatus_LPUART_FramingError LPUART framing error.
kStatus_LPUART_ParityError LPUART parity error.
kStatus_LPUART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_LPUART_IdleLineDetected IDLE flag.
kStatus_LPUART_Timeout LPUART times out.

15.2.6.2 enum lpuart_parity_mode_t

Enumerator

kLPUART_ParityDisabled Parity disabled.
kLPUART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kLPUART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

15.2.6.3 enum lpuart_data_bits_t

Enumerator

kLPUART_EightDataBits Eight data bit.
kLPUART_SevenDataBits Seven data bit.

15.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.
kLPUART_TwoStopBit Two stop bits.

15.2.6.5 enum lpuart_transmit_cts_source_t

Enumerator

kLPUART_CtsSourcePin CTS resource is the LPUART_CTS pin.
kLPUART_CtsSourceMatchResult CTS resource is the match result.

15.2.6.6 enum lpuart_transmit_cts_config_t

Enumerator

kLPUART_CtsSampleAtStart CTS input is sampled at the start of each character.
kLPUART_CtsSampleAtIdle CTS input is sampled when the transmitter is idle.

15.2.6.7 enum lpuart_idle_type_select_t

Enumerator

kLPUART_IdleTypeStartBit Start counting after a valid start bit.
kLPUART_IdleTypeStopBit Start counting after a stop bit.

15.2.6.8 enum lpuart_idle_config_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

kLPUART_IdleCharacter1 the number of idle characters.

kLPUART_IdleCharacter2 the number of idle characters.
kLPUART_IdleCharacter4 the number of idle characters.
kLPUART_IdleCharacter8 the number of idle characters.
kLPUART_IdleCharacter16 the number of idle characters.
kLPUART_IdleCharacter32 the number of idle characters.
kLPUART_IdleCharacter64 the number of idle characters.
kLPUART_IdleCharacter128 the number of idle characters.

15.2.6.9 enum _lpuart_interrupt_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

kLPUART_LinBreakInterruptEnable LIN break detect. bit 7
kLPUART_RxActiveEdgeInterruptEnable Receive Active Edge. bit 6
kLPUART_TxDataRegEmptyInterruptEnable Transmit data register empty. bit 23
kLPUART_TransmissionCompleteInterruptEnable Transmission complete. bit 22
kLPUART_RxDataRegFullInterruptEnable Receiver data register full. bit 21
kLPUART_IdleLineInterruptEnable Idle line. bit 20
kLPUART_RxOverrunInterruptEnable Receiver Overrun. bit 27
kLPUART_NoiseErrorInterruptEnable Noise error flag. bit 26
kLPUART_FramingErrorInterruptEnable Framing error flag. bit 25
kLPUART_ParityErrorInterruptEnable Parity error flag. bit 24
kLPUART_Match1InterruptEnable Parity error flag. bit 15
kLPUART_Match2InterruptEnable Parity error flag. bit 14
kLPUART_TxFifoOverflowInterruptEnable Transmit FIFO Overflow. bit 9
kLPUART_RxFifoUnderflowInterruptEnable Receive FIFO Underflow. bit 8

15.2.6.10 enum _lpuart_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

kLPUART_TxDataRegEmptyFlag Transmit data register empty flag, sets when transmit buffer is empty. bit 23
kLPUART_TransmissionCompleteFlag Transmission complete flag, sets when transmission activity complete. bit 22
kLPUART_RxDataRegFullFlag Receive data register full flag, sets when the receive data buffer is full. bit 21
kLPUART_IdleLineFlag Idle line detect flag, sets when idle line detected. bit 20
kLPUART_RxOverrunFlag Receive Overrun, sets when new data is received before data is read from receive register. bit 19

kLPUART_NoiseErrorFlag Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18
kLPUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17
kLPUART_ParityErrorFlag If parity enabled, sets upon parity error detection. bit 16
kLPUART_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31
kLPUART_RxActiveEdgeFlag Receive pin active edge interrupt flag, sets when active edge detected. bit 30
kLPUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start. bit 24
kLPUART_DataMatch1Flag The next character to be read from LPUART_DATA matches MA1. bit 15
kLPUART_DataMatch2Flag The next character to be read from LPUART_DATA matches MA2. bit 14
kLPUART_TxFifoEmptyFlag TXEMPT bit, sets if transmit buffer is empty. bit 7
kLPUART_RxFifoEmptyFlag RXEMPT bit, sets if receive buffer is empty. bit 6
kLPUART_TxFifoOverflowFlag TXOF bit, sets if transmit buffer overflow occurred. bit 1
kLPUART_RxFifoUnderflowFlag RXUF bit, sets if receive buffer underflow occurred. bit 0

15.2.7 Function Documentation

15.2.7.1 static void LPUART_SoftwareReset (LPUART_Type ∗ *base*) [inline], [static]

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

base	LPUART peripheral base address.

15.2.7.2 status_t LPUART_Init (LPUART_Type ∗ *base*, const lpuart_config_t ∗ *config*, uint32_t *srcClock_Hz*)

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefault-Config() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
*  lpuart_config_t lpuartConfig;
*  lpuartConfig.baudRate_Bps = 115200U;
*  lpuartConfig.parityMode = kLPUART_ParityDisabled;
*  lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
*  lpuartConfig.isMsb = false;
*  lpuartConfig.stopBitCount = kLPUART_OneStopBit;
*  lpuartConfig.txFifoWatermark = 0;
```

```
*  lpuartConfig.rxFifoWatermark = 1;
*  LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

Parameters

base	LPUART peripheral base address.
config	Pointer to a user-defined configuration structure.
srcClock_Hz	LPUART clock source frequency in HZ.

Return values

kStatus_LPUART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	LPUART initialize succeed

15.2.7.3 void LPUART_Deinit (LPUART_Type ∗ *base*)

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

base	LPUART peripheral base address.

15.2.7.4 void LPUART_GetDefaultConfig (lpuart_config_t ∗ *config*)

This function initializes the LPUART configuration structure to a default value. The default values are-: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

config	Pointer to a configuration structure.

15.2.7.5 status_t LPUART_SetBaudRate (LPUART_Type ∗ *base,* uint32_t *baudRate_Bps,* uint32_t *srcClock_Hz*)

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
*   LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

base	LPUART peripheral base address.
baudRate_Bps	LPUART baudrate to be set.
srcClock_Hz	LPUART clock source frequency in HZ.

Return values

kStatus_LPUART_-BaudrateNotSupport	Baudrate is not supported in the current clock source.
kStatus_Success	Set baudrate succeeded.

15.2.7.6 void LPUART_Enable9bitMode (LPUART_Type ∗ *base,* bool *enable*)

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

base	LPUART peripheral base address.
enable	true to enable, flase to disable.

15.2.7.7 static void LPUART_SetMatchAddress (LPUART_Type ∗ *base,* uint16_t *address1,* uint16_t *address2*) [inline], [static]

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

> Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

base	LPUART peripheral base address.
address1	LPUART slave address1.
address2	LPUART slave address2.

15.2.7.8 static void LPUART_EnableMatchAddress (LPUART_Type ∗ *base,* bool *match1,* bool *match2*) [inline], [static]

Parameters

base	LPUART peripheral base address.
match1	true to enable match address1, false to disable.
match2	true to enable match address2, false to disable.

15.2.7.9 static void LPUART_SetRxFifoWatermark (LPUART_Type ∗ *base,* uint8_t *water*) [inline], [static]

Parameters

base	LPUART peripheral base address.
water	Rx FIFO watermark.

15.2.7.10 static void LPUART_SetTxFifoWatermark (LPUART_Type ∗ *base,* uint8_t *water*) [inline], [static]

Parameters

base	LPUART peripheral base address.
water	Tx FIFO watermark.

15.2.7.11 uint32_t LPUART_GetStatusFlags (LPUART_Type ∗ *base*)

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators _lpuart_flags. To check for a specific status, compare the return value with enumerators in the _lpuart_flags. For example, to check whether the TX is empty:

```
*     if (kLPUART_TxDataRegEmptyFlag &
```

```
      LPUART_GetStatusFlags(LPUART1))
*     {
*         ...
*     }
*
```

Parameters

base	LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

15.2.7.12 status_t LPUART_ClearStatusFlags (LPUART_Type ∗ *base,* uint32_t *mask*)

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

base	LPUART peripheral base address.
mask	the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Returns

0 succeed, others failed.

Return values

kStatus_LPUART_FlagCannotClearManually	The flag can't be cleared by this function but it is cleared automatically by hardware.

kStatus_Success	Status in the mask are cleared.

15.2.7.13 void LPUART_EnableInterrupts (LPUART_Type ∗ *base*, uint32_t *mask*)

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
*     LPUART_EnableInterrupts(LPUART1,
      kLPUART_TxDataRegEmptyInterruptEnable |
      kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	LPUART peripheral base address.
mask	The interrupts to enable. Logical OR of _lpuart_interrupt_enable.

15.2.7.14 void LPUART_DisableInterrupts (LPUART_Type ∗ *base*, uint32_t *mask*)

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     LPUART_DisableInterrupts(LPUART1,
      kLPUART_TxDataRegEmptyInterruptEnable |
      kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	LPUART peripheral base address.
mask	The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

15.2.7.15 uint32_t LPUART_GetEnabledInterrupts (LPUART_Type ∗ *base*)

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
*     if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
*
```

Parameters

base	LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

15.2.7.16 uint32_t LPUART_GetInstance (LPUART_Type * *base*)

Parameters

base	LPUART peripheral base address.

Returns

LPUART instance.

15.2.7.17 static void LPUART_EnableTx (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART transmitter.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

15.2.7.18 static void LPUART_EnableRx (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

15.2.7.19 static void LPUART_WriteByte (LPUART_Type ∗ *base*, uint8_t *data*) [inline], [static]

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

base	LPUART peripheral base address.
data	Data write to the TX register.

15.2.7.20 static uint8_t LPUART_ReadByte (LPUART_Type ∗ *base*) [inline], [static]

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

base	LPUART peripheral base address.

Returns

Data read from data register.

15.2.7.21 static uint8_t LPUART_GetRxFifoCount (LPUART_Type ∗ *base*) [inline], [static]

Parameters

base	LPUART peripheral base address.

Returns

rx FIFO data count.

15.2.7.22 static uint8_t LPUART_GetTxFifoCount (LPUART_Type * *base*) [inline], [static]

Parameters

base	LPUART peripheral base address.

Returns

tx FIFO data count.

15.2.7.23 void LPUART_SendAddress (LPUART_Type * *base*, uint8_t *address*)

Parameters

base	LPUART peripheral base address.
address	LPUART slave address.

15.2.7.24 status_t LPUART_WriteBlocking (LPUART_Type * *base*, const uint8_t * *data*, size_t *length*)

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

base	LPUART peripheral base address.
data	Start address of the data to write.

length	Size of the data to write.

Return values

kStatus_LPUART_-Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully wrote all data.

15.2.7.25 status_t LPUART_ReadBlocking (LPUART_Type ∗ *base,* uint8_t ∗ *data,* size_t *length*)

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

base	LPUART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_LPUART_Rx-HardwareOverrun	Receiver overrun happened while receiving data.
kStatus_LPUART_Noise-Error	Noise error happened while receiving data.
kStatus_LPUART_-FramingError	Framing error happened while receiving data.
kStatus_LPUART_Parity-Error	Parity error happened while receiving data.
kStatus_LPUART_-Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully received all data.

15.2.7.26 void LPUART_TransferCreateHandle (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle,* lpuart_transfer_callback_t *callback,* void ∗ *userData*)

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

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
callback	Callback function.
userData	User data.

15.2.7.27 status_t LPUART_TransferSendNonBlocking (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*, lpuart_transfer_t ∗ *xfer*)

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note

> The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see lpuart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_LPUART_TxBusy	Previous transmission still not finished, data not all written to the TX register.

kStatus_InvalidArgument	Invalid argument.

15.2.7.28 void LPUART_TransferStartRingBuffer (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*, uint8_t ∗ *ringBuffer*, size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
ringBuffer	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	size of the ring buffer.

15.2.7.29 void LPUART_TransferStopRingBuffer (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

15.2.7.30 size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*)

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

15.2.7.31 void LPUART_TransferAbortSend (LPUART_Type * *base*, lpuart_handle_t * *handle*)

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

15.2.7.32 status_t LPUART_TransferGetSendCount (LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferIn-Progress	No send in progress.

kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

15.2.7.33 status_t LPUART_TransferReceiveNonBlocking (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle,* lpuart_transfer_t ∗ *xfer,* size_t ∗ *receivedBytes*)

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see uart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into the transmit queue.
kStatus_LPUART_Rx-Busy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

15.2.7.34 void LPUART_TransferAbortReceive (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle*)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

15.2.7.35 status_t LPUART_TransferGetReceiveCount (LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count*)

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

Parameters

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

Return values

kStatus_NoTransferInProgress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

15.2.7.36 void LPUART_TransferHandleIRQ (LPUART_Type * *base*, void * *irqHandle*)

This function handles the LPUART transmit and receive IRQ request.

Parameters

base	LPUART peripheral base address.
irqHandle	LPUART handle pointer.

15.2.7.37 void LPUART_TransferHandleErrorIRQ (LPUART_Type * *base*, void * *irqHandle*)

This function handles the LPUART error IRQ request.

Parameters

base	LPUART peripheral base address.
irqHandle	LPUART handle pointer.

Chapter 16
MMDVSQ: Memory-Mapped Divide and Square Root

16.1 Overview

The MCUXpresso SDK provides driver for the Memory-Mapped Divide and Square Root (MMDVSQ) module of MCUXpresso SDK devices.

ARM processor cores in the Cortex-M family implementing the ARMv6-M instruction set architecture do not include hardware support for integer division operations. However, in certain deeply-embedded application spaces, hardware support for this class of arithmetic operations along with an unsigned square root function is important to maximize the system performance and minimize the device power dissipation. Accordingly, the MMDVSQ module is included to serve as a memory-mapped co-processor located in a special address space within the system memory map accessible only to the processor core. The MMDVSQ module supports execution of the integer division operations defined in the ARMv7-M instruction set architecture plus an unsigned integer square root operation. The supported integer division operations include 32/32 signed (SDIV) and unsigned (UDIV) calculations.

16.2 Function groups

16.2.1 MMDVSQ functional Operation

This group implements the MMDVSQ functional API.

16.2.2 MMDVSQ status Operation

This group implements the MMDVSQ status API.

16.3 Typical use case and example

Example: Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_-examples/mmdvsq

Enumerations

- enum mmdvsq_execution_status_t {
  kMMDVSQ_IdleSquareRoot = 0x01U,
  kMMDVSQ_IdleDivide = 0x02U,
  kMMDVSQ_BusySquareRoot = 0x05U,
  kMMDVSQ_BusyDivide = 0x06U }
  MMDVSQ execution status.

- enum mmdvsq_fast_start_select_t {
  kMMDVSQ_EnableFastStart = 0U,
  kMMDVSQ_DisableFastStart }

  MMDVSQ divide fast start select.

Driver version

- #define FSL_MMSVSQ_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

  Version 2.0.3.

MMDVSQ functional Operation

- int32_t MMDVSQ_GetDivideRemainder (MMDVSQ_Type ∗base, int32_t dividend, int32_t divisor, bool isUnsigned)

  Performs the MMDVSQ division operation and returns the remainder.
- int32_t MMDVSQ_GetDivideQuotient (MMDVSQ_Type ∗base, int32_t dividend, int32_t divisor, bool isUnsigned)

  Performs the MMDVSQ division operation and returns the quotient.
- uint16_t MMDVSQ_Sqrt (MMDVSQ_Type ∗base, uint32_t radicand)

  Performs the MMDVSQ square root operation.

MMDVSQ status Operation

- static mmdvsq_execution_status_t MMDVSQ_GetExecutionStatus (MMDVSQ_Type ∗base)

  Gets the MMDVSQ execution status.
- static void MMDVSQ_SetFastStartConfig (MMDVSQ_Type ∗base, mmdvsq_fast_start_select_t mode)

  Configures MMDVSQ fast start mode.
- static void MMDVSQ_SetDivideByZeroConfig (MMDVSQ_Type ∗base, bool isDivByZero)

  Configures the MMDVSQ divide-by-zero mode.

16.4 Macro Definition Documentation

16.4.1 #define FSL_MMSVSQ_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

16.5 Enumeration Type Documentation

16.5.1 enum mmdvsq_execution_status_t

Enumerator

kMMDVSQ_IdleSquareRoot MMDVSQ is idle; the last calculation was a square root.
kMMDVSQ_IdleDivide MMDVSQ is idle; the last calculation was division.
kMMDVSQ_BusySquareRoot MMDVSQ is busy processing a square root calculation.
kMMDVSQ_BusyDivide MMDVSQ is busy processing a division calculation.

16.5.2 enum mmdvsq_fast_start_select_t

Enumerator

kMMDVSQ_EnableFastStart Division operation is initiated by a write to the DSOR register.
kMMDVSQ_DisableFastStart Division operation is initiated by a write to CSR[SRT] = 1; normal start instead fast start.

16.6 Function Documentation

16.6.1 int32_t MMDVSQ_GetDivideRemainder (MMDVSQ_Type * *base*, int32_t *dividend*, int32_t *divisor*, bool *isUnsigned*)

Parameters

base	MMDVSQ peripheral address
dividend	Dividend value
divisor	Divisor value
isUnsigned	Mode of unsigned divide • true unsigned divide • false signed divide

16.6.2 int32_t MMDVSQ_GetDivideQuotient (MMDVSQ_Type * *base*, int32_t *dividend*, int32_t *divisor*, bool *isUnsigned*)

Parameters

base	MMDVSQ peripheral address
dividend	Dividend value
divisor	Divisor value
isUnsigned	Mode of unsigned divide • true unsigned divide • false signed divide

16.6.3 uint16_t MMDVSQ_Sqrt (MMDVSQ_Type * *base*, uint32_t *radicand*)

This function performs the MMDVSQ square root operation and returns the square root result of a given radicand value.

Parameters

base	MMDVSQ peripheral address
radicand	Radicand value

16.6.4 static mmdvsq_execution_status_t MMDVSQ_GetExecutionStatus (MMDVSQ_Type ∗ *base*) [inline], [static]

This function checks the current MMDVSQ execution status of the combined CSR[BUSY, DIV, SQRT] indicators.

Parameters

base	MMDVSQ peripheral address

Returns

Current MMDVSQ execution status

16.6.5 static void MMDVSQ_SetFastStartConfig (MMDVSQ_Type ∗ *base*, mmdvsq_fast_start_select_t *mode*) [inline], [static]

This function sets the MMDVSQ division fast start. The MMDVSQ supports two mechanisms for initiating a division operation. The default mechanism is a “fast start” where a write to the DSOR register begins the division. Alternatively, the start mechanism can begin after a write to the CSR register with CSR[SRT] set.

Parameters

base	MMDVSQ peripheral address
mode	Mode of Divide-Fast-Start • kMmdvsqDivideFastStart = 0 • kMmdvsqDivideNormalStart = 1

16.6.6 static void MMDVSQ_SetDivideByZeroConfig (MMDVSQ_Type ∗ *base*, bool *isDivByZero*) [inline], [static]

This function configures the MMDVSQ response to divide-by-zero calculations. If both CSR[DZ] and CSR[DZE] are set, then a subsequent read of the RES register is error-terminated to signal the processor of the attempted divide-by-zero. Otherwise, the register contents are returned.

Parameters

base	MMDVSQ peripheral address
isDivByZero	Mode of Divide-By-Zero • kMmdvsqDivideByZeroDis = 0 • kMmdvsqDivideByZeroEn = 1

Chapter 17
MSCAN: Scalable Controller Area Network

17.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Scalable Controller Area Network (MSCAN) module of MCUXpresso SDK devices.

Modules

- MSCAN Driver

17.2 MSCAN Driver

17.2.1 Overview

This section describes the programming interface of the MSCAN driver. The MSCAN driver configures MSCAN module and provides functional and transactional interfaces to build the MSCAN application.

17.2.2 Typical use case

17.2.2.1 Message Buffer Send Operation

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

17.2.2.2 Message Buffer Receive Operation

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

17.2.2.3 Receive FIFO Operation

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

17.2.2.4 Calculate

Provides static functions to calculate improved timing configuration.

The feature need to be enabled by user like that.

```
#define FSL_FEATURE_FLEXCAN_HAS_IMPROVED_TIMING_CONFIG (1)
```

Data Structures

- struct MSCAN_IDR1Type
    MSCAN IDR1 struct. More...
- struct MSCAN_IDR3Type
    MSCAN IDR3 struct. More...
- union IDR1_3_UNION
    MSCAN idr1 and idr3 union. More...
- struct MSCAN_ExtendIDType
    MSCAN extend ID struct. More...
- struct MSCAN_StandardIDType
    MSCAN standard ID struct. More...
- struct mscan_mb_t

MsCAN message buffer structure. More...
- struct mscan_frame_t
  MsCAN frame structure. More...
- struct mscan_idfilter_config_t
  MsCAN module acceptance filter configuration structure. More...
- struct mscan_config_t
  MsCAN module configuration structure. More...
- struct mscan_timing_config_t
  MsCAN protocol timing characteristic configuration structure. More...
- struct mscan_mb_transfer_t
  MSCAN Message Buffer transfer. More...
- struct mscan_handle_t
  MsCAN handle structure. More...

Macros

- #define MSCAN_RX_MB_STD_MASK(id)
  MsCAN Rx Message Buffer Mask helper macro.
- #define MSCAN_RX_MB_EXT_MASK(id)
  Extend Rx Message Buffer Mask helper macro.

Typedefs

- typedef void(∗ mscan_transfer_callback_t)(MSCAN_Type ∗base, mscan_handle_t ∗handle, status-_t status, void ∗userData)
  MsCAN transfer callback function.

Enumerations

- enum {
  kStatus_MSCAN_TxBusy = MAKE_STATUS(kStatusGroup_MSCAN, 0),
  kStatus_MSCAN_TxIdle = MAKE_STATUS(kStatusGroup_MSCAN, 1),
  kStatus_MSCAN_TxSwitchToRx,
  kStatus_MSCAN_RxBusy = MAKE_STATUS(kStatusGroup_MSCAN, 3),
  kStatus_MSCAN_RxIdle = MAKE_STATUS(kStatusGroup_MSCAN, 4),
  kStatus_MSCAN_RxOverflow = MAKE_STATUS(kStatusGroup_MSCAN, 5),
  kStatus_MSCAN_RxFifoBusy = MAKE_STATUS(kStatusGroup_MSCAN, 6),
  kStatus_MSCAN_RxFifoIdle = MAKE_STATUS(kStatusGroup_MSCAN, 7),
  kStatus_MSCAN_RxFifoOverflow = MAKE_STATUS(kStatusGroup_MSCAN, 8),
  kStatus_MSCAN_RxFifoWarning = MAKE_STATUS(kStatusGroup_MSCAN, 9),
  kStatus_MSCAN_ErrorStatus = MAKE_STATUS(kStatusGroup_MSCAN, 10),
  kStatus_MSCAN_UnHandled = MAKE_STATUS(kStatusGroup_MSCAN, 11) }
  FlexCAN transfer status.
- enum mscan_frame_format_t {
  kMSCAN_FrameFormatStandard = 0x0U,

kMSCAN_FrameFormatExtend = 0x1U }
MsCAN frame format.
- enum mscan_frame_type_t {
kMSCAN_FrameTypeData = 0x0U,
kMSCAN_FrameTypeRemote = 0x1U }
MsCAN frame type.
- enum mscan_clock_source_t {
kMSCAN_ClkSrcOsc = 0x0U,
kMSCAN_ClkSrcBus = 0x1U }
MsCAN clock source.
- enum mscan_busoffrec_mode_t {
kMSCAN_BusoffrecAuto = 0x0U,
kMSCAN_BusoffrecUsr = 0x1U }
MsCAN bus-off recovery mode.
- enum _mscan_tx_buffer_empty_flag {
kMSCAN_TxBuf0Empty = 0x1U,
kMSCAN_TxBuf1Empty = 0x2U,
kMSCAN_TxBuf2Empty = 0x4U,
kMSCAN_TxBufFull = 0x0U }
MsCAN Tx buffer empty flag.
- enum mscan_id_filter_mode_t {
kMSCAN_Filter32Bit = 0x0U,
kMSCAN_Filter16Bit = 0x1U,
kMSCAN_Filter8Bit = 0x2U,
kMSCAN_FilterClose = 0x3U }
MsCAN id filter mode.
- enum _mscan_interrupt_enable {
kMSCAN_WakeUpInterruptEnable = MSCAN_CANRIER_WUPIE_MASK,
kMSCAN_StatusChangeInterruptEnable = MSCAN_CANRIER_CSCIE_MASK,
kMSCAN_RxStatusChangeInterruptEnable = MSCAN_CANRIER_RSTATE_MASK,
kMSCAN_TxStatusChangeInterruptEnable = MSCAN_CANRIER_TSTATE_MASK,
kMSCAN_OverrunInterruptEnable = MSCAN_CANRIER_OVRIE_MASK,
kMSCAN_RxFullInterruptEnable = MSCAN_CANRIER_RXFIE_MASK,
kMSCAN_TxEmptyInterruptEnable = MSCAN_CANTIER_TXEIE_MASK }
MsCAN interrupt configuration structure, default settings all disabled.

Driver version

- #define FSL_MSCAN_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))
MsCAN driver version.

Initialization and deinitialization

- void MSCAN_Init (MSCAN_Type ∗base, const mscan_config_t ∗config, uint32_t sourceClock_Hz)
Initializes a MsCAN instance.

- void MSCAN_Deinit (MSCAN_Type *base)
  De-initializes a MsCAN instance.
- void MSCAN_GetDefaultConfig (mscan_config_t *config)
  Gets the default configuration structure.

Configuration.

- static uint8_t MSCAN_GetTxBufferEmptyFlag (MSCAN_Type *base)
  Get the transmit buffer empty status.
- static void MSCAN_TxBufferSelect (MSCAN_Type *base, uint8_t txBuf)
  The selection of the actual transmit message buffer.
- static uint8_t MSCAN_GetTxBufferSelect (MSCAN_Type *base)
  Get the actual transmit message buffer.
- static void MSCAN_TxBufferLaunch (MSCAN_Type *base, uint8_t txBuf)
  Clear TFLG to schedule for transmission.
- static uint8_t MSCAN_GetTxBufferStatusFlags (MSCAN_Type *base, uint8_t mask)
  Get Tx buffer status flag.
- static uint8_t MSCAN_GetRxBufferFullFlag (MSCAN_Type *base)
  Check Receive Buffer Full Flag.
- static void MSCAN_ClearRxBufferFullFlag (MSCAN_Type *base)
  Clear Receive buffer Full flag.
- static uint8_t **MSCAN_ReadRIDR0** (MSCAN_Type *base)
- static uint8_t **MSCAN_ReadRIDR1** (MSCAN_Type *base)
- static uint8_t **MSCAN_ReadRIDR2** (MSCAN_Type *base)
- static uint8_t **MSCAN_ReadRIDR3** (MSCAN_Type *base)
- static void **MSCAN_WriteTIDR0** (MSCAN_Type *base, uint8_t id)
- static void **MSCAN_WriteTIDR1** (MSCAN_Type *base, uint8_t id)
- static void **MSCAN_WriteTIDR2** (MSCAN_Type *base, uint8_t id)
- static void **MSCAN_WriteTIDR3** (MSCAN_Type *base, uint8_t id)
- static void **MSCAN_SetIDFilterMode** (MSCAN_Type *base, mscan_id_filter_mode_t mode)
- static void **MSCAN_WriteIDAR0** (MSCAN_Type *base, uint8_t *pID)
- static void **MSCAN_WriteIDAR1** (MSCAN_Type *base, uint8_t *pID)
- static void **MSCAN_WriteIDMR0** (MSCAN_Type *base, uint8_t *pID)
- static void **MSCAN_WriteIDMR1** (MSCAN_Type *base, uint8_t *pID)
- void MSCAN_SetTimingConfig (MSCAN_Type *base, const mscan_timing_config_t *config)
  Sets the MsCAN protocol timing characteristic.

Status

- static uint8_t MSCAN_GetTxBufEmptyFlags (MSCAN_Type *base)
  Gets the MsCAN Tx buffer empty flags.

Interrupts

- static void MSCAN_EnableTxInterrupts (MSCAN_Type *base, uint8_t mask)
  Enables MsCAN Transmitter interrupts according to the provided mask.
- static void MSCAN_DisableTxInterrupts (MSCAN_Type *base, uint8_t mask)
  Disables MsCAN Transmitter interrupts according to the provided mask.

- static void MSCAN_EnableRxInterrupts (MSCAN_Type ∗base, uint8_t mask)
  Enables MsCAN Receiver interrupts according to the provided mask.
- static void MSCAN_DisableRxInterrupts (MSCAN_Type ∗base, uint8_t mask)
  Disables MsCAN Receiver interrupts according to the provided mask.
- static void MSCAN_AbortTxRequest (MSCAN_Type ∗base, uint8_t mask)
  Abort MsCAN Tx request.

Bus Operations

- static void MSCAN_Enable (MSCAN_Type ∗base, bool enable)
  Enables or disables the MsCAN module operation.
- status_t MSCAN_WriteTxMb (MSCAN_Type ∗base, mscan_frame_t ∗pTxFrame)
  Writes a MsCAN Message to the Transmit Message Buffer.
- status_t MSCAN_ReadRxMb (MSCAN_Type ∗base, mscan_frame_t ∗pRxFrame)
  Reads a MsCAN Message from Receive Message Buffer.

Transactional

- void MSCAN_TransferCreateHandle (MSCAN_Type ∗base, mscan_handle_t ∗handle, mscan_transfer_callback_t callback, void ∗userData)
  Initializes the MsCAN handle.
- status_t MSCAN_TransferSendBlocking (MSCAN_Type ∗base, mscan_frame_t ∗pTxFrame)
  Performs a polling send transaction on the CAN bus.
- status_t MSCAN_TransferReceiveBlocking (MSCAN_Type ∗base, mscan_frame_t ∗pRxFrame)
  Performs a polling receive transaction on the CAN bus.
- status_t MSCAN_TransferSendNonBlocking (MSCAN_Type ∗base, mscan_handle_t ∗handle, mscan_mb_transfer_t ∗xfer)
  Sends a message using IRQ.
- status_t MSCAN_TransferReceiveNonBlocking (MSCAN_Type ∗base, mscan_handle_t ∗handle, mscan_mb_transfer_t ∗xfer)
  Receives a message using IRQ.
- void MSCAN_TransferAbortSend (MSCAN_Type ∗base, mscan_handle_t ∗handle, uint8_t mask)
  Aborts the interrupt driven message send process.
- void MSCAN_TransferAbortReceive (MSCAN_Type ∗base, mscan_handle_t ∗handle, uint8_t mask)
  Aborts the interrupt driven message receive process.
- void MSCAN_TransferHandleIRQ (MSCAN_Type ∗base, mscan_handle_t ∗handle)
  MSCAN IRQ handle function.

17.2.3 Data Structure Documentation

17.2.3.1 struct MSCAN_IDR1Type

Data Fields

- uint8_t EID17_15: 3

Extended Format Identifier 17-15.
- uint8_t R_TEIDE: 1
  ID Extended.
- uint8_t R_TSRR: 1
  Substitute Remote Request.
- uint8_t EID20_18_OR_SID2_0: 3
  Extended Format Identifier 18-20 or standard format bit 0-2.

17.2.3.2 struct MSCAN_IDR3Type

Data Fields

- uint8_t ERTR: 1
  Remote Transmission Request.
- uint8_t EID6_0: 7
  Extended Format Identifier 6-0.

17.2.3.3 union IDR1_3_UNION

Data Fields

- MSCAN_IDR1Type IDR1
  structure for identifier 1
- MSCAN_IDR3Type IDR3
  structure for identifier 3
- uint8_t Bytes
  bytes

17.2.3.4 struct MSCAN_ExtendIDType

Data Fields

- uint32_t EID6_0: 7
  ID[0:6].
- uint32_t EID14_7: 8
  ID[14:7].
- uint32_t EID17_15: 3
  ID[17:15].
- uint32_t EID20_18: 3
  ID[20:18].
- uint32_t EID28_21: 8
  ID[28:21].

17.2.3.5 struct MSCAN_StandardIDType

Data Fields

- uint32_t EID2_0: 3
  ID[0:2].
- uint32_t EID10_3: 8
  ID[10:3].

17.2.3.6 struct mscan_mb_t

Data Fields

- uint8_t EIDR0
  Extended Identifier Register 0.
- uint8_t EIDR1
  Extended Identifier Register 1.
- uint8_t EIDR2
  Extended Identifier Register 2.
- uint8_t EIDR3
  Extended Identifier Register 3.
- uint8_t EDSR [8]
  Extended Data Segment Register.
- uint8_t DLR
  data length field
- uint8_t BPR
  Buffer Priority Register.
- uint8_t TSRH
  Time Stamp Register High.
- uint8_t TSRL
  Time Stamp Register Low.

17.2.3.7 struct mscan_frame_t

Data Fields

- union {
  MSCAN_StandardIDType StdID
  standard format
  MSCAN_ExtendIDType ExtID
  extend format
  uint32_t ID
  Identifire with 32 bit format.
  } ID_Type

  identifier union
- uint8_t DLR
  data length

- uint8_t BPR
  transmit buffer priority
- mscan_frame_type_t type
  remote frame or data frame
- mscan_frame_format_t format
  extend frame or standard frame
- uint8_t TSRH
  time stamp high byte
- uint8_t TSRL
  time stamp low byte
- uint8_t DSR [8]
  data segment
- uint32_t dataWord0
  MSCAN Frame payload word0.
- uint32_t dataWord1
  MSCAN Frame payload word1.
- uint8_t dataByte0
  MSCAN Frame payload byte0.
- uint8_t dataByte1
  MSCAN Frame payload byte1.
- uint8_t dataByte2
  MSCAN Frame payload byte2.
- uint8_t dataByte3
  MSCAN Frame payload byte3.
- uint8_t dataByte4
  MSCAN Frame payload byte4.
- uint8_t dataByte5
  MSCAN Frame payload byte5.
- uint8_t dataByte6
  MSCAN Frame payload byte6.
- uint8_t dataByte7
  MSCAN Frame payload byte7.

Field Documentation

(1) uint32_t mscan_frame_t::dataWord0

(2) uint32_t mscan_frame_t::dataWord1

(3) uint8_t mscan_frame_t::dataByte0

(4) uint8_t mscan_frame_t::dataByte1

(5) uint8_t mscan_frame_t::dataByte2

(6) uint8_t mscan_frame_t::dataByte3

(7) uint8_t mscan_frame_t::dataByte4

(8) uint8_t mscan_frame_t::dataByte5

(9) uint8_t mscan_frame_t::dataByte6

(10) uint8_t mscan_frame_t::dataByte7

17.2.3.8 struct mscan_idfilter_config_t

Data Fields

- mscan_id_filter_mode_t filterMode
  MSCAN Identifier Acceptance Filter Mode.
- uint32_t u32IDAR0
  MSCAN Identifier Acceptance Register n of First Bank.
- uint32_t u32IDAR1
  MSCAN Identifier Acceptance Register n of Second Bank.
- uint32_t u32IDMR0
  MSCAN Identifier Mask Register n of First Bank.
- uint32_t u32IDMR1
  MSCAN Identifier Mask Register n of Second Bank.

17.2.3.9 struct mscan_config_t

Data Fields

- uint32_t baudRate
  MsCAN baud rate in bps.
- bool enableTimer
  Enable or Disable free running timer.
- bool enableWakeup
  Enable or Disable Wakeup Mode.
- mscan_clock_source_t clkSrc
  Clock source for MsCAN Protocol Engine.
- bool enableLoopBack
  Enable or Disable Loop Back Self Test Mode.
- bool enableListen
  Enable or Disable Listen Only Mode.
- mscan_busoffrec_mode_t busoffrecMode
  Bus-Off Recovery Mode.

Field Documentation

(1) uint32_t mscan_config_t::baudRate

(2) bool mscan_config_t::enableTimer

(3) bool mscan_config_t::enableWakeup

(4) mscan_clock_source_t mscan_config_t::clkSrc

(5) bool mscan_config_t::enableLoopBack

(6) bool mscan_config_t::enableListen

(7) mscan_busoffrec_mode_t mscan_config_t::busoffrecMode

17.2.3.10 struct mscan_timing_config_t

Data Fields

- uint8_t priDiv
  Baud rate prescaler.
- uint8_t sJumpwidth
  Sync Jump Width.
- uint8_t timeSeg1
  Time Segment 1.
- uint8_t timeSeg2
  Time Segment 2.
- uint8_t samp
  Number of samples per bit time.

Field Documentation

(1) uint8_t mscan_timing_config_t::priDiv

(2) uint8_t mscan_timing_config_t::sJumpwidth

(3) uint8_t mscan_timing_config_t::timeSeg1

(4) uint8_t mscan_timing_config_t::timeSeg2

(5) uint8_t mscan_timing_config_t::samp

17.2.3.11 struct mscan_mb_transfer_t

Data Fields

- mscan_frame_t ∗ frame
  The buffer of CAN Message to be transfer.
- uint8_t mask
  The mask of Tx buffer.

Field Documentation

(1) mscan_frame_t∗ mscan_mb_transfer_t::frame

(2) uint8_t mscan_mb_transfer_t::mask

17.2.3.12 struct _mscan_handle

MsCAN handle structure definition.

Data Fields

- mscan_transfer_callback_t callback
  Callback function.
- void ∗ userData
  MsCAN callback function parameter.
- mscan_frame_t ∗volatile mbFrameBuf
  The buffer for received data from Message Buffers.
- volatile uint8_t mbStateTx
  Message Buffer transfer state.
- volatile uint8_t mbStateRx
  Message Buffer transfer state.

Field Documentation

(1) mscan_transfer_callback_t mscan_handle_t::callback

(2) void∗ mscan_handle_t::userData

(3) mscan_frame_t∗ volatile mscan_handle_t::mbFrameBuf

(4) volatile uint8_t mscan_handle_t::mbStateTx

(5) volatile uint8_t mscan_handle_t::mbStateRx

17.2.4 Macro Definition Documentation

17.2.4.1 #define FSL_MSCAN_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))

17.2.4.2 #define MSCAN_RX_MB_STD_MASK(*id*)

Value:

```
((uint32_t)((((uint32_t)(id)&0x7) << 21) | \
                (((uint32_t)(id) >> 3) & 0xFF) << 24)))
```

Standard Rx Message Buffer Mask helper macro.

17.2.4.3 #define MSCAN_RX_MB_EXT_MASK(*id*)

Value:

```
((uint32_t)(((((uint32_t)(id) >> 21) & 0xFF) << 24) | ((((uint32_t)(id) >> 18) & 0x7) << 21) | \
                ((((uint32_t)(id) >> 15) & 0x7) << 16) | ((((uint32_t)(id) >> 7) & 0xFF)) << 8) | \
                (((uint32_t)(id)&0x7F) << 1)))
```

17.2.5 Typedef Documentation

17.2.5.1 typedef void(∗ mscan_transfer_callback_t)(MSCAN_Type ∗base, mscan_handle_t ∗handle, status_t status, void ∗userData)

The MsCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_-MSCAN_ErrorStatus, the result parameter is the Content of MsCAN status register which can be used to get the working status(or error status) of MsCAN module. If the status equals to other MsCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other MsCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

17.2.6 Enumeration Type Documentation

17.2.6.1 anonymous enum

Enumerator

kStatus_MSCAN_TxBusy Tx Message Buffer is Busy.
kStatus_MSCAN_TxIdle Tx Message Buffer is Idle.
kStatus_MSCAN_TxSwitchToRx Remote Message is send out and Message buffer changed to Receive one.
kStatus_MSCAN_RxBusy Rx Message Buffer is Busy.
kStatus_MSCAN_RxIdle Rx Message Buffer is Idle.
kStatus_MSCAN_RxOverflow Rx Message Buffer is Overflowed.
kStatus_MSCAN_RxFifoBusy Rx Message FIFO is Busy.
kStatus_MSCAN_RxFifoIdle Rx Message FIFO is Idle.
kStatus_MSCAN_RxFifoOverflow Rx Message FIFO is overflowed.
kStatus_MSCAN_RxFifoWarning Rx Message FIFO is almost overflowed.
kStatus_MSCAN_ErrorStatus FlexCAN Module Error and Status.
kStatus_MSCAN_UnHandled UnHadled Interrupt asserted.

17.2.6.2 enum mscan_frame_format_t

Enumerator

kMSCAN_FrameFormatStandard Standard frame format attribute.
kMSCAN_FrameFormatExtend Extend frame format attribute.

17.2.6.3 enum mscan_frame_type_t

Enumerator

kMSCAN_FrameTypeData Data frame type attribute.
kMSCAN_FrameTypeRemote Remote frame type attribute.

17.2.6.4 enum mscan_clock_source_t

Enumerator

kMSCAN_ClkSrcOsc MsCAN Protocol Engine clock from Oscillator.
kMSCAN_ClkSrcBus MsCAN Protocol Engine clock from Bus Clock.

17.2.6.5 enum mscan_busoffrec_mode_t

Enumerator

kMSCAN_BusoffrecAuto MsCAN automatic bus-off recovery.
kMSCAN_BusoffrecUsr MsCAN bus-off recovery upon user request.

17.2.6.6 enum _mscan_tx_buffer_empty_flag

Enumerator

kMSCAN_TxBuf0Empty MsCAN Tx Buffer 0 empty.
kMSCAN_TxBuf1Empty MsCAN Tx Buffer 1 empty.
kMSCAN_TxBuf2Empty MsCAN Tx Buffer 2 empty.
kMSCAN_TxBufFull MsCAN Tx Buffer all not empty.

17.2.6.7 enum mscan_id_filter_mode_t

Enumerator

kMSCAN_Filter32Bit Two 32-bit acceptance filters.
kMSCAN_Filter16Bit Four 16-bit acceptance filters.
kMSCAN_Filter8Bit Eight 8-bit acceptance filters.
kMSCAN_FilterClose Filter closed.

17.2.6.8 enum _mscan_interrupt_enable

This structure contains the settings for all of the MsCAN Module interrupt configurations.

Enumerator

kMSCAN_WakeUpInterruptEnable Wake Up interrupt.
kMSCAN_StatusChangeInterruptEnable Status change interrupt.
kMSCAN_RxStatusChangeInterruptEnable Rx status change interrupt.
kMSCAN_TxStatusChangeInterruptEnable Tx status change interrupt.
kMSCAN_OverrunInterruptEnable Overrun interrupt.
kMSCAN_RxFullInterruptEnable Rx buffer full interrupt.
kMSCAN_TxEmptyInterruptEnable Tx buffer empty interrupt.

17.2.7 Function Documentation

17.2.7.1 void MSCAN_Init (MSCAN_Type ∗ *base*, const mscan_config_t ∗ *config*, uint32_t *sourceClock_Hz*)

This function initializes the MsCAN module with user-defined settings. This example shows how to set up the mscan_config_t parameters and how to call the MSCAN_Init function by passing in these parameters.

```
*   mscan_config_t mscanConfig;
*   mscanConfig.clkSrc            = kMSCAN_ClkSrcOsc;
*   mscanConfig.baudRate          = 1250000U;
*   mscanConfig.enableTimer       = false;
*   mscanConfig.enableLoopBack    = false;
*   mscanConfig.enableWakeup      = false;
*   mscanConfig.enableListen      = false;
*   mscanConfig.busoffrecMode     = kMSCAN_BusoffrecAuto;
*   mscanConfig.filterConfig.filterMode = kMSCAN_Filter32Bit;
*   MSCAN_Init(MSCAN, &mscanConfig, 8000000UL);
*
```

Parameters

base	MsCAN peripheral base address.
config	Pointer to the user-defined configuration structure.
sourceClock_-Hz	MsCAN Protocol Engine clock source frequency in Hz.

17.2.7.2 void MSCAN_Deinit (MSCAN_Type ∗ *base*)

This function disables the MsCAN module clock and sets all register values to the reset value.

Parameters

base	MsCAN peripheral base address.

17.2.7.3 void MSCAN_GetDefaultConfig (mscan_config_t ∗ *config*)

This function initializes the MsCAN configuration structure to default values.

Parameters

config	Pointer to the MsCAN configuration structure.

17.2.7.4 static uint8_t MSCAN_GetTxBufferEmptyFlag (MSCAN_Type * *base*) [inline], [static]

This flag indicates that the associated transmit message buffer is empty.

Parameters

base	MsCAN peripheral base address.

17.2.7.5 static void MSCAN_TxBufferSelect (MSCAN_Type * *base,* uint8_t *txBuf*) [inline], [static]

To get the next available transmit buffer, read the CANTFLG register and write its value back into the CANTBSEL register.

Parameters

base	MsCAN peripheral base address.
txBuf	The value read from CANTFLG.

17.2.7.6 static uint8_t MSCAN_GetTxBufferSelect (MSCAN_Type * *base*) [inline], [static]

After write TFLG value back into the CANTBSEL register, read again CANBSEL to get the actual trasnsmit message buffer.

Parameters

base	MsCAN peripheral base address.

17.2.7.7 static void MSCAN_TxBufferLaunch (MSCAN_Type * *base,* uint8_t *txBuf*) [inline], [static]

The CPU must clear the flag after a message is set up in the transmit buffer and is due for transmission.

Parameters

base	MsCAN peripheral base address.
txBuf	Message buffer(s) to be cleared.

17.2.7.8 static uint8_t MSCAN_GetTxBufferStatusFlags (MSCAN_Type * *base,* uint8_t *mask*) [inline], [static]

The bit is set after successful transmission.

Parameters

base	MsCAN peripheral base address.
mask	Message buffer(s) mask.

17.2.7.9 static uint8_t MSCAN_GetRxBufferFullFlag (MSCAN_Type * *base*) [inline], [static]

RXF is set by the MSCAN when a new message is shifted in the receiver FIFO. This flag indicates whether the shifted buffer is loaded with a correctly received message.

Parameters

base	MsCAN peripheral base address.

17.2.7.10 static void MSCAN_ClearRxBufferFullFlag (MSCAN_Type * *base*) [inline], [static]

After the CPU has read that message from the RxFG buffer in the receiver FIFO The RXF flag must be cleared to release the buffer.

Parameters

base	MsCAN peripheral base address.

17.2.7.11 void MSCAN_SetTimingConfig (MSCAN_Type * *base,* const mscan_timing_config_t * *config*)

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MSCAN_Init() and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling MSCAN_SetTimingConfig() overrides the baud rate set in MSCAN_Init().

Parameters

base	MsCAN peripheral base address.
config	Pointer to the timing configuration structure.

17.2.7.12 static uint8_t MSCAN_GetTxBufEmptyFlags (MSCAN_Type * *base*) [inline], [static]

This function gets MsCAN Tx buffer empty flags. It's returned as the value of the enumerators _mscan_-tx_buffer_empty_flag.

Parameters

base	MsCAN peripheral base address.

Returns

Tx buffer empty flags in the _mscan_tx_buffer_empty_flag.

17.2.7.13 static void MSCAN_EnableTxInterrupts (MSCAN_Type * *base*, uint8_t *mask*) [inline], [static]

This function enables the MsCAN Tx empty interrupts according to the mask.

Parameters

base	MsCAN peripheral base address.
mask	The Tx interrupts mask to enable.

17.2.7.14 static void MSCAN_DisableTxInterrupts (MSCAN_Type * *base*, uint8_t *mask*) [inline], [static]

This function disables the MsCAN Tx emtpy interrupts according to the mask.

Parameters

base	MsCAN peripheral base address.
mask	The Tx interrupts mask to disable.

17.2.7.15 static void MSCAN_EnableRxInterrupts (MSCAN_Type * *base*, uint8_t *mask*) [inline], [static]

This function enables the MsCAN Rx interrupts according to the provided mask which is a logical OR of enumeration members, see _mscan_interrupt_enable.

Parameters

base	MsCAN peripheral base address.
mask	The interrupts to enable. Logical OR of _mscan_interrupt_enable.

17.2.7.16 static void MSCAN_DisableRxInterrupts (MSCAN_Type * *base*, uint8_t *mask*) [inline], [static]

This function disables the MsCAN Rx interrupts according to the provided mask which is a logical OR of enumeration members, see _mscan_interrupt_enable.

Parameters

base	MsCAN peripheral base address.
mask	The interrupts to disable. Logical OR of _mscan_interrupt_enable.

17.2.7.17 static void MSCAN_AbortTxRequest (MSCAN_Type * *base*, uint8_t *mask*) [inline], [static]

This function allows abort request of queued messages.

Parameters

base	MsCAN peripheral base address.
mask	The Tx mask to abort.

17.2.7.18 static void MSCAN_Enable (MSCAN_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the MsCAN module.

Parameters

base	MsCAN base pointer.
enable	true to enable, false to disable.

17.2.7.19 status_t MSCAN_WriteTxMb (MSCAN_Type * *base*, mscan_frame_t * *pTxFrame*)

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

base	MsCAN peripheral base address.
pTxFrame	Pointer to CAN message frame to be sent.

Return values

kStatus_Success	- Write Tx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

17.2.7.20 status_t MSCAN_ReadRxMb (MSCAN_Type * *base*, mscan_frame_t * *pRxFrame*)

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

base	MsCAN peripheral base address.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Rx Message Buffer is full and has been read successfully.

kStatus_Fail	- Rx Message Buffer is empty.

17.2.7.21 void MSCAN_TransferCreateHandle (MSCAN_Type * *base*, mscan_handle_t * *handle*, mscan_transfer_callback_t *callback*, void * *userData*)

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

Parameters

base	MsCAN peripheral base address.
handle	MsCAN handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

17.2.7.22 status_t MSCAN_TransferSendBlocking (MSCAN_Type * *base*, mscan_frame_t * *pTxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	MsCAN peripheral base pointer.
pTxFrame	Pointer to CAN message frame to be sent.

Return values

kStatus_Success	- Write Tx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

17.2.7.23 status_t MSCAN_TransferReceiveBlocking (MSCAN_Type * *base*, mscan_frame_t * *pRxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	MsCAN peripheral base pointer.
pRxFrame	Pointer to CAN message frame to be received.

Return values

kStatus_Success	- Read Rx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

17.2.7.24 status_t MSCAN_TransferSendNonBlocking (MSCAN_Type ∗ *base*, mscan_handle_t ∗ *handle*, mscan_mb_transfer_t ∗ *xfer*)

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

base	MsCAN peripheral base address.
handle	MsCAN handle pointer.
xfer	MsCAN Message Buffer transfer structure. See the mscan_mb_transfer_t.

Return values

kStatus_Success	Start Tx Message Buffer sending process successfully.
kStatus_Fail	Write Tx Message Buffer failed.

17.2.7.25 status_t MSCAN_TransferReceiveNonBlocking (MSCAN_Type ∗ *base*, mscan_handle_t ∗ *handle*, mscan_mb_transfer_t ∗ *xfer*)

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

base	MsCAN peripheral base address.

handle	MsCAN handle pointer.
xfer	MsCAN Message Buffer transfer structure. See the mscan_mb_transfer_t.

Return values

kStatus_Success	- Start Rx Message Buffer receiving process successfully.
kStatus_MSCAN_RxBusy	- Rx Message Buffer is in use.

17.2.7.26 void MSCAN_TransferAbortSend (MSCAN_Type * *base*, mscan_handle_t * *handle*, uint8_t *mask*)

This function aborts the interrupt driven message send process.

Parameters

base	MsCAN peripheral base address.
handle	MsCAN handle pointer.
mask	The MsCAN Tx Message Buffer mask.

17.2.7.27 void MSCAN_TransferAbortReceive (MSCAN_Type * *base*, mscan_handle_t * *handle*, uint8_t *mask*)

This function aborts the interrupt driven message receive process.

Parameters

base	MsCAN peripheral base address.
handle	MsCAN handle pointer.
mask	The MsCAN Rx Message Buffer mask.

17.2.7.28 void MSCAN_TransferHandleIRQ (MSCAN_Type * *base*, mscan_handle_t * *handle*)

This function handles the MSCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

base	MSCAN peripheral base address.
handle	MSCAN handle pointer.

Chapter 18
PDB: Programmable Delay Block

18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Programmable Delay Block (PDB) module of MCUXpresso SDK devices.

The PDB driver includes a basic PDB counter, trigger generators for ADC, DAC, and pulse-out.

The basic PDB counter can be used as a general programmable timer with an interrupt. The counter increases automatically with the divided clock signal after it is triggered to start by an external trigger input or the software trigger. There are "milestones" for the output trigger event. When the counter is equal to any of these "milestones", the corresponding trigger is generated and sent out to other modules. These "milestones" are for the following events.

- Counter delay interrupt, which is the interrupt for the PDB module
- ADC pre-trigger to trigger the ADC conversion
- DAC interval trigger to trigger the DAC buffer and move the buffer read pointer
- Pulse-out triggers to generate a single of rising and falling edges, which can be assembled to a window.

The "milestone" values have a flexible load mode. To call the APIs to set these value is equivalent to writing data to their buffer. The loading event occurs as the load mode describes. This design ensures that all "milestones" can be updated at the same time.

18.2 Typical use case

18.2.1 Working as basic PDB counter with a PDB interrupt.

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

18.2.2 Working with an additional trigger. The ADC trigger is used as an example.

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

Data Structures

- struct pdb_config_t
  PDB module configuration. More...
- struct pdb_adc_pretrigger_config_t
  PDB ADC Pre-trigger configuration. More...
- struct pdb_dac_trigger_config_t
  PDB DAC trigger configuration. More...

Enumerations

- enum _pdb_status_flags {
  kPDB_LoadOKFlag = PDB_SC_LDOK_MASK,
  kPDB_DelayEventFlag = PDB_SC_PDBIF_MASK }
  PDB flags.
- enum _pdb_adc_pretrigger_flags {
  kPDB_ADCPreTriggerChannel0Flag = PDB_S_CF(1U << 0),
  kPDB_ADCPreTriggerChannel1Flag = PDB_S_CF(1U << 1),
  kPDB_ADCPreTriggerChannel2Flag = PDB_S_CF(1U << 2),
  kPDB_ADCPreTriggerChannel3Flag = PDB_S_CF(1U << 3),
  kPDB_ADCPreTriggerChannel0ErrorFlag = PDB_S_ERR(1U << 0),
  kPDB_ADCPreTriggerChannel1ErrorFlag = PDB_S_ERR(1U << 1),
  kPDB_ADCPreTriggerChannel2ErrorFlag = PDB_S_ERR(1U << 2),
  kPDB_ADCPreTriggerChannel3ErrorFlag = PDB_S_ERR(1U << 3) }
  PDB ADC PreTrigger channel flags.
- enum _pdb_interrupt_enable {
  kPDB_SequenceErrorInterruptEnable = PDB_SC_PDBEIE_MASK,
  kPDB_DelayInterruptEnable = PDB_SC_PDBIE_MASK }
  PDB buffer interrupts.
- enum pdb_load_value_mode_t {
  kPDB_LoadValueImmediately = 0U,
  kPDB_LoadValueOnCounterOverflow = 1U,
  kPDB_LoadValueOnTriggerInput = 2U,
  kPDB_LoadValueOnCounterOverflowOrTriggerInput = 3U }
  PDB load value mode.
- enum pdb_prescaler_divider_t {
  kPDB_PrescalerDivider1 = 0U,
  kPDB_PrescalerDivider2 = 1U,
  kPDB_PrescalerDivider4 = 2U,
  kPDB_PrescalerDivider8 = 3U,
  kPDB_PrescalerDivider16 = 4U,
  kPDB_PrescalerDivider32 = 5U,
  kPDB_PrescalerDivider64 = 6U,
  kPDB_PrescalerDivider128 = 7U }
  Prescaler divider.
- enum pdb_divider_multiplication_factor_t {
  kPDB_DividerMultiplicationFactor1 = 0U,
  kPDB_DividerMultiplicationFactor10 = 1U,
  kPDB_DividerMultiplicationFactor20 = 2U,
  kPDB_DividerMultiplicationFactor40 = 3U }
  Multiplication factor select for prescaler.
- enum pdb_trigger_input_source_t {

kPDB_TriggerInput0 = 0U,
kPDB_TriggerInput1 = 1U,
kPDB_TriggerInput2 = 2U,
kPDB_TriggerInput3 = 3U,
kPDB_TriggerInput4 = 4U,
kPDB_TriggerInput5 = 5U,
kPDB_TriggerInput6 = 6U,
kPDB_TriggerInput7 = 7U,
kPDB_TriggerInput8 = 8U,
kPDB_TriggerInput9 = 9U,
kPDB_TriggerInput10 = 10U,
kPDB_TriggerInput11 = 11U,
kPDB_TriggerInput12 = 12U,
kPDB_TriggerInput13 = 13U,
kPDB_TriggerInput14 = 14U,
kPDB_TriggerSoftware = 15U }
    Trigger input source.
- enum pdb_adc_trigger_channel_t {
kPDB_ADCTriggerChannel0 = 0U,
kPDB_ADCTriggerChannel1 = 1U,
kPDB_ADCTriggerChannel2 = 2U,
kPDB_ADCTriggerChannel3 = 3U }
    List of PDB ADC trigger channels.
- enum pdb_adc_pretrigger_t {
kPDB_ADCPreTrigger0 = 0U,
kPDB_ADCPreTrigger1 = 1U,
kPDB_ADCPreTrigger2 = 2U,
kPDB_ADCPreTrigger3 = 3U,
kPDB_ADCPreTrigger4 = 4U,
kPDB_ADCPreTrigger5 = 5U,
kPDB_ADCPreTrigger6 = 6U,
kPDB_ADCPreTrigger7 = 7U }
    List of PDB ADC pretrigger.
- enum pdb_dac_trigger_channel_t {
kPDB_DACTriggerChannel0 = 0U,
kPDB_DACTriggerChannel1 = 1U }
    List of PDB DAC trigger channels.
- enum pdb_pulse_out_trigger_channel_t {
kPDB_PulseOutTriggerChannel0 = 0U,
kPDB_PulseOutTriggerChannel1 = 1U,
kPDB_PulseOutTriggerChannel2 = 2U,
kPDB_PulseOutTriggerChannel3 = 3U }
    List of PDB pulse out trigger channels.
- enum pdb_pulse_out_channel_mask_t {

kPDB_PulseOutChannel0Mask = (1U << 0U),
kPDB_PulseOutChannel1Mask = (1U << 1U),
kPDB_PulseOutChannel2Mask = (1U << 2U),
kPDB_PulseOutChannel3Mask = (1U << 3U) }
List of PDB pulse out trigger channels mask.

Driver version

- #define FSL_PDB_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  PDB driver version 2.0.4.

Initialization

- void PDB_Init (PDB_Type *base, const pdb_config_t *config)
  Initializes the PDB module.
- void PDB_Deinit (PDB_Type *base)
  De-initializes the PDB module.
- void PDB_GetDefaultConfig (pdb_config_t *config)
  Initializes the PDB user configuration structure.
- static void PDB_Enable (PDB_Type *base, bool enable)
  Enables the PDB module.

Basic Counter

- static void PDB_DoSoftwareTrigger (PDB_Type *base)
  Triggers the PDB counter by software.
- static void PDB_DoLoadValues (PDB_Type *base)
  Loads the counter values.
- static void PDB_EnableDMA (PDB_Type *base, bool enable)
  Enables the DMA for the PDB module.
- static void PDB_EnableInterrupts (PDB_Type *base, uint32_t mask)
  Enables the interrupts for the PDB module.
- static void PDB_DisableInterrupts (PDB_Type *base, uint32_t mask)
  Disables the interrupts for the PDB module.
- static uint32_t PDB_GetStatusFlags (PDB_Type *base)
  Gets the status flags of the PDB module.
- static void PDB_ClearStatusFlags (PDB_Type *base, uint32_t mask)
  Clears the status flags of the PDB module.
- static void PDB_SetModulusValue (PDB_Type *base, uint32_t value)
  Specifies the counter period.
- static uint32_t PDB_GetCounterValue (PDB_Type *base)
  Gets the PDB counter's current value.
- static void PDB_SetCounterDelayValue (PDB_Type *base, uint32_t value)
  Sets the value for the PDB counter delay event.

ADC Pre-trigger

- static void PDB_SetADCPreTriggerConfig (PDB_Type *base, pdb_adc_trigger_channel_t channel, pdb_adc_pretrigger_config_t *config)
  Configures the ADC pre-trigger in the PDB module.

- static void PDB_SetADCPreTriggerDelayValue (PDB_Type *base, pdb_adc_trigger_channel_t channel, pdb_adc_pretrigger_t pretriggerNumber, uint32_t value)
  Sets the value for the ADC pre-trigger delay event.
- static uint32_t PDB_GetADCPreTriggerStatusFlags (PDB_Type *base, pdb_adc_trigger_channel_t channel)
  Gets the ADC pre-trigger's status flags.
- static void PDB_ClearADCPreTriggerStatusFlags (PDB_Type *base, pdb_adc_trigger_channel_t channel, uint32_t mask)
  Clears the ADC pre-trigger status flags.

Pulse-Out Trigger

- static void PDB_EnablePulseOutTrigger (PDB_Type *base, pdb_pulse_out_channel_mask_t channelMask, bool enable)
  Enables the pulse out trigger channels.
- static void PDB_SetPulseOutTriggerDelayValue (PDB_Type *base, pdb_pulse_out_trigger_channel_t channel, uint32_t value1, uint32_t value2)
  Sets event values for the pulse out trigger.

18.3 Data Structure Documentation

18.3.1 struct pdb_config_t

Data Fields

- pdb_load_value_mode_t loadValueMode
  Select the load value mode.
- pdb_prescaler_divider_t prescalerDivider
  Select the prescaler divider.
- pdb_divider_multiplication_factor_t dividerMultiplicationFactor
  Multiplication factor select for prescaler.
- pdb_trigger_input_source_t triggerInputSource
  Select the trigger input source.
- bool enableContinuousMode
  Enable the PDB operation in Continuous mode.

Field Documentation

(1) pdb_load_value_mode_t pdb_config_t::loadValueMode

(2) pdb_prescaler_divider_t pdb_config_t::prescalerDivider

(3) pdb_divider_multiplication_factor_t pdb_config_t::dividerMultiplicationFactor

(4) pdb_trigger_input_source_t pdb_config_t::triggerInputSource

(5) bool pdb_config_t::enableContinuousMode

18.3.2 struct pdb_adc_pretrigger_config_t

Data Fields

- uint32_t enablePreTriggerMask
  PDB Channel Pre-trigger Enable.
- uint32_t enableOutputMask
  PDB Channel Pre-trigger Output Select.
- uint32_t enableBackToBackOperationMask
  PDB Channel pre-trigger Back-to-Back Operation Enable.

Field Documentation

(1) uint32_t pdb_adc_pretrigger_config_t::enablePreTriggerMask

(2) uint32_t pdb_adc_pretrigger_config_t::enableOutputMask

PDB channel's corresponding pre-trigger asserts when the counter reaches the channel delay register.

(3) uint32_t pdb_adc_pretrigger_config_t::enableBackToBackOperationMask

Back-to-back operation enables the ADC conversions complete to trigger the next PDB channel pre-trigger and trigger output, so that the ADC conversions can be triggered on next set of configuration and results registers.

18.3.3 struct pdb_dac_trigger_config_t

Data Fields

- bool enableExternalTriggerInput
  Enables the external trigger for DAC interval counter.
- bool enableIntervalTrigger
  Enables the DAC interval trigger.

Field Documentation

(1) bool pdb_dac_trigger_config_t::enableExternalTriggerInput

(2) bool pdb_dac_trigger_config_t::enableIntervalTrigger

18.4 Macro Definition Documentation

18.4.1 #define FSL_PDB_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

18.5 Enumeration Type Documentation

18.5.1 enum _pdb_status_flags

Enumerator

kPDB_LoadOKFlag This flag is automatically cleared when the values in buffers are loaded into the internal registers after the LDOK bit is set or the PDBEN is cleared.
kPDB_DelayEventFlag PDB timer delay event flag.

18.5.2 enum _pdb_adc_pretrigger_flags

Enumerator

kPDB_ADCPreTriggerChannel0Flag Pre-trigger 0 flag.
kPDB_ADCPreTriggerChannel1Flag Pre-trigger 1 flag.
kPDB_ADCPreTriggerChannel2Flag Pre-trigger 2 flag.
kPDB_ADCPreTriggerChannel3Flag Pre-trigger 3 flag.
kPDB_ADCPreTriggerChannel0ErrorFlag Pre-trigger 0 Error.
kPDB_ADCPreTriggerChannel1ErrorFlag Pre-trigger 1 Error.
kPDB_ADCPreTriggerChannel2ErrorFlag Pre-trigger 2 Error.
kPDB_ADCPreTriggerChannel3ErrorFlag Pre-trigger 3 Error.

18.5.3 enum _pdb_interrupt_enable

Enumerator

kPDB_SequenceErrorInterruptEnable PDB sequence error interrupt enable.
kPDB_DelayInterruptEnable PDB delay interrupt enable.

18.5.4 enum pdb_load_value_mode_t

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx_MOD, PDBx_IDLY)
- ADC trigger (PDBx_CHnDLYm)
- DAC trigger (PDBx_DACINTx)
- CMP trigger (PDBx_POyDLY)

Enumerator

kPDB_LoadValueImmediately Load immediately after 1 is written to LDOK.

kPDB_LoadValueOnCounterOverflow Load when the PDB counter overflows (reaches the MOD register value).
kPDB_LoadValueOnTriggerInput Load a trigger input event is detected.
kPDB_LoadValueOnCounterOverflowOrTriggerInput Load either when the PDB counter overflows or a trigger input is detected.

18.5.5 enum pdb_prescaler_divider_t

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

Enumerator

kPDB_PrescalerDivider1 Divider x1.
kPDB_PrescalerDivider2 Divider x2.
kPDB_PrescalerDivider4 Divider x4.
kPDB_PrescalerDivider8 Divider x8.
kPDB_PrescalerDivider16 Divider x16.
kPDB_PrescalerDivider32 Divider x32.
kPDB_PrescalerDivider64 Divider x64.
kPDB_PrescalerDivider128 Divider x128.

18.5.6 enum pdb_divider_multiplication_factor_t

Selects the multiplication factor of the prescaler divider for the counter clock.

Enumerator

kPDB_DividerMultiplicationFactor1 Multiplication factor is 1.
kPDB_DividerMultiplicationFactor10 Multiplication factor is 10.
kPDB_DividerMultiplicationFactor20 Multiplication factor is 20.
kPDB_DividerMultiplicationFactor40 Multiplication factor is 40.

18.5.7 enum pdb_trigger_input_source_t

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

Enumerator

kPDB_TriggerInput0 Trigger-In 0.
kPDB_TriggerInput1 Trigger-In 1.

kPDB_TriggerInput2 Trigger-In 2.
kPDB_TriggerInput3 Trigger-In 3.
kPDB_TriggerInput4 Trigger-In 4.
kPDB_TriggerInput5 Trigger-In 5.
kPDB_TriggerInput6 Trigger-In 6.
kPDB_TriggerInput7 Trigger-In 7.
kPDB_TriggerInput8 Trigger-In 8.
kPDB_TriggerInput9 Trigger-In 9.
kPDB_TriggerInput10 Trigger-In 10.
kPDB_TriggerInput11 Trigger-In 11.
kPDB_TriggerInput12 Trigger-In 12.
kPDB_TriggerInput13 Trigger-In 13.
kPDB_TriggerInput14 Trigger-In 14.
kPDB_TriggerSoftware Trigger-In 15, software trigger.

18.5.8 enum pdb_adc_trigger_channel_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPDB_ADCTriggerChannel0 PDB ADC trigger channel number 0.
kPDB_ADCTriggerChannel1 PDB ADC trigger channel number 1.
kPDB_ADCTriggerChannel2 PDB ADC trigger channel number 2.
kPDB_ADCTriggerChannel3 PDB ADC trigger channel number 3.

18.5.9 enum pdb_adc_pretrigger_t

Note

Actual number of available pretrigger channels is SoC dependent

Enumerator

kPDB_ADCPreTrigger0 PDB ADC pretrigger number 0.
kPDB_ADCPreTrigger1 PDB ADC pretrigger number 1.
kPDB_ADCPreTrigger2 PDB ADC pretrigger number 2.
kPDB_ADCPreTrigger3 PDB ADC pretrigger number 3.
kPDB_ADCPreTrigger4 PDB ADC pretrigger number 4.
kPDB_ADCPreTrigger5 PDB ADC pretrigger number 5.
kPDB_ADCPreTrigger6 PDB ADC pretrigger number 6.
kPDB_ADCPreTrigger7 PDB ADC pretrigger number 7.

18.5.10 enum pdb_dac_trigger_channel_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPDB_DACTriggerChannel0 PDB DAC trigger channel number 0.
kPDB_DACTriggerChannel1 PDB DAC trigger channel number 1.

18.5.11 enum pdb_pulse_out_trigger_channel_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPDB_PulseOutTriggerChannel0 PDB pulse out trigger channel number 0.
kPDB_PulseOutTriggerChannel1 PDB pulse out trigger channel number 1.
kPDB_PulseOutTriggerChannel2 PDB pulse out trigger channel number 2.
kPDB_PulseOutTriggerChannel3 PDB pulse out trigger channel number 3.

18.5.12 enum pdb_pulse_out_channel_mask_t

Note

Actual number of available channels mask is SoC dependent

Enumerator

kPDB_PulseOutChannel0Mask PDB pulse out trigger channel number 0 mask.
kPDB_PulseOutChannel1Mask PDB pulse out trigger channel number 1 mask.
kPDB_PulseOutChannel2Mask PDB pulse out trigger channel number 2 mask.
kPDB_PulseOutChannel3Mask PDB pulse out trigger channel number 3 mask.

18.6 Function Documentation

18.6.1 void PDB_Init (PDB_Type * *base*, const pdb_config_t * *config*)

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

- Enable the clock for PDB instance.
- Configure the PDB module.
- Enable the PDB module.

Parameters

base	PDB peripheral base address.
config	Pointer to the configuration structure. See "pdb_config_t".

18.6.2 void PDB_Deinit (PDB_Type * *base*)

Parameters

base	PDB peripheral base address.

18.6.3 void PDB_GetDefaultConfig (pdb_config_t * *config*)

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

```
*   config->loadValueMode = kPDB_LoadValueImmediately;
*   config->prescalerDivider = kPDB_PrescalerDivider1;
*   config->dividerMultiplicationFactor = kPDB_DividerMultiplicationFactor1
     ;
*   config->triggerInputSource = kPDB_TriggerSoftware;
*   config->enableContinuousMode = false;
*
```

Parameters

config	Pointer to configuration structure. See "pdb_config_t".

18.6.4 static void PDB_Enable (PDB_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	PDB peripheral base address.
enable	Enable the module or not.

18.6.5 static void PDB_DoSoftwareTrigger (PDB_Type * *base*) [inline], [static]

Parameters

base	PDB peripheral base address.

18.6.6 static void PDB_DoLoadValues (PDB_Type * *base*) [inline], [static]

This function loads the counter values from the internal buffer. See "pdb_load_value_mode_t" about PDB's load mode.

Parameters

base	PDB peripheral base address.

18.6.7 static void PDB_EnableDMA (PDB_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	PDB peripheral base address.
enable	Enable the feature or not.

18.6.8 static void PDB_EnableInterrupts (PDB_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
mask	Mask value for interrupts. See "_pdb_interrupt_enable".

18.6.9 static void PDB_DisableInterrupts (PDB_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
mask	Mask value for interrupts. See "_pdb_interrupt_enable".

18.6.10 static uint32_t PDB_GetStatusFlags (PDB_Type * *base*) [inline], [static]

Parameters

base	PDB peripheral base address.

Returns

Mask value for asserted flags. See "_pdb_status_flags".

18.6.11 static void PDB_ClearStatusFlags (PDB_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
mask	Mask value of flags. See "_pdb_status_flags".

18.6.12 static void PDB_SetModulusValue (PDB_Type * *base*, uint32_t *value*) [inline], [static]

Parameters

base	PDB peripheral base address.
value	Setting value for the modulus. 16-bit is available.

18.6.13 static uint32_t PDB_GetCounterValue (PDB_Type * *base*) [inline], [static]

Parameters

base	PDB peripheral base address.

Returns

PDB counter's current value.

18.6.14 static void PDB_SetCounterDelayValue (PDB_Type * *base*, uint32_t *value*) [inline], [static]

Parameters

base	PDB peripheral base address.
value	Setting value for PDB counter delay event. 16-bit is available.

18.6.15 static void PDB_SetADCPreTriggerConfig (PDB_Type * *base*, pdb_adc_trigger_channel_t *channel*, pdb_adc_pretrigger_config_t * *config*) [inline], [static]

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.
config	Pointer to the configuration structure. See "pdb_adc_pretrigger_config_t".

18.6.16 static void PDB_SetADCPreTriggerDelayValue (PDB_Type * *base*, pdb_adc_trigger_channel_t *channel*, pdb_adc_pretrigger_t *pretriggerNumber*, uint32_t *value*) [inline], [static]

This function sets the value for ADC pre-trigger delay event. It specifies the delay value for the channel's corresponding pre-trigger. The pre-trigger asserts when the PDB counter is equal to the set value.

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.
pretrigger-Number	Channel group index for ADC instance.
value	Setting value for ADC pre-trigger delay event. 16-bit is available.

18.6.17 static uint32_t PDB_GetADCPreTriggerStatusFlags (PDB_Type * *base*, pdb_adc_trigger_channel_t *channel*) [inline], [static]

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.

Returns

Mask value for asserted flags. See "_pdb_adc_pretrigger_flags".

18.6.18 static void PDB_ClearADCPreTriggerStatusFlags (PDB_Type * *base*, pdb_adc_trigger_channel_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.
mask	Mask value for flags. See "_pdb_adc_pretrigger_flags".

18.6.19 static void PDB_EnablePulseOutTrigger (PDB_Type * *base*, pdb_pulse_out_channel_mask_t *channelMask*, bool *enable*) [inline], [static]

Parameters

base	PDB peripheral base address.
channelMask	Channel mask value for multiple pulse out trigger channel.
enable	Whether the feature is enabled or not.

18.6.20 static void PDB_SetPulseOutTriggerDelayValue (PDB_Type ∗ *base*, pdb_pulse_out_trigger_channel_t *channel*, uint32_t *value1*, uint32_t *value2*) [inline], [static]

This function is used to set event values for the pulse output trigger. These pulse output trigger delay values specify the delay for the PDB Pulse-out. Pulse-out goes high when the PDB counter is equal to the pulse output high value (value1). Pulse-out goes low when the PDB counter is equal to the pulse output low value (value2).

Parameters

base	PDB peripheral base address.
channel	Channel index for pulse out trigger channel.
value1	Setting value for pulse out high.
value2	Setting value for pulse out low.

Chapter 19
PMC: Power Management Controller

19.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Power Management Controller (PMC) module of MCUXpresso SDK devices. The PMC module contains internal voltage regulator, power on reset, low-voltage detect system, and high-voltage detect system.

Data Structures

- struct pmc_low_volt_detect_config_t
  Low-voltage Detect Configuration Structure. More...
- struct pmc_low_volt_warning_config_t
  Low-voltage Warning Configuration Structure. More...

Driver version

- #define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  PMC driver version.

Power Management Controller Control APIs

- void PMC_ConfigureLowVoltDetect (PMC_Type ∗base, const pmc_low_volt_detect_config_t ∗config)
  Configures the low-voltage detect setting.
- static bool PMC_GetLowVoltDetectFlag (PMC_Type ∗base)
  Gets the Low-voltage Detect Flag status.
- static void PMC_ClearLowVoltDetectFlag (PMC_Type ∗base)
  Acknowledges clearing the Low-voltage Detect flag.
- void PMC_ConfigureLowVoltWarning (PMC_Type ∗base, const pmc_low_volt_warning_config_t ∗config)
  Configures the low-voltage warning setting.
- static bool PMC_GetLowVoltWarningFlag (PMC_Type ∗base)
  Gets the Low-voltage Warning Flag status.
- static void PMC_ClearLowVoltWarningFlag (PMC_Type ∗base)
  Acknowledges the Low-voltage Warning flag.

19.2 Data Structure Documentation

19.2.1 struct pmc_low_volt_detect_config_t

Data Fields

- bool enableInt

Enable interrupt when Low-voltage detect.
- bool enableReset
  Enable system reset when Low-voltage detect.

19.2.2 struct pmc_low_volt_warning_config_t

Data Fields

- bool enableInt
  Enable interrupt when low-voltage warning.

19.3 Macro Definition Documentation

19.3.1 #define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Version 2.0.3.

19.4 Function Documentation

19.4.1 void PMC_ConfigureLowVoltDetect (PMC_Type * *base*, const pmc_low_volt_detect_config_t * *config*)

This function configures the low-voltage detect setting, including the trip point voltage setting, enables or disables the interrupt, enables or disables the system reset.

Parameters

base	PMC peripheral base address.
config	Low-voltage detect configuration structure.

19.4.2 static bool PMC_GetLowVoltDetectFlag (PMC_Type * *base*) [inline], [static]

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

Parameters

base	PMC peripheral base address.

Returns

Current low-voltage detect flag
- true: Low-voltage detected
- false: Low-voltage not detected

19.4.3 static void PMC_ClearLowVoltDetectFlag (PMC_Type * *base*) [inline], [static]

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

Parameters

base	PMC peripheral base address.

19.4.4 void PMC_ConfigureLowVoltWarning (PMC_Type * *base*, const pmc_low_volt_warning_config_t * *config*)

This function configures the low-voltage warning setting, including the trip point voltage setting and enabling or disabling the interrupt.

Parameters

base	PMC peripheral base address.
config	Low-voltage warning configuration structure.

19.4.5 static bool PMC_GetLowVoltWarningFlag (PMC_Type * *base*) [inline], [static]

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

Parameters

base	PMC peripheral base address.

Returns

Current LVWF status
- true: Low-voltage Warning Flag is set.
- false: the Low-voltage Warning does not happen.

19.4.6 static void PMC_ClearLowVoltWarningFlag (PMC_Type * base) [inline], [static]

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

base	PMC peripheral base address.

Chapter 20
PORT: Port Control and Interrupts

20.1 Overview

The MCUXpresso SDK provides a driver for the Port Control and Interrupts (PORT) module of MCU-Xpresso SDK devices.

Data Structures

- struct port_digital_filter_config_t
  PORT digital filter feature configuration definition. More...
- struct port_pin_config_t
  PORT pin configuration structure. More...

Enumerations

- enum _port_pull {
  kPORT_PullDisable = 0U,
  kPORT_PullDown = 2U,
  kPORT_PullUp = 3U }
  Internal resistor pull feature selection.
- enum _port_passive_filter_enable {
  kPORT_PassiveFilterDisable = 0U,
  kPORT_PassiveFilterEnable = 1U }
  Passive filter feature enable/disable.
- enum _port_drive_strength {
  kPORT_LowDriveStrength = 0U,
  kPORT_HighDriveStrength = 1U }
  Configures the drive strength.
- enum _port_lock_register {
  kPORT_UnlockRegister = 0U,
  kPORT_LockRegister = 1U }
  Unlock/lock the pin control register field[15:0].
- enum port_mux_t {

kPORT_PinDisabledOrAnalog = 0U,
kPORT_MuxAsGpio = 1U,
kPORT_MuxAlt2 = 2U,
kPORT_MuxAlt3 = 3U,
kPORT_MuxAlt4 = 4U,
kPORT_MuxAlt5 = 5U,
kPORT_MuxAlt6 = 6U,
kPORT_MuxAlt7 = 7U,
kPORT_MuxAlt8 = 8U,
kPORT_MuxAlt9 = 9U,
kPORT_MuxAlt10 = 10U,
kPORT_MuxAlt11 = 11U,
kPORT_MuxAlt12 = 12U,
kPORT_MuxAlt13 = 13U,
kPORT_MuxAlt14 = 14U,
kPORT_MuxAlt15 = 15U }
Pin mux selection.

- enum port_interrupt_t {
kPORT_InterruptOrDMADisabled = 0x0U,
kPORT_DMARisingEdge = 0x1U,
kPORT_DMAFallingEdge = 0x2U,
kPORT_DMAEitherEdge = 0x3U,
kPORT_FlagRisingEdge = 0x05U,
kPORT_FlagFallingEdge = 0x06U,
kPORT_FlagEitherEdge = 0x07U,
kPORT_InterruptLogicZero = 0x8U,
kPORT_InterruptRisingEdge = 0x9U,
kPORT_InterruptFallingEdge = 0xAU,
kPORT_InterruptEitherEdge = 0xBU,
kPORT_InterruptLogicOne = 0xCU,
kPORT_ActiveHighTriggerOutputEnable = 0xDU,
kPORT_ActiveLowTriggerOutputEnable = 0xEU }
Configures the interrupt generation condition.
- enum port_digital_filter_clock_source_t {
kPORT_BusClock = 0U,
kPORT_LpoClock = 1U }
Digital filter clock source selection.

Driver version

- #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
PORT driver version.

Configuration

- static void PORT_SetPinConfig (PORT_Type ∗base, uint32_t pin, const port_pin_config_t ∗config)

Sets the port PCR register.
- static void PORT_SetMultiplePinsConfig (PORT_Type ∗base, uint32_t mask, const port_pin_-config_t ∗config)
  Sets the port PCR register for multiple pins.
- static void PORT_SetPinMux (PORT_Type ∗base, uint32_t pin, port_mux_t mux)
  Configures the pin muxing.
- static void PORT_EnablePinsDigitalFilter (PORT_Type ∗base, uint32_t mask, bool enable)
  Enables the digital filter in one port, each bit of the 32-bit register represents one pin.
- static void PORT_SetDigitalFilterConfig (PORT_Type ∗base, const port_digital_filter_config_-t ∗config)
  Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Interrupt

- static void PORT_SetPinInterruptConfig (PORT_Type ∗base, uint32_t pin, port_interrupt_t config)
  Configures the port pin interrupt/DMA request.
- static void PORT_SetPinDriveStrength (PORT_Type ∗base, uint32_t pin, uint8_t strength)
  Configures the port pin drive strength.
- static uint32_t PORT_GetPinsInterruptFlags (PORT_Type ∗base)
  Reads the whole port status flag.
- static void PORT_ClearPinsInterruptFlags (PORT_Type ∗base, uint32_t mask)
  Clears the multiple pin interrupt status flag.

20.2 Data Structure Documentation

20.2.1 struct port_digital_filter_config_t

Data Fields

- uint32_t digitalFilterWidth
  Set digital filter width.
- port_digital_filter_clock_source_t clockSource
  Set digital filter clockSource.

20.2.2 struct port_pin_config_t

Data Fields

- uint16_t pullSelect: 2
  No-pull/pull-down/pull-up select.
- uint16_t passiveFilterEnable: 1
  Passive filter enable/disable.
- uint16_t driveStrength: 1
  Fast/slow drive strength configure.
- uint16_t mux: 3
  Pin mux Configure.
- uint16_t lockRegister: 1
  Lock/unlock the PCR field[15:0].

20.3 Macro Definition Documentation

20.3.1 #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

20.4 Enumeration Type Documentation

20.4.1 enum _port_pull

Enumerator

kPORT_PullDisable Internal pull-up/down resistor is disabled.
kPORT_PullDown Internal pull-down resistor is enabled.
kPORT_PullUp Internal pull-up resistor is enabled.

20.4.2 enum _port_passive_filter_enable

Enumerator

kPORT_PassiveFilterDisable Passive input filter is disabled.
kPORT_PassiveFilterEnable Passive input filter is enabled.

20.4.3 enum _port_drive_strength

Enumerator

kPORT_LowDriveStrength Low-drive strength is configured.
kPORT_HighDriveStrength High-drive strength is configured.

20.4.4 enum _port_lock_register

Enumerator

kPORT_UnlockRegister Pin Control Register fields [15:0] are not locked.
kPORT_LockRegister Pin Control Register fields [15:0] are locked.

20.4.5 enum port_mux_t

Enumerator

kPORT_PinDisabledOrAnalog Corresponding pin is disabled, but is used as an analog pin.
kPORT_MuxAsGpio Corresponding pin is configured as GPIO.

kPORT_MuxAlt2 Chip-specific.
kPORT_MuxAlt3 Chip-specific.
kPORT_MuxAlt4 Chip-specific.
kPORT_MuxAlt5 Chip-specific.
kPORT_MuxAlt6 Chip-specific.
kPORT_MuxAlt7 Chip-specific.
kPORT_MuxAlt8 Chip-specific.
kPORT_MuxAlt9 Chip-specific.
kPORT_MuxAlt10 Chip-specific.
kPORT_MuxAlt11 Chip-specific.
kPORT_MuxAlt12 Chip-specific.
kPORT_MuxAlt13 Chip-specific.
kPORT_MuxAlt14 Chip-specific.
kPORT_MuxAlt15 Chip-specific.

20.4.6 enum port_interrupt_t

Enumerator

kPORT_InterruptOrDMADisabled Interrupt/DMA request is disabled.
kPORT_DMARisingEdge DMA request on rising edge.
kPORT_DMAFallingEdge DMA request on falling edge.
kPORT_DMAEitherEdge DMA request on either edge.
kPORT_FlagRisingEdge Flag sets on rising edge.
kPORT_FlagFallingEdge Flag sets on falling edge.
kPORT_FlagEitherEdge Flag sets on either edge.
kPORT_InterruptLogicZero Interrupt when logic zero.
kPORT_InterruptRisingEdge Interrupt on rising edge.
kPORT_InterruptFallingEdge Interrupt on falling edge.
kPORT_InterruptEitherEdge Interrupt on either edge.
kPORT_InterruptLogicOne Interrupt when logic one.
kPORT_ActiveHighTriggerOutputEnable Enable active high-trigger output.
kPORT_ActiveLowTriggerOutputEnable Enable active low-trigger output.

20.4.7 enum port_digital_filter_clock_source_t

Enumerator

kPORT_BusClock Digital filters are clocked by the bus clock.
kPORT_LpoClock Digital filters are clocked by the 1 kHz LPO clock.

20.5 Function Documentation

20.5.1 static void PORT_SetPinConfig (PORT_Type * *base*, uint32_t *pin*, const port_pin_config_t * *config*) [inline], [static]

This is an example to define an input pin or output pin PCR configuration.

```
* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
*      kPORT_PullUp,
*      kPORT_FastSlewRate,
*      kPORT_PassiveFilterDisable,
*      kPORT_OpenDrainDisable,
*      kPORT_LowDriveStrength,
*      kPORT_MuxAsGpio,
*      kPORT_UnLockRegister,
* };
*
```

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
config	PORT PCR register configuration structure.

20.5.2 static void PORT_SetMultiplePinsConfig (PORT_Type * *base*, uint32_t *mask*, const port_pin_config_t * *config*) [inline], [static]

This is an example to define input pins or output pins PCR configuration.

```
* Define a digital input pin PCR configuration
* port_pin_config_t config = {
*      kPORT_PullUp ,
*      kPORT_PullEnable,
*      kPORT_FastSlewRate,
*      kPORT_PassiveFilterDisable,
*      kPORT_OpenDrainDisable,
*      kPORT_LowDriveStrength,
*      kPORT_MuxAsGpio,
*      kPORT_UnlockRegister,
* };
*
```

Parameters

base	PORT peripheral base pointer.
mask	PORT pin number macro.
config	PORT PCR register configuration structure.

20.5.3 static void PORT_SetPinMux (PORT_Type * *base*, uint32_t *pin*, port_mux_t *mux*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
mux	pin muxing slot selection. • kPORT_PinDisabledOrAnalog: Pin disabled or work in analog function. • kPORT_MuxAsGpio : Set as GPIO. • kPORT_MuxAlt2 : chip-specific. • kPORT_MuxAlt3 : chip-specific. • kPORT_MuxAlt4 : chip-specific. • kPORT_MuxAlt5 : chip-specific. • kPORT_MuxAlt6 : chip-specific. • kPORT_MuxAlt7 : chip-specific.

Note

: This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

20.5.4 static void PORT_EnablePinsDigitalFilter (PORT_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

base	PORT peripheral base pointer.

mask	PORT pin number macro.
enable	PORT digital filter configuration.

20.5.5 static void PORT_SetDigitalFilterConfig (PORT_Type * *base*, const port_digital_filter_config_t * *config*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
config	PORT digital filter configuration structure.

20.5.6 static void PORT_SetPinInterruptConfig (PORT_Type * *base*, uint32_t *pin*, port_interrupt_t *config*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
config	PORT pin interrupt configuration. • kPORT_InterruptOrDMADisabled: Interrupt/DMA request disabled. • kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit). • kPORT_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit). • kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit). • kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit). • kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit). • kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit). • kPORT_InterruptLogicZero : Interrupt when logic zero. • kPORT_InterruptRisingEdge : Interrupt on rising edge. • kPORT_InterruptFallingEdge: Interrupt on falling edge. • kPORT_InterruptEitherEdge : Interrupt on either edge. • kPORT_InterruptLogicOne : Interrupt when logic one. • kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit). • kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

20.5.7 static void PORT_SetPinDriveStrength (PORT_Type * *base*, uint32_t *pin*, uint8_t *strength*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
strength	PORT pin drive strength • kPORT_LowDriveStrength = 0U - Low-drive strength is configured. • kPORT_HighDriveStrength = 1U - High-drive strength is configured.

20.5.8 static uint32_t PORT_GetPinsInterruptFlags (PORT_Type * *base*) [inline], [static]

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

base	PORT peripheral base pointer.

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

20.5.9 static void PORT_ClearPinsInterruptFlags (PORT_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
mask	PORT pin number macro.

Chapter 21
PWT: Pulse Width Timer

21.1 Overview

The MCUXpresso SDK provides a driver for the Pulse Width Timer (PWT) of MCUXpresso SDK devices.

21.2 Function groups

The PWT driver supports capture or measure the pulse width mapping on its input channels. The counter of PWT has two selectable clock sources, and supports up to BUS_CLK with internal timer clock. PWT module supports programmable positive or negative pulse edges, and programmable interrupt generation upon pulse width values or counter overflow.

21.2.1 Initialization and deinitialization

The function PWT_Init() initializes the PWT with specified configurations. The function PWT_GetDefaultConfig() gets the default configurations. The initialization function configures the PWT for the requested register update mode for register with buffers.

The function PWT_Deinit() disables the PWT counter and turns off the module clock.

21.2.2 Reset

The function PWT_Reset() is built into PWT as a mechanism used to reset/restart the pulse width timer.

21.2.3 Status

Provides functions to get and clear the PWT status.

21.2.4 Interrupt

Provides functions to enable/disable PWT interrupts and get current enabled interrupts.

21.2.5 Start & Stop timer

The function PWT_StartTimer() starts the PWT time counter.

The function PWT_StopTimer() stops the PWT time counter.

21.2.6 GetInterrupt

Provides functions to generate Overflow/Pulse Width Data Ready Interrupt.

21.2.7 Get Timer value

The function PWT_GetCurrentTimerCount() is set to read the current counter value.

The function PWT_ReadPositivePulseWidth() is set to read the positive pulse width.

The function PWT_ReadNegativePulseWidth() is set to read the negative pulse width.

21.2.8 PWT Operations

Input capture operations

The input capture operations sets up an channel for input capture.

The function EdgeCapture can be used to measure the pulse width of a signal. A channel is used during capture with the input signal coming through a channel n. The capture edge for each channel, and any filter value to be used when processing the input signal.

21.3 Typical use case

21.3.1 PWT measure

This is an example code to measure the pulse width:

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

Data Structures

- struct pwt_config_t
  PWT configuration structure. More...

Macros

- #define FSL_PWT_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  Version 2.0.1.

Enumerations

- enum pwt_clock_source_t {
  kPWT_BusClock = 0U,
  kPWT_AlternativeClock }
  PWT clock source selection.

- enum pwt_clock_prescale_t {
  kPWT_Prescale_Divide_1 = 0U,
  kPWT_Prescale_Divide_2,
  kPWT_Prescale_Divide_4,
  kPWT_Prescale_Divide_8,
  kPWT_Prescale_Divide_16,
  kPWT_Prescale_Divide_32,
  kPWT_Prescale_Divide_64,
  kPWT_Prescale_Divide_128 }
  PWT prescaler factor selection for clock source.
- enum pwt_input_select_t {
  kPWT_InputPort_0 = 0U,
  kPWT_InputPort_1,
  kPWT_InputPort_2,
  kPWT_InputPort_3 }
  PWT input port selection.
- enum _pwt_interrupt_enable {
  kPWT_PulseWidthReadyInterruptEnable = PWT_CS_PRDYIE_MASK,
  kPWT_CounterOverflowInterruptEnable = PWT_CS_POVIE_MASK }
  List of PWT interrupts.
- enum _pwt_status_flags {
  kPWT_CounterOverflowFlag = PWT_CS_PWTOV_MASK,
  kPWT_PulseWidthValidFlag = PWT_CS_PWTRDY_MASK }
  List of PWT flags.

Functions

- static uint16_t PWT_GetCurrentTimerCount (PWT_Type *base)
  Reads the current counter value.
- static uint16_t PWT_ReadPositivePulseWidth (PWT_Type *base)
  Reads the positive pulse width.
- static uint16_t PWT_ReadNegativePulseWidth (PWT_Type *base)
  Reads the negative pulse width.
- static void PWT_Reset (PWT_Type *base)
  Performs a software reset on the PWT module.

Initialization and deinitialization

- void PWT_Init (PWT_Type *base, const pwt_config_t *config)
  Ungates the PWT clock and configures the peripheral for basic operation.
- void PWT_Deinit (PWT_Type *base)
  Gates the PWT clock.
- void PWT_GetDefaultConfig (pwt_config_t *config)
  Fills in the PWT configuration structure with the default settings.

Interrupt Interface

- static void PWT_EnableInterrupts (PWT_Type *base, uint32_t mask)

Enables the selected PWT interrupts.
- static void PWT_DisableInterrupts (PWT_Type ∗base, uint32_t mask)
  Disables the selected PWT interrupts.
- static uint32_t PWT_GetEnabledInterrupts (PWT_Type ∗base)
  Gets the enabled PWT interrupts.

Status Interface

- static uint32_t PWT_GetStatusFlags (PWT_Type ∗base)
  Gets the PWT status flags.
- static void PWT_ClearStatusFlags (PWT_Type ∗base, uint32_t mask)
  Clears the PWT status flags.

Timer Start and Stop

- static void PWT_StartTimer (PWT_Type ∗base)
  Starts the PWT counter.
- static void PWT_StopTimer (PWT_Type ∗base)
  Stops the PWT counter.

21.4 Data Structure Documentation

21.4.1 struct pwt_config_t

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

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

Data Fields

- pwt_clock_source_t clockSource
  Clock source for the counter.
- pwt_clock_prescale_t prescale
  Pre-scaler to divide down the clock.
- pwt_input_select_t inputSelect
  PWT Pulse input port selection.
- bool enableFirstCounterLoad
  true: Load the first counter value to registers; false: Do not load first counter value

21.5 Enumeration Type Documentation

21.5.1 enum pwt_clock_source_t

Enumerator

kPWT_BusClock The Bus clock is used as the clock source of PWT counter.
kPWT_AlternativeClock Alternative clock is used as the clock source of PWT counter.

21.5.2 enum pwt_clock_prescale_t

Enumerator

kPWT_Prescale_Divide_1 PWT clock divided by 1.
kPWT_Prescale_Divide_2 PWT clock divided by 2.
kPWT_Prescale_Divide_4 PWT clock divided by 4.
kPWT_Prescale_Divide_8 PWT clock divided by 8.
kPWT_Prescale_Divide_16 PWT clock divided by 16.
kPWT_Prescale_Divide_32 PWT clock divided by 32.
kPWT_Prescale_Divide_64 PWT clock divided by 64.
kPWT_Prescale_Divide_128 PWT clock divided by 128.

21.5.3 enum pwt_input_select_t

Enumerator

kPWT_InputPort_0 PWT input comes from PWTIN[0].
kPWT_InputPort_1 PWT input comes from PWTIN[1].
kPWT_InputPort_2 PWT input comes from PWTIN[2].
kPWT_InputPort_3 PWT input comes from PWTIN[3].

21.5.4 enum _pwt_interrupt_enable

Enumerator

kPWT_PulseWidthReadyInterruptEnable Pulse width data ready interrupt.
kPWT_CounterOverflowInterruptEnable Counter overflow interrupt.

21.5.5 enum _pwt_status_flags

Enumerator

kPWT_CounterOverflowFlag Counter overflow flag.
kPWT_PulseWidthValidFlag Pulse width valid flag.

21.6 Function Documentation

21.6.1 void PWT_Init (PWT_Type * *base*, const pwt_config_t * *config*)

Note

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

Parameters

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

21.6.2 void PWT_Deinit (PWT_Type ∗ *base*)

Parameters

base	PWT peripheral base address

21.6.3 void PWT_GetDefaultConfig (pwt_config_t ∗ *config*)

The default values are:

```
*   config->clockSource = kPWT_BusClock;
*   config->prescale = kPWT_Prescale_Divide_1;
*   config->inputSelect = kPWT_InputPort_0;
*   config->enableFirstCounterLoad = false;
*
```

Parameters

config	Pointer to the user configuration structure.

21.6.4 static void PWT_EnableInterrupts (PWT_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PWT peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration pwt_-interrupt_enable_t

21.6.5 static void PWT_DisableInterrupts (PWT_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PWT peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration pwt_-interrupt_enable_t

21.6.6 static uint32_t PWT_GetEnabledInterrupts (PWT_Type * *base*) [inline], [static]

Parameters

base	PWT peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration pwt_interrupt_enable-_t

21.6.7 static uint32_t PWT_GetStatusFlags (PWT_Type * *base*) [inline], [static]

Parameters

base	PWT peripheral base address

Returns

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

21.6.8 static void PWT_ClearStatusFlags (PWT_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	PWT peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration pwt_-status_flags_t

21.6.9 static void PWT_StartTimer (PWT_Type ∗ *base*) [inline], [static]

Parameters

base	PWT peripheral base address

21.6.10 static void PWT_StopTimer (PWT_Type ∗ *base*) [inline], [static]

Parameters

base	PWT peripheral base address

21.6.11 static uint16_t PWT_GetCurrentTimerCount (PWT_Type ∗ *base*) [inline], [static]

This function returns the timer counting value

Parameters

base	PWT peripheral base address

Returns

Current 16-bit timer counter value

21.6.12 static uint16_t PWT_ReadPositivePulseWidth (PWT_Type ∗ *base*) [inline], [static]

This function reads the low and high registers and returns the 16-bit positive pulse width

Parameters

base	PWT peripheral base address.

Returns

The 16-bit positive pulse width.

21.6.13 static uint16_t PWT_ReadNegativePulseWidth (PWT_Type * *base*) [inline], [static]

This function reads the low and high registers and returns the 16-bit negative pulse width

Parameters

base	PWT peripheral base address.

Returns

The 16-bit negative pulse width.

21.6.14 static void PWT_Reset (PWT_Type * *base*) [inline], [static]

Parameters

base	PWT peripheral base address

Chapter 22
RCM: Reset Control Module Driver

22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Reset Control Module (RCM) module of MCUXpresso SDK devices.

Data Structures

- struct rcm_version_id_t
  IP version ID definition. More...
- struct rcm_reset_pin_filter_config_t
  Reset pin filter configuration. More...

Enumerations

- enum rcm_reset_source_t {
  kRCM_SourceLvd = RCM_SRS_LVD_MASK,
  kRCM_SourceLoc = RCM_SRS_LOC_MASK,
  kRCM_SourceLol = RCM_SRS_LOL_MASK,
  kRCM_SourceWdog = RCM_SRS_WDOG_MASK,
  kRCM_SourcePin = RCM_SRS_PIN_MASK,
  kRCM_SourcePor = RCM_SRS_POR_MASK,
  kRCM_SourceLockup = RCM_SRS_LOCKUP_MASK,
  kRCM_SourceSw = RCM_SRS_SW_MASK,
  kRCM_SourceMdmap = RCM_SRS_MDM_AP_MASK,
  kRCM_SourceSackerr = RCM_SRS_SACKERR_MASK }
  System Reset Source Name definitions.
- enum rcm_run_wait_filter_mode_t {
  kRCM_FilterDisable = 0U,
  kRCM_FilterBusClock = 1U,
  kRCM_FilterLpoClock = 2U }
  Reset pin filter select in Run and Wait modes.
- enum rcm_boot_rom_config_t {
  kRCM_BootFlash = 0U,
  kRCM_BootRomCfg0 = 1U,
  kRCM_BootRomFopt = 2U,
  kRCM_BootRomBoth = 3U }
  Boot from ROM configuration.
- enum rcm_reset_delay_t {
  kRCM_ResetDelay8Lpo = 0U,
  kRCM_ResetDelay32Lpo = 1U,
  kRCM_ResetDelay128Lpo = 2U,

kRCM_ResetDelay512Lpo = 3U }
Maximum delay time from interrupt asserts to system reset.
- enum rcm_interrupt_enable_t {
kRCM_IntNone = 0U,
kRCM_IntLossOfClk = RCM_SRIE_LOC_MASK,
kRCM_IntLossOfLock = RCM_SRIE_LOL_MASK,
kRCM_IntWatchDog = RCM_SRIE_WDOG_MASK,
kRCM_IntExternalPin = RCM_SRIE_PIN_MASK,
kRCM_IntGlobal = RCM_SRIE_GIE_MASK,
kRCM_IntCoreLockup = RCM_SRIE_LOCKUP_MASK,
kRCM_IntSoftware = RCM_SRIE_SW_MASK,
kRCM_IntStopModeAckErr = RCM_SRIE_SACKERR_MASK,
kRCM_IntAll }
System reset interrupt enable bit definitions.

Driver version

- #define FSL_RCM_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
RCM driver version 2.0.4.

Reset Control Module APIs

- static void RCM_GetVersionId (RCM_Type ∗base, rcm_version_id_t ∗versionId)
Gets the RCM version ID.
- static uint32_t RCM_GetPreviousResetSources (RCM_Type ∗base)
Gets the reset source status which caused a previous reset.
- static uint32_t RCM_GetStickyResetSources (RCM_Type ∗base)
Gets the sticky reset source status.
- static void RCM_ClearStickyResetSources (RCM_Type ∗base, uint32_t sourceMasks)
Clears the sticky reset source status.
- void RCM_ConfigureResetPinFilter (RCM_Type ∗base, const rcm_reset_pin_filter_config_t ∗config)
Configures the reset pin filter.
- static rcm_boot_rom_config_t RCM_GetBootRomSource (RCM_Type ∗base)
Gets the ROM boot source.
- static void RCM_ClearBootRomSource (RCM_Type ∗base)
Clears the ROM boot source flag.
- void RCM_SetForceBootRomSource (RCM_Type ∗base, rcm_boot_rom_config_t config)
Forces the boot from ROM.
- static void RCM_SetSystemResetInterruptConfig (RCM_Type ∗base, uint32_t intMask, rcm_reset_delay_t delay)
Sets the system reset interrupt configuration.

22.2 Data Structure Documentation

22.2.1 struct rcm_version_id_t

Data Fields

- uint16_t feature
  Feature Specification Number.
- uint8_t minor
  Minor version number.
- uint8_t major
  Major version number.

Field Documentation

(1) uint16_t rcm_version_id_t::feature

(2) uint8_t rcm_version_id_t::minor

(3) uint8_t rcm_version_id_t::major

22.2.2 struct rcm_reset_pin_filter_config_t

Data Fields

- bool enableFilterInStop
  Reset pin filter select in stop mode.
- rcm_run_wait_filter_mode_t filterInRunWait
  Reset pin filter in run/wait mode.
- uint8_t busClockFilterCount
  Reset pin bus clock filter width.

Field Documentation

(1) bool rcm_reset_pin_filter_config_t::enableFilterInStop

(2) rcm_run_wait_filter_mode_t rcm_reset_pin_filter_config_t::filterInRunWait

(3) uint8_t rcm_reset_pin_filter_config_t::busClockFilterCount

22.3 Macro Definition Documentation

22.3.1 #define FSL_RCM_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

22.4 Enumeration Type Documentation

22.4.1 enum rcm_reset_source_t

Enumerator

kRCM_SourceLvd Low-voltage detect reset.

kRCM_SourceLoc Loss of clock reset.
kRCM_SourceLol Loss of lock reset.
kRCM_SourceWdog Watchdog reset.
kRCM_SourcePin External pin reset.
kRCM_SourcePor Power on reset.
kRCM_SourceLockup Core lock up reset.
kRCM_SourceSw Software reset.
kRCM_SourceMdmap MDM-AP system reset.
kRCM_SourceSackerr Parameter could get all reset flags.

22.4.2 enum rcm_run_wait_filter_mode_t

Enumerator

kRCM_FilterDisable All filtering disabled.
kRCM_FilterBusClock Bus clock filter enabled.
kRCM_FilterLpoClock LPO clock filter enabled.

22.4.3 enum rcm_boot_rom_config_t

Enumerator

kRCM_BootFlash Boot from flash.
kRCM_BootRomCfg0 Boot from boot ROM due to BOOTCFG0.
kRCM_BootRomFopt Boot from boot ROM due to FOPT[7].
kRCM_BootRomBoth Boot from boot ROM due to both BOOTCFG0 and FOPT[7].

22.4.4 enum rcm_reset_delay_t

Enumerator

kRCM_ResetDelay8Lpo Delay 8 LPO cycles.
kRCM_ResetDelay32Lpo Delay 32 LPO cycles.
kRCM_ResetDelay128Lpo Delay 128 LPO cycles.
kRCM_ResetDelay512Lpo Delay 512 LPO cycles.

22.4.5 enum rcm_interrupt_enable_t

Enumerator

kRCM_IntNone No interrupt enabled.

kRCM_IntLossOfClk Loss of clock interrupt.
kRCM_IntLossOfLock Loss of lock interrupt.
kRCM_IntWatchDog Watch dog interrupt.
kRCM_IntExternalPin External pin interrupt.
kRCM_IntGlobal Global interrupts.
kRCM_IntCoreLockup Core lock up interrupt.
kRCM_IntSoftware software interrupt
kRCM_IntStopModeAckErr Stop mode ACK error interrupt.
kRCM_IntAll Enable all interrupts.

22.5 Function Documentation

22.5.1 static void RCM_GetVersionId (RCM_Type * *base*, rcm_version_id_t * *versionId*) [inline], [static]

This function gets the RCM version ID including the major version number, the minor version number, and the feature specification number.

Parameters

base	RCM peripheral base address.
versionId	Pointer to the version ID structure.

22.5.2 static uint32_t RCM_GetPreviousResetSources (RCM_Type * *base*) [inline], [static]

This function gets the current reset source status. Use source masks defined in the rcm_reset_source_t to get the desired source status.

This is an example.

```
* uint32_t resetStatus;
*
* To get all reset source statuses.
* resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;
*
* To test whether the MCU is reset using Watchdog.
* resetStatus = RCM_GetPreviousResetSources(RCM) &
     kRCM_SourceWdog;
*
* To test multiple reset sources.
* resetStatus = RCM_GetPreviousResetSources(RCM) & (
     kRCM_SourceWdog | kRCM_SourcePin);
*
```

Parameters

base	RCM peripheral base address.

Returns

All reset source status bit map.

22.5.3 static uint32_t RCM_GetStickyResetSources (RCM_Type * *base*) [inline], [static]

This function gets the current reset source status that has not been cleared by software for a specific source.

This is an example.

```
* uint32_t resetStatus;
*
* To get all reset source statuses.
* resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceAll;
*
* To test whether the MCU is reset using Watchdog.
* resetStatus = RCM_GetStickyResetSources(RCM) &
      kRCM_SourceWdog;
*
* To test multiple reset sources.
* resetStatus = RCM_GetStickyResetSources(RCM) & (
      kRCM_SourceWdog | kRCM_SourcePin);
*
```

Parameters

base	RCM peripheral base address.

Returns

All reset source status bit map.

22.5.4 static void RCM_ClearStickyResetSources (RCM_Type * *base*, uint32_t *sourceMasks*) [inline], [static]

This function clears the sticky system reset flags indicated by source masks.

This is an example.

```
* Clears multiple reset sources.
* RCM_ClearStickyResetSources(kRCM_SourceWdog |
      kRCM_SourcePin);
*
```

Parameters

base	RCM peripheral base address.
sourceMasks	reset source status bit map

22.5.5 void RCM_ConfigureResetPinFilter (RCM_Type * *base*, const rcm_reset_pin_filter_config_t * *config*)

This function sets the reset pin filter including the filter source, filter width, and so on.

Parameters

base	RCM peripheral base address.
config	Pointer to the configuration structure.

22.5.6 static rcm_boot_rom_config_t RCM_GetBootRomSource (RCM_Type * *base*) [inline], [static]

This function gets the ROM boot source during the last chip reset.

Parameters

base	RCM peripheral base address.

Returns

The ROM boot source.

22.5.7 static void RCM_ClearBootRomSource (RCM_Type * *base*) [inline], [static]

This function clears the ROM boot source flag.

Parameters

base	Register base address of RCM

22.5.8 void RCM_SetForceBootRomSource (RCM_Type ∗ *base*, rcm_boot_rom_config_t *config*)

This function forces booting from ROM during all subsequent system resets.

Parameters

base	RCM peripheral base address.
config	Boot configuration.

22.5.9 static void RCM_SetSystemResetInterruptConfig (RCM_Type ∗ *base*, uint32_t *intMask*, rcm_reset_delay_t *delay*) [inline], [static]

For a graceful shut down, the RCM supports delaying the assertion of the system reset for a period of time when the reset interrupt is generated. This function can be used to enable the interrupt and the delay period. The interrupts are passed in as bit mask. See rcm_int_t for details. For example, to delay a reset for 512 LPO cycles after the WDOG timeout or loss-of-clock occurs, configure as follows: RCM_SetSystemResetInterruptConfig(kRCM_IntWatchDog | kRCM_IntLossOfClk, kRCM_ResetDelay512Lpo);

Parameters

base	RCM peripheral base address.
intMask	Bit mask of the system reset interrupts to enable. See rcm_interrupt_enable_t for details.
delay	Bit mask of the system reset interrupts to enable.

Chapter 23
RTC: Real Time Clock

23.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC) of MCUXpresso SDK devices.

23.2 Function groups

The RTC driver supports operating the module as a time counter.

23.2.1 Initialization and deinitialization

The function RTC_Init() initializes the RTC with specified configurations. The function RTC_GetDefaultConfig() gets the default configurations.

The function RTC_Deinit() disables the RTC timer and disables the module clock.

23.2.2 Set & Get Datetime

The function RTC_SetDatetime() sets the timer period in seconds. Users pass 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.

23.2.3 Set & Get Alarm

The function RTC_SetAlarm() sets the alarm time period in seconds. Users pass 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.

23.2.4 Start & Stop timer

The function RTC_StartTimer() starts the RTC time counter.

The function RTC_StopTimer() stops the RTC time counter.

23.2.5 Status

Provides functions to get and clear the RTC status.

23.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

23.2.7 RTC Oscillator

Some SoC's allow control of the RTC oscillator through the RTC module.

The function RTC_SetOscCapLoad() allows the user to modify the capacitor load configuration of the RTC oscillator.

23.2.8 Monotonic Counter

Some SoC's have a 64-bit Monotonic counter available in the RTC module.

The function RTC_SetMonotonicCounter() writes a 64-bit to the counter.

The function RTC_GetMonotonicCounter() reads the monotonic counter and returns the 64-bit counter value to the user.

The function RTC_IncrementMonotonicCounter() increments the Monotonic Counter by one.

23.3 Typical use case

23.3.1 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

Data Structures

- struct rtc_datetime_t
  Structure is used to hold the date and time. More...
- struct rtc_config_t
  RTC config structure. More...

Enumerations

- enum rtc_interrupt_enable_t {
  kRTC_TimeInvalidInterruptEnable = (1U << 0U),
  kRTC_TimeOverflowInterruptEnable = (1U << 1U),
  kRTC_AlarmInterruptEnable = (1U << 2U),

kRTC_SecondsInterruptEnable = (1U << 3U) }
List of RTC interrupts.
- enum rtc_status_flags_t {
kRTC_TimeInvalidFlag = (1U << 0U),
kRTC_TimeOverflowFlag = (1U << 1U),
kRTC_AlarmFlag = (1U << 2U) }
List of RTC flags.

Functions

- static void RTC_SetClockSource (RTC_Type ∗base)
  Set RTC clock source.
- 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, 2, 1))
  Version 2.2.1.

Initialization and deinitialization

- void RTC_Init (RTC_Type ∗base, const rtc_config_t ∗config)
  Ungates the RTC clock and configures the peripheral for basic operation.
- static void RTC_Deinit (RTC_Type ∗base)
  Stops the timer and gate the RTC clock.
- void RTC_GetDefaultConfig (rtc_config_t ∗config)
  Fills in the RTC config struct with the default settings.

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

- void RTC_EnableInterrupts (RTC_Type ∗base, uint32_t mask)
  Enables the selected RTC interrupts.
- void RTC_DisableInterrupts (RTC_Type ∗base, uint32_t mask)
  Disables the selected RTC interrupts.
- uint32_t RTC_GetEnabledInterrupts (RTC_Type ∗base)
  Gets the enabled RTC interrupts.

Status Interface

- uint32_t RTC_GetStatusFlags (RTC_Type *base)
  Gets the RTC status flags.
- 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.

23.4 Data Structure Documentation

23.4.1 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.

Field Documentation

(1) uint16_t rtc_datetime_t::year

(2) uint8_t rtc_datetime_t::month

(3) uint8_t rtc_datetime_t::day

(4) uint8_t rtc_datetime_t::hour

(5) uint8_t rtc_datetime_t::minute

(6) uint8_t rtc_datetime_t::second

23.4.2 struct rtc_config_t

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

The config struct can be made const so it resides in flash

Data Fields

- bool wakeupSelect
  true: Wakeup pin outputs the 32 KHz clock; false:Wakeup pin used to wakeup the chip
- bool updateMode
  true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked
- bool supervisorAccess
  true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported
- uint32_t compensationInterval
  Compensation interval that is written to the CIR field in RTC TCR Register.
- uint32_t compensationTime
  Compensation time that is written to the TCR field in RTC TCR Register.

23.5 Enumeration Type Documentation

23.5.1 enum rtc_interrupt_enable_t

Enumerator

kRTC_TimeInvalidInterruptEnable Time invalid interrupt.
kRTC_TimeOverflowInterruptEnable Time overflow interrupt.
kRTC_AlarmInterruptEnable Alarm interrupt.
kRTC_SecondsInterruptEnable Seconds interrupt.

23.5.2 enum rtc_status_flags_t

Enumerator

kRTC_TimeInvalidFlag Time invalid flag.
kRTC_TimeOverflowFlag Time overflow flag.
kRTC_AlarmFlag Alarm flag.

23.6 Function Documentation

23.6.1 void RTC_Init (RTC_Type ∗ *base*, const rtc_config_t ∗ *config*)

This function issues a software reset if the timer invalid flag is set.

Note

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

Parameters

base	RTC peripheral base address
config	Pointer to the user's RTC configuration structure.

23.6.2 static void RTC_Deinit (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

23.6.3 void RTC_GetDefaultConfig (rtc_config_t * *config*)

The default values are as follows.

```
*    config->wakeupSelect = false;
*    config->updateMode = false;
*    config->supervisorAccess = false;
*    config->compensationInterval = 0;
*    config->compensationTime = 0;
*
```

Parameters

config	Pointer to the user's RTC configuration structure.

23.6.4 status_t RTC_SetDatetime (RTC_Type * *base*, const rtc_datetime_t * *datetime*)

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

base	RTC peripheral base address
datetime	Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

23.6.5 void RTC_GetDatetime (RTC_Type ∗ *base*, rtc_datetime_t ∗ *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to the structure where the date and time details are stored.

23.6.6 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 the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

23.6.7 void RTC_GetAlarm (RTC_Type ∗ *base*, rtc_datetime_t ∗ *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to the structure where the alarm date and time details are stored.

23.6.8 void RTC_EnableInterrupts (RTC_Type ∗ *base,* uint32_t *mask*)

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

23.6.9 void RTC_DisableInterrupts (RTC_Type ∗ *base,* uint32_t *mask*)

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

23.6.10 uint32_t RTC_GetEnabledInterrupts (RTC_Type ∗ *base*)

Parameters

base	RTC peripheral base address

Returns

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

23.6.11 uint32_t RTC_GetStatusFlags (RTC_Type ∗ *base*)

Parameters

base	RTC peripheral base address

Returns

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

23.6.12 void RTC_ClearStatusFlags (RTC_Type * *base,* uint32_t *mask*)

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

23.6.13 static void RTC_SetClockSource (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Note

After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

23.6.14 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

23.6.15 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

23.6.16 static void RTC_Reset (RTC_Type * *base*) [inline], [static]

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

base	RTC peripheral base address

Chapter 24
SIM: System Integration Module Driver

24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Integration Module (SIM) of MCU-Xpresso SDK devices.

Data Structures

- struct sim_uid_t
  Unique ID. More...

Enumerations

- enum _sim_flash_mode {
  kSIM_FlashDisableInWait = SIM_FCFG1_FLASHDOZE_MASK,
  kSIM_FlashDisable = SIM_FCFG1_FLASHDIS_MASK }
  Flash enable mode.

Functions

- void SIM_GetUniqueId (sim_uid_t ∗uid)
  Gets the unique identification register value.
- static void SIM_SetFlashMode (uint8_t mode)
  Sets the flash enable mode.

Driver version

- #define **FSL_SIM_DRIVER_VERSION** (MAKE_VERSION(2, 1, 3))

24.2 Data Structure Documentation

24.2.1 struct sim_uid_t

Data Fields

- uint32_t MH
  UIDMH.
- uint32_t ML
  UIDML.
- uint32_t L
  UIDL.

Field Documentation

(1) uint32_t sim_uid_t::MH

(2) uint32_t sim_uid_t::ML

(3) uint32_t sim_uid_t::L

24.3 Enumeration Type Documentation

24.3.1 enum _sim_flash_mode

Enumerator

kSIM_FlashDisableInWait Disable flash in wait mode.
kSIM_FlashDisable Disable flash in normal mode.

24.4 Function Documentation

24.4.1 void SIM_GetUniqueId (sim_uid_t * *uid*)

Parameters

uid	Pointer to the structure to save the UID value.

24.4.2 static void SIM_SetFlashMode (uint8_t *mode*) [inline], [static]

Parameters

mode	The mode to set; see _sim_flash_mode for mode details.

Chapter 25
SMC: System Mode Controller Driver

25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Mode Controller (SMC) module of MCUXpresso SDK devices. The SMC module sequences the system in and out of all low-power stop and run modes.

API functions are provided to configure the system for working in a dedicated power mode. For different power modes, SMC_SetPowerModexxx() function accepts different parameters. System power mode state transitions are not available between power modes. For details about available transitions, see the power mode transitions section in the SoC reference manual.

25.2 Typical use case

25.2.1 Enter wait or stop modes

SMC driver provides APIs to set MCU to different wait modes and stop modes. Pre and post functions are used for setting the modes. The pre functions and post functions are used as follows.

Disable/enable the interrupt through PRIMASK. This is an example use case. The application sets the wakeup interrupt and calls SMC function SMC_SetPowerModeStop to set the MCU to STOP mode, but the wakeup interrupt happens so quickly that the ISR completes before the function SMC_SetPowerModeStop. As a result, the MCU enters the STOP mode and never is woken up by the interrupt. In this use case, the application first disables the interrupt through PRIMASK, sets the wakeup interrupt, and enters the STOP mode. After wakeup, enable the interrupt through PRIMASK. The MCU can still be woken up by disabling the interrupt through PRIMASK. The pre and post functions handle the PRIMASK.

```
SMC_PreEnterStopModes();

/* Enable the wakeup interrupt here. */

SMC_SetPowerModeStop(SMC, kSMC_PartialStop);

SMC_PostExitStopModes();
```

For legacy Kinetis, when entering stop modes, the flash speculation might be interrupted. As a result, the prefetched code or data might be broken. To make sure the flash is idle when entring the stop modes, smc driver allocates a RAM region, the code to enter stop modes are excuted in RAM, thus the flash is idle and no prefetch is performed while entring stop modes. Application should make sure that, the rw data of fsl_smc.c is located in memory region which is not powered off in stop modes, especially LLS2 modes.

For STOP, VLPS, and LLS3, the whole RAM are powered up, so after woken up, the RAM function could continue excuting. For VLLS mode, the system resets after woken up, the RAM content might be re-initialized. For LLS2 mode, only part of RAM are powered on, so application must make sure that, the rw data of fsl_smc.c is located in memory region which is not powered off, otherwise after woken up, the MCU could not get right code to excute.

Data Structures

- struct smc_version_id_t
  IP version ID definition. More...
- struct smc_param_t
  IP parameter definition. More...

Enumerations

- enum smc_power_mode_protection_t {
  kSMC_AllowPowerModeVlp = SMC_PMPROT_AVLP_MASK,
  kSMC_AllowPowerModeAll }
  Power Modes Protection.
- enum smc_power_state_t {
  kSMC_PowerStateRun = 0x01U << 0U,
  kSMC_PowerStateStop = 0x01U << 1U,
  kSMC_PowerStateVlpr = 0x01U << 2U,
  kSMC_PowerStateVlpw = 0x01U << 3U,
  kSMC_PowerStateVlps = 0x01U << 4U }
  Power Modes in PMSTAT.
- enum smc_run_mode_t {
  kSMC_RunNormal = 0U,
  kSMC_RunVlpr = 2U }
  Run mode definition.
- enum smc_stop_mode_t {
  kSMC_StopNormal = 0U,
  kSMC_StopVlps = 2U }
  Stop mode definition.
- enum smc_partial_stop_option_t {
  kSMC_PartialStop = 0U,
  kSMC_PartialStop1 = 1U,
  kSMC_PartialStop2 = 2U }
  Partial STOP option.
- enum { kStatus_SMC_StopAbort = MAKE_STATUS(kStatusGroup_POWER, 0) }
  _smc_status, SMC configuration status.

Driver version

- #define FSL_SMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))
  SMC driver version.

System mode controller APIs

- static void SMC_GetVersionId (SMC_Type ∗base, smc_version_id_t ∗versionId)
  Gets the SMC version ID.

- void SMC_GetParam (SMC_Type ∗base, smc_param_t ∗param)
  Gets the SMC parameter.
- static void SMC_SetPowerModeProtection (SMC_Type ∗base, uint8_t allowedModes)
  Configures all power mode protection settings.
- static smc_power_state_t SMC_GetPowerModeState (SMC_Type ∗base)
  Gets the current power mode status.
- void SMC_PreEnterStopModes (void)
  Prepares to enter stop modes.
- void SMC_PostExitStopModes (void)
  Recovers after wake up from stop modes.
- void SMC_PreEnterWaitModes (void)
  Prepares to enter wait modes.
- void SMC_PostExitWaitModes (void)
  Recovers after wake up from stop modes.
- status_t SMC_SetPowerModeRun (SMC_Type ∗base)
  Configures the system to RUN power mode.
- status_t SMC_SetPowerModeWait (SMC_Type ∗base)
  Configures the system to WAIT power mode.
- status_t SMC_SetPowerModeStop (SMC_Type ∗base, smc_partial_stop_option_t option)
  Configures the system to Stop power mode.
- status_t SMC_SetPowerModeVlpr (SMC_Type ∗base)
  Configures the system to VLPR power mode.
- status_t SMC_SetPowerModeVlpw (SMC_Type ∗base)
  Configures the system to VLPW power mode.
- status_t SMC_SetPowerModeVlps (SMC_Type ∗base)
  Configures the system to VLPS power mode.

25.3 Data Structure Documentation

25.3.1 struct smc_version_id_t

Data Fields

- uint16_t feature
  Feature Specification Number.
- uint8_t minor
  Minor version number.
- uint8_t major
  Major version number.

Field Documentation

(1) uint16_t smc_version_id_t::feature

(2) uint8_t smc_version_id_t::minor

(3) uint8_t smc_version_id_t::major

25.3.2 struct smc_param_t

Data Fields

- bool hsrunEnable
  HSRUN mode enable.
- bool llsEnable
  LLS mode enable.
- bool lls2Enable
  LLS2 mode enable.
- bool vlls0Enable
  VLLS0 mode enable.

Field Documentation

(1) bool smc_param_t::hsrunEnable

(2) bool smc_param_t::llsEnable

(3) bool smc_param_t::lls2Enable

(4) bool smc_param_t::vlls0Enable

25.4 Enumeration Type Documentation

25.4.1 enum smc_power_mode_protection_t

Enumerator

kSMC_AllowPowerModeVlp Allow Very-Low-power Mode.
kSMC_AllowPowerModeAll Allow all power mode.

25.4.2 enum smc_power_state_t

Enumerator

kSMC_PowerStateRun 0000_0001 - Current power mode is RUN
kSMC_PowerStateStop 0000_0010 - Current power mode is STOP
kSMC_PowerStateVlpr 0000_0100 - Current power mode is VLPR
kSMC_PowerStateVlpw 0000_1000 - Current power mode is VLPW
kSMC_PowerStateVlps 0001_0000 - Current power mode is VLPS

25.4.3 enum smc_run_mode_t

Enumerator

kSMC_RunNormal Normal RUN mode.

kSMC_RunVlpr Very-low-power RUN mode.

25.4.4 enum smc_stop_mode_t

Enumerator

kSMC_StopNormal Normal STOP mode.
kSMC_StopVlps Very-low-power STOP mode.

25.4.5 enum smc_partial_stop_option_t

Enumerator

kSMC_PartialStop STOP - Normal Stop mode.
kSMC_PartialStop1 Partial Stop with both system and bus clocks disabled.
kSMC_PartialStop2 Partial Stop with system clock disabled and bus clock enabled.

25.4.6 anonymous enum

Enumerator

kStatus_SMC_StopAbort Entering Stop mode is abort.

25.5 Function Documentation

25.5.1 static void SMC_GetVersionId (SMC_Type * *base*, smc_version_id_t * *versionId*) [inline], [static]

This function gets the SMC version ID, including major version number, minor version number, and feature specification number.

Parameters

base	SMC peripheral base address.
versionId	Pointer to the version ID structure.

25.5.2 void SMC_GetParam (SMC_Type * *base*, smc_param_t * *param*)

This function gets the SMC parameter including the enabled power mdoes.

Parameters

base	SMC peripheral base address.
param	Pointer to the SMC param structure.

25.5.3 static void SMC_SetPowerModeProtection (SMC_Type * *base,* uint8_t *allowedModes*) [inline], [static]

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the smc_power_mode_protection_t. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map. For example, to allow LLS and VLLS, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeVlls | kSMC_AllowPowerModeVlps). To allow all modes, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeAll).

Parameters

base	SMC peripheral base address.
allowedModes	Bitmap of the allowed power modes.

25.5.4 static smc_power_state_t SMC_GetPowerModeState (SMC_Type * *base*) [inline], [static]

This function returns the current power mode status. After the application switches the power mode, it should always check the status to check whether it runs into the specified mode or not. The application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the smc_power_state_t for information about the power status.

Parameters

base	SMC peripheral base address.

Returns

Current power mode status.

25.5.5 void SMC_PreEnterStopModes (void)

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

25.5.6 void SMC_PostExitStopModes (void)

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with SMC-_PreEnterStopModes.

25.5.7 void SMC_PreEnterWaitModes (void)

This function should be called before entering WAIT/VLPW modes.

25.5.8 void SMC_PostExitWaitModes (void)

This function should be called after wake up from WAIT/VLPW modes. It is used with SMC_PreEnter-WaitModes.

25.5.9 status_t SMC_SetPowerModeRun (SMC_Type ∗ *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

25.5.10 status_t SMC_SetPowerModeWait (SMC_Type ∗ *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

25.5.11 status_t SMC_SetPowerModeStop (SMC_Type * *base*, smc_partial_stop_option_t *option*)

Parameters

base	SMC peripheral base address.
option	Partial Stop mode option.

Returns

SMC configuration error code.

25.5.12 status_t SMC_SetPowerModeVlpr (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

25.5.13 status_t SMC_SetPowerModeVlpw (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

25.5.14 status_t SMC_SetPowerModeVlps (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

Chapter 26
TRGMUX: Trigger Mux Driver

26.1 Overview

The MCUXpresso SDK provides driver for the Trigger Mux (TRGMUX) module of MCUXpresso SDK devices.

26.2 Typical use case

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

Enumerations

- enum { kStatus_TRGMUX_Locked = MAKE_STATUS(kStatusGroup_TRGMUX, 0) }
  TRGMUX configure status.
- enum trgmux_trigger_input_t {
  kTRGMUX_TriggerInput0 = TRGMUX_TRGCFG_SEL0_SHIFT,
  kTRGMUX_TriggerInput1 = TRGMUX_TRGCFG_SEL1_SHIFT,
  kTRGMUX_TriggerInput2 = TRGMUX_TRGCFG_SEL2_SHIFT,
  kTRGMUX_TriggerInput3 = TRGMUX_TRGCFG_SEL3_SHIFT }
  Defines the MUX select for peripheral trigger input.

Driver version

- #define FSL_TRGMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  TRGMUX driver version.

TRGMUX Functional Operation

- static void TRGMUX_LockRegister (TRGMUX_Type ∗base, uint32_t index)
  Sets the flag of the register which is used to mark writeable.
- status_t TRGMUX_SetTriggerSource (TRGMUX_Type ∗base, uint32_t index, trgmux_trigger_input_t input, uint32_t trigger_src)
  Configures the trigger source of the appointed peripheral.

26.3 Macro Definition Documentation

26.3.1 #define FSL_TRGMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

26.4 Enumeration Type Documentation

26.4.1 anonymous enum

Enumerator

kStatus_TRGMUX_Locked Configure failed for register is locked.

26.4.2 enum trgmux_trigger_input_t

Enumerator

kTRGMUX_TriggerInput0 The MUX select for peripheral trigger input 0.
kTRGMUX_TriggerInput1 The MUX select for peripheral trigger input 1.
kTRGMUX_TriggerInput2 The MUX select for peripheral trigger input 2.
kTRGMUX_TriggerInput3 The MUX select for peripheral trigger input 3.

26.5 Function Documentation

26.5.1 static void TRGMUX_LockRegister (TRGMUX_Type * *base*, uint32_t *index*) [inline], [static]

The function sets the flag of the register which is used to mark writeable. Example:

```
TRGMUX_LockRegister(TRGMUX0,kTRGMUX_Trgmux0Dmamux0);
```

Parameters

base	TRGMUX peripheral base address.
index	The index of the TRGMUX register, see the enum trgmux_device_t defined in <SO-C>.h.

26.5.2 status_t TRGMUX_SetTriggerSource (TRGMUX_Type * *base*, uint32_t *index*, trgmux_trigger_input_t *input*, uint32_t *trigger_src*)

The function configures the trigger source of the appointed peripheral. Example:

```
TRGMUX_SetTriggerSource(TRGMUX0, kTRGMUX_Trgmux0Dmamux0,
      kTRGMUX_TriggerInput0, kTRGMUX_SourcePortPin);
```

Parameters

base	TRGMUX peripheral base address.
index	The index of the TRGMUX register, see the enum trgmux_device_t defined in <SO-C>.h.
input	The MUX select for peripheral trigger input
trigger_src	The trigger inputs for various peripherals. See the enum trgmux_source_t defined in <SOC>.h.

Return values

kStatus_Success	Configured successfully.
kStatus_TRGMUX_-Locked	Configuration failed because the register is locked.

Chapter 27
WDOG32: 32-bit Watchdog Timer

27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the WDOG32 module of MCUXpresso SDK devices.

27.2 Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/wdog32

Data Structures

- struct wdog32_work_mode_t
  Defines WDOG32 work mode. More...
- struct wdog32_config_t
  Describes WDOG32 configuration structure. More...

Enumerations

- enum wdog32_clock_source_t {
  kWDOG32_ClockSource0 = 0U,
  kWDOG32_ClockSource1 = 1U,
  kWDOG32_ClockSource2 = 2U,
  kWDOG32_ClockSource3 = 3U }
  Describes WDOG32 clock source.
- enum wdog32_clock_prescaler_t {
  kWDOG32_ClockPrescalerDivide1 = 0x0U,
  kWDOG32_ClockPrescalerDivide256 = 0x1U }
  Describes the selection of the clock prescaler.
- enum wdog32_test_mode_t {
  kWDOG32_TestModeDisabled = 0U,
  kWDOG32_UserModeEnabled = 1U,
  kWDOG32_LowByteTest = 2U,
  kWDOG32_HighByteTest = 3U }
  Describes WDOG32 test mode.
- enum _wdog32_interrupt_enable_t { kWDOG32_InterruptEnable = WDOG_CS_INT_MASK }
  WDOG32 interrupt configuration structure.
- enum _wdog32_status_flags_t {
  kWDOG32_RunningFlag = WDOG_CS_EN_MASK,
  kWDOG32_InterruptFlag = WDOG_CS_FLG_MASK }
  WDOG32 status flags.

Unlock sequence

- #define WDOG_FIRST_WORD_OF_UNLOCK (WDOG_UPDATE_KEY & 0xFFFFU)
  First word of unlock sequence.
- #define WDOG_SECOND_WORD_OF_UNLOCK ((WDOG_UPDATE_KEY >> 16U) & 0xFFFFU)
  Second word of unlock sequence.

Refresh sequence

- #define WDOG_FIRST_WORD_OF_REFRESH (WDOG_REFRESH_KEY & 0xFFFFU)
  First word of refresh sequence.
- #define WDOG_SECOND_WORD_OF_REFRESH ((WDOG_REFRESH_KEY >> 16U) & 0xFFFFU)
  Second word of refresh sequence.

Driver version

- #define FSL_WDOG32_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  WDOG32 driver version.

WDOG32 Initialization and De-initialization

- void WDOG32_GetDefaultConfig (wdog32_config_t ∗config)
  Initializes the WDOG32 configuration structure.
- void WDOG32_Init (WDOG_Type ∗base, const wdog32_config_t ∗config)
  Initializes the WDOG32 module.
- void WDOG32_Deinit (WDOG_Type ∗base)
  De-initializes the WDOG32 module.

WDOG32 functional Operation

- void WDOG32_Unlock (WDOG_Type ∗base)
  Unlocks the WDOG32 register written.
- void WDOG32_Enable (WDOG_Type ∗base)
  Enables the WDOG32 module.
- void WDOG32_Disable (WDOG_Type ∗base)
  Disables the WDOG32 module.
- void WDOG32_EnableInterrupts (WDOG_Type ∗base, uint32_t mask)
  Enables the WDOG32 interrupt.
- void WDOG32_DisableInterrupts (WDOG_Type ∗base, uint32_t mask)
  Disables the WDOG32 interrupt.
- static uint32_t WDOG32_GetStatusFlags (WDOG_Type ∗base)
  Gets the WDOG32 all status flags.
- void WDOG32_ClearStatusFlags (WDOG_Type ∗base, uint32_t mask)
  Clears the WDOG32 flag.
- void WDOG32_SetTimeoutValue (WDOG_Type ∗base, uint16_t timeoutCount)
  Sets the WDOG32 timeout value.
- void WDOG32_SetWindowValue (WDOG_Type ∗base, uint16_t windowValue)
  Sets the WDOG32 window value.
- static void WDOG32_Refresh (WDOG_Type ∗base)

Refreshes the WDOG32 timer.
- static uint16_t WDOG32_GetCounterValue (WDOG_Type *base)
  Gets the WDOG32 counter value.

27.3 Data Structure Documentation

27.3.1 struct wdog32_work_mode_t

Data Fields

- bool enableWait
  Enables or disables WDOG32 in wait mode.
- bool enableStop
  Enables or disables WDOG32 in stop mode.
- bool enableDebug
  Enables or disables WDOG32 in debug mode.

27.3.2 struct wdog32_config_t

Data Fields

- bool enableWdog32
  Enables or disables WDOG32.
- wdog32_clock_source_t clockSource
  Clock source select.
- wdog32_clock_prescaler_t prescaler
  Clock prescaler value.
- wdog32_work_mode_t workMode
  Configures WDOG32 work mode in debug stop and wait mode.
- wdog32_test_mode_t testMode
  Configures WDOG32 test mode.
- bool enableUpdate
  Update write-once register enable.
- bool enableInterrupt
  Enables or disables WDOG32 interrupt.
- bool enableWindowMode
  Enables or disables WDOG32 window mode.
- uint16_t windowValue
  Window value.
- uint16_t timeoutValue
  Timeout value.

27.4 Macro Definition Documentation

27.4.1 #define FSL_WDOG32_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

27.5 Enumeration Type Documentation

27.5.1 enum wdog32_clock_source_t

Enumerator

kWDOG32_ClockSource0 Clock source 0.
kWDOG32_ClockSource1 Clock source 1.
kWDOG32_ClockSource2 Clock source 2.
kWDOG32_ClockSource3 Clock source 3.

27.5.2 enum wdog32_clock_prescaler_t

Enumerator

kWDOG32_ClockPrescalerDivide1 Divided by 1.
kWDOG32_ClockPrescalerDivide256 Divided by 256.

27.5.3 enum wdog32_test_mode_t

Enumerator

kWDOG32_TestModeDisabled Test Mode disabled.
kWDOG32_UserModeEnabled User Mode enabled.
kWDOG32_LowByteTest Test Mode enabled, only low byte is used.
kWDOG32_HighByteTest Test Mode enabled, only high byte is used.

27.5.4 enum _wdog32_interrupt_enable_t

This structure contains the settings for all of the WDOG32 interrupt configurations.

Enumerator

kWDOG32_InterruptEnable Interrupt is generated before forcing a reset.

27.5.5 enum _wdog32_status_flags_t

This structure contains the WDOG32 status flags for use in the WDOG32 functions.

Enumerator

kWDOG32_RunningFlag Running flag, set when WDOG32 is enabled.
kWDOG32_InterruptFlag Interrupt flag, set when interrupt occurs.

27.6 Function Documentation

27.6.1 void WDOG32_GetDefaultConfig (wdog32_config_t * *config*)

This function initializes the WDOG32 configuration structure to default values. The default values are:

```
*   wdog32Config->enableWdog32 = true;
*   wdog32Config->clockSource = kWDOG32_ClockSource1;
*   wdog32Config->prescaler = kWDOG32_ClockPrescalerDivide1;
*   wdog32Config->workMode.enableWait = true;
*   wdog32Config->workMode.enableStop = false;
*   wdog32Config->workMode.enableDebug = false;
*   wdog32Config->testMode = kWDOG32_TestModeDisabled;
*   wdog32Config->enableUpdate = true;
*   wdog32Config->enableInterrupt = false;
*   wdog32Config->enableWindowMode = false;
*   wdog32Config->windowValue = 0U;
*   wdog32Config->timeoutValue = 0xFFFFU;
*
```

Parameters

config	Pointer to the WDOG32 configuration structure.

See Also

wdog32_config_t

27.6.2 void WDOG32_Init (WDOG_Type * *base*, const wdog32_config_t * *config*)

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, enable-Update must be set to true in the configuration.

Example:

```
*   wdog32_config_t config;
*   WDOG32_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   config.enableUpdate = true;
*   WDOG32_Init(wdog_base,&config);
*
```

Parameters

base	WDOG32 peripheral base address.
config	The configuration of the WDOG32.

27.6.3 void WDOG32_Deinit (WDOG_Type * *base*)

This function shuts down the WDOG32. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

base	WDOG32 peripheral base address.

27.6.4 void WDOG32_Unlock (WDOG_Type * *base*)

This function unlocks the WDOG32 register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

base	WDOG32 peripheral base address

27.6.5 void WDOG32_Enable (WDOG_Type * *base*)

This function writes a value into the WDOG_CS register to enable the WDOG32. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

base	WDOG32 peripheral base address.

27.6.6 void WDOG32_Disable (WDOG_Type * *base*)

This function writes a value into the WDOG_CS register to disable the WDOG32. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

base	WDOG32 peripheral base address

27.6.7 void WDOG32_EnableInterrupts (WDOG_Type * *base*, uint32_t *mask*)

This function writes a value into the WDOG_CS register to enable the WDOG32 interrupt. The WDOG_-CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

base	WDOG32 peripheral base address.
mask	The interrupts to enable. The parameter can be a combination of the following source if defined: • kWDOG32_InterruptEnable

27.6.8 void WDOG32_DisableInterrupts (WDOG_Type * *base*, uint32_t *mask*)

This function writes a value into the WDOG_CS register to disable the WDOG32 interrupt. The WDOG_-CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

base	WDOG32 peripheral base address.
mask	The interrupts to disabled. The parameter can be a combination of the following source if defined: • kWDOG32_InterruptEnable

27.6.9 static uint32_t WDOG32_GetStatusFlags (WDOG_Type * *base*) [inline], [static]

This function gets all status flags.

Example to get the running flag:

```
*   uint32_t status;
*   status = WDOG32_GetStatusFlags(wdog_base) &
      kWDOG32_RunningFlag;
*
```

Parameters

base	WDOG32 peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

_wdog32_status_flags_t
- true: related status flag has been set.
- false: related status flag is not set.

27.6.10 void WDOG32_ClearStatusFlags (WDOG_Type ∗ *base*, uint32_t *mask*)

This function clears the WDOG32 status flag.

Example to clear an interrupt flag:

```
*   WDOG32_ClearStatusFlags(wdog_base,
      kWDOG32_InterruptFlag);
*
```

Parameters

base	WDOG32 peripheral base address.
mask	The status flags to clear. The parameter can be any combination of the following values: • kWDOG32_InterruptFlag

27.6.11 void WDOG32_SetTimeoutValue (WDOG_Type ∗ *base*, uint16_t *timeoutCount*)

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. To ensure the reconfiguration fits the timing of WCT, unlock function will be called inline.

Parameters

base	WDOG32 peripheral base address
timeoutCount	WDOG32 timeout value, count of WDOG32 clock ticks.

27.6.12 void WDOG32_SetWindowValue (WDOG_Type ∗ *base,* uint16_t *windowValue*)

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

base	WDOG32 peripheral base address.
windowValue	WDOG32 window value.

27.6.13 static void WDOG32_Refresh (WDOG_Type ∗ *base*) [inline], [static]

This function feeds the WDOG32. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

base	WDOG32 peripheral base address

27.6.14 static uint16_t WDOG32_GetCounterValue (WDOG_Type ∗ *base*) [inline], [static]

This function gets the WDOG32 counter value.

Parameters

base	WDOG32 peripheral base address.

Returns

Current WDOG32 counter value.

Chapter 28
Clock Driver

28.1 Overview

The MCUXpresso SDK provides APIs for MCUXpresso SDK devices' clock operation.

Modules

- System Clock Generator (SCG)

Files

- file fsl_clock.h

Data Structures

- struct scg_sys_clk_config_t
  SCG system clock configuration. More...
- struct scg_sosc_config_t
  SCG system OSC configuration. More...
- struct scg_sirc_config_t
  SCG slow IRC clock configuration. More...
- struct scg_firc_trim_config_t
  SCG fast IRC clock trim configuration. More...
- struct scg_firc_config_t
  SCG fast IRC clock configuration. More...
- struct scg_lpfll_trim_config_t
  SCG LPFLL clock trim configuration. More...
- struct scg_lpfll_config_t
  SCG low power FLL configuration. More...

Macros

- #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0
  Configure whether driver controls clock.
- #define RTC_CLOCKS
  Clock ip name array for RTC.
- #define PORT_CLOCKS
  Clock ip name array for PORT.
- #define LPI2C_CLOCKS
  Clock ip name array for LPI2C.
- #define TSI_CLOCKS
  Clock ip name array for TSI.
- #define LPUART_CLOCKS
  Clock ip name array for LPUART.
- #define LPTMR_CLOCKS

Clock ip name array for LPTMR.
- #define ADC12_CLOCKS
  Clock ip name array for ADC12.
- #define LPSPI_CLOCKS
  Clock ip name array for LPSPI.
- #define LPIT_CLOCKS
  Clock ip name array for LPIT.
- #define CRC_CLOCKS
  Clock ip name array for CRC.
- #define CMP_CLOCKS
  Clock ip name array for CMP.
- #define FLASH_CLOCKS
  Clock ip name array for FLASH.
- #define EWM_CLOCKS
  Clock ip name array for EWM.
- #define FTM_CLOCKS
  Clock ip name array for FLEXTMR.
- #define PDB_CLOCKS
  Clock ip name array for PDB.
- #define PWT_CLOCKS
  Clock ip name array for PWT.
- #define MSCAN_CLOCKS
  Clock ip name array for MSCAN.
- #define LPO_CLK_FREQ 128000U
  LPO clock frequency.
- #define CLOCK_GetOsc0ErClkFreq CLOCK_GetErClkFreq
  For compatible with other MCG platforms.

Enumerations

- enum clock_name_t {
  kCLOCK_CoreSysClk,
  kCLOCK_BusClk,
  kCLOCK_FlashClk,
  kCLOCK_ScgSysOscClk,
  kCLOCK_ScgSircClk,
  kCLOCK_ScgFircClk,
  kCLOCK_ScgLpFllClk,
  kCLOCK_ScgSysOscAsyncDiv2Clk,
  kCLOCK_ScgSircAsyncDiv2Clk,
  kCLOCK_ScgFircAsyncDiv2Clk,
  kCLOCK_ScgLpFllAsyncDiv2Clk,
  kCLOCK_LpoClk,
  kCLOCK_ErClk }
  Clock name used to get clock frequency.
- enum clock_ip_src_t {

kCLOCK_IpSrcNoneOrExt = 0U,
kCLOCK_IpSrcSysOscAsync = 1U,
kCLOCK_IpSrcSircAsync = 2U,
kCLOCK_IpSrcFircAsync = 3U,
kCLOCK_IpSrcLpFllAsync = 5U }
Clock source for peripherals that support various clock selections.
- enum clock_ip_name_t
Peripheral clock name difinition used for clock gate, clock source and clock divider setting.
- enum {
kStatus_SCG_Busy = MAKE_STATUS(kStatusGroup_SCG, 1),
kStatus_SCG_InvalidSrc = MAKE_STATUS(kStatusGroup_SCG, 2) }
SCG status return codes.
- enum scg_sys_clk_t {
kSCG_SysClkSlow,
kSCG_SysClkCore }
SCG system clock type.
- enum scg_sys_clk_src_t {
kSCG_SysClkSrcSysOsc = 1U,
kSCG_SysClkSrcSirc = 2U,
kSCG_SysClkSrcFirc = 3U,
kSCG_SysClkSrcLpFll = 5U }
SCG system clock source.
- enum scg_sys_clk_div_t {
kSCG_SysClkDivBy1 = 0U,
kSCG_SysClkDivBy2 = 1U,
kSCG_SysClkDivBy3 = 2U,
kSCG_SysClkDivBy4 = 3U,
kSCG_SysClkDivBy5 = 4U,
kSCG_SysClkDivBy6 = 5U,
kSCG_SysClkDivBy7 = 6U,
kSCG_SysClkDivBy8 = 7U,
kSCG_SysClkDivBy9 = 8U,
kSCG_SysClkDivBy10 = 9U,
kSCG_SysClkDivBy11 = 10U,
kSCG_SysClkDivBy12 = 11U,
kSCG_SysClkDivBy13 = 12U,
kSCG_SysClkDivBy14 = 13U,
kSCG_SysClkDivBy15 = 14U,
kSCG_SysClkDivBy16 = 15U }
SCG system clock divider value.
- enum clock_clkout_src_t {
kClockClkoutSelScgSlow = 0U,
kClockClkoutSelSysOsc = 1U,
kClockClkoutSelSirc = 2U,
kClockClkoutSelFirc = 3U,
kClockClkoutSelLpFll = 5U }

SCG clock out configuration (CLKOUTSEL).
- enum scg_async_clk_t { kSCG_AsyncDiv2Clk }
  SCG asynchronous clock type.
- enum scg_async_clk_div_t {
  kSCG_AsyncClkDisable = 0U,
  kSCG_AsyncClkDivBy1 = 1U,
  kSCG_AsyncClkDivBy2 = 2U,
  kSCG_AsyncClkDivBy4 = 3U,
  kSCG_AsyncClkDivBy8 = 4U,
  kSCG_AsyncClkDivBy16 = 5U,
  kSCG_AsyncClkDivBy32 = 6U,
  kSCG_AsyncClkDivBy64 = 7U }
  SCG asynchronous clock divider value.
- enum scg_sosc_monitor_mode_t {
  kSCG_SysOscMonitorDisable = 0U,
  kSCG_SysOscMonitorInt = SCG_SOSCCSR_SOSCCM_MASK,
  kSCG_SysOscMonitorReset }
  SCG system OSC monitor mode.
- enum scg_sosc_mode_t {
  kSCG_SysOscModeExt = 0U,
  kSCG_SysOscModeOscLowPower = SCG_SOSCCFG_EREFS_MASK,
  kSCG_SysOscModeOscHighGain = SCG_SOSCCFG_EREFS_MASK | SCG_SOSCCFG_HGO_-MASK }
  OSC work mode.
- enum {
  kSCG_SysOscEnable = SCG_SOSCCSR_SOSCEN_MASK,
  kSCG_SysOscEnableInStop = SCG_SOSCCSR_SOSCSTEN_MASK,
  kSCG_SysOscEnableInLowPower = SCG_SOSCCSR_SOSCLPEN_MASK,
  kSCG_SysOscEnableErClk = SCG_SOSCCSR_SOSCERCLKEN_MASK }
  OSC enable mode.
- enum scg_sirc_range_t {
  kSCG_SircRangeLow,
  kSCG_SircRangeHigh }
  SCG slow IRC clock frequency range.
- enum {
  kSCG_SircEnable = SCG_SIRCCSR_SIRCEN_MASK,
  kSCG_SircEnableInStop = SCG_SIRCCSR_SIRCSTEN_MASK,
  kSCG_SircEnableInLowPower = SCG_SIRCCSR_SIRCLPEN_MASK }
  SIRC enable mode.
- enum scg_firc_trim_mode_t {
  kSCG_FircTrimNonUpdate = SCG_FIRCCSR_FIRCTREN_MASK,
  kSCG_FircTrimUpdate = SCG_FIRCCSR_FIRCTREN_MASK | SCG_FIRCCSR_FIRCTRUP_-MASK }
  SCG fast IRC trim mode.
- enum scg_firc_trim_div_t {

kSCG_FircTrimDivBy1,
kSCG_FircTrimDivBy128,
kSCG_FircTrimDivBy256,
kSCG_FircTrimDivBy512,
kSCG_FircTrimDivBy1024,
kSCG_FircTrimDivBy2048 }
SCG fast IRC trim predivided value for system OSC.
- enum scg_firc_trim_src_t { kSCG_FircTrimSrcSysOsc = 2U }
SCG fast IRC trim source.
- enum scg_firc_range_t { kSCG_FircRange48M }
SCG fast IRC clock frequency range.
- enum {
kSCG_FircEnable = SCG_FIRCCSR_FIRCEN_MASK,
kSCG_FircEnableInStop = SCG_FIRCCSR_FIRCSTEN_MASK,
kSCG_FircEnableInLowPower = SCG_FIRCCSR_FIRCLPEN_MASK,
kSCG_FircDisableRegulator = SCG_FIRCCSR_FIRCREGOFF_MASK }
FIRC enable mode.
- enum { kSCG_LpFllEnable = SCG_LPFLLCSR_LPFLLEN_MASK }
LPFLL enable mode.
- enum scg_lpfll_range_t { kSCG_LpFllRange48M }
SCG LPFLL clock frequency range.
- enum scg_lpfll_trim_mode_t {
kSCG_LpFllTrimNonUpdate = SCG_LPFLLCSR_LPFLLTREN_MASK,
kSCG_LpFllTrimUpdate = SCG_LPFLLCSR_LPFLLTREN_MASK | SCG_LPFLLCSR_LPFLL-TRUP_MASK }
SCG LPFLL trim mode.
- enum scg_lpfll_trim_src_t {
kSCG_LpFllTrimSrcSirc = 0U,
kSCG_LpFllTrimSrcFirc = 1U,
kSCG_LpFllTrimSrcSysOsc = 2U,
kSCG_LpFllTrimSrcRtcOsc = 3U }
SCG LPFLL trim source.
- enum scg_lpfll_lock_mode_t {
kSCG_LpFllLock1Lsb = 0U,
kSCG_LpFllLock2Lsb = 1U }
SCG LPFLL lock mode.

Functions

- static void CLOCK_EnableClock (clock_ip_name_t name)
Enable the clock for specific IP.
- static void CLOCK_DisableClock (clock_ip_name_t name)
Disable the clock for specific IP.
- static void CLOCK_SetIpSrc (clock_ip_name_t name, clock_ip_src_t src)
Set the clock source for specific IP module.
- uint32_t CLOCK_GetFreq (clock_name_t clockName)
Gets the clock frequency for a specific clock name.
- uint32_t CLOCK_GetCoreSysClkFreq (void)
Get the core clock or system clock frequency.

- uint32_t CLOCK_GetBusClkFreq (void)
  Get the bus clock frequency.
- uint32_t CLOCK_GetFlashClkFreq (void)
  Get the flash clock frequency.
- uint32_t CLOCK_GetErClkFreq (void)
  Get the external reference clock frequency (ERCLK).
- uint32_t CLOCK_GetIpFreq (clock_ip_name_t name)
  Gets the clock frequency for a specific IP module.

Variables

- volatile uint32_t g_xtal0Freq
  External XTAL0 (OSC0/SYSOSC) clock frequency.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  CLOCK driver version 2.3.0.

MCU System Clock.

- uint32_t CLOCK_GetSysClkFreq (scg_sys_clk_t type)
  Gets the SCG system clock frequency.
- static void CLOCK_SetVlprModeSysClkConfig (const scg_sys_clk_config_t *config)
  Sets the system clock configuration for VLPR mode.
- static void CLOCK_SetRunModeSysClkConfig (const scg_sys_clk_config_t *config)
  Sets the system clock configuration for RUN mode.
- static void CLOCK_GetCurSysClkConfig (scg_sys_clk_config_t *config)
  Gets the system clock configuration in the current power mode.
- static void CLOCK_SetClkOutSel (clock_clkout_src_t setting)
  Sets the clock out selection.

SCG System OSC Clock.

- status_t CLOCK_InitSysOsc (const scg_sosc_config_t *config)
  Initializes the SCG system OSC.
- status_t CLOCK_DeinitSysOsc (void)
  De-initializes the SCG system OSC.
- static void CLOCK_SetSysOscAsyncClkDiv (scg_async_clk_t asyncClk, scg_async_clk_div_t divider)
  Set the asynchronous clock divider.
- uint32_t CLOCK_GetSysOscFreq (void)
  Gets the SCG system OSC clock frequency (SYSOSC).
- uint32_t CLOCK_GetSysOscAsyncFreq (scg_async_clk_t type)
  Gets the SCG asynchronous clock frequency from the system OSC.
- static bool CLOCK_IsSysOscErr (void)
  Checks whether the system OSC clock error occurs.
- static void CLOCK_ClearSysOscErr (void)
  Clears the system OSC clock error.
- static void CLOCK_SetSysOscMonitorMode (scg_sosc_monitor_mode_t mode)

Sets the system OSC monitor mode.
- static bool CLOCK_IsSysOscValid (void)
  Checks whether the system OSC clock is valid.

SCG Slow IRC Clock.

- status_t CLOCK_InitSirc (const scg_sirc_config_t *config)
  Initializes the SCG slow IRC clock.
- status_t CLOCK_DeinitSirc (void)
  De-initializes the SCG slow IRC.
- static void CLOCK_SetSircAsyncClkDiv (scg_async_clk_t asyncClk, scg_async_clk_div_t divider)
  Set the asynchronous clock divider.
- uint32_t CLOCK_GetSircFreq (void)
  Gets the SCG SIRC clock frequency.
- uint32_t CLOCK_GetSircAsyncFreq (scg_async_clk_t type)
  Gets the SCG asynchronous clock frequency from the SIRC.
- static bool CLOCK_IsSircValid (void)
  Checks whether the SIRC clock is valid.

SCG Fast IRC Clock.

- status_t CLOCK_InitFirc (const scg_firc_config_t *config)
  Initializes the SCG fast IRC clock.
- status_t CLOCK_DeinitFirc (void)
  De-initializes the SCG fast IRC.
- static void CLOCK_SetFircAsyncClkDiv (scg_async_clk_t asyncClk, scg_async_clk_div_t divider)
  Set the asynchronous clock divider.
- uint32_t CLOCK_GetFircFreq (void)
  Gets the SCG FIRC clock frequency.
- uint32_t CLOCK_GetFircAsyncFreq (scg_async_clk_t type)
  Gets the SCG asynchronous clock frequency from the FIRC.
- static bool CLOCK_IsFircErr (void)
  Checks whether the FIRC clock error occurs.
- static void CLOCK_ClearFircErr (void)
  Clears the FIRC clock error.
- static bool CLOCK_IsFircValid (void)
  Checks whether the FIRC clock is valid.

SCG Low Power FLL Clock.

- status_t CLOCK_InitLpFll (const scg_lpfll_config_t *config)
  Initializes the SCG LPFLL clock.
- status_t CLOCK_DeinitLpFll (void)
  De-initializes the SCG LPFLL.
- static void CLOCK_SetLpFllAsyncClkDiv (scg_async_clk_t asyncClk, scg_async_clk_div_t divider)
  Set the asynchronous clock divider.
- uint32_t CLOCK_GetLpFllFreq (void)
  Gets the SCG LPFLL clock frequency.
- uint32_t CLOCK_GetLpFllAsyncFreq (scg_async_clk_t type)
  Gets the SCG asynchronous clock frequency from the LPFLL.

- static bool CLOCK_IsLpFllValid (void)
  Checks whether the LPFLL clock is valid.

External clock frequency

- static void CLOCK_SetXtal0Freq (uint32_t freq)
  Sets the XTAL0 frequency based on board settings.

28.2 Data Structure Documentation

28.2.1 struct scg_sys_clk_config_t

Data Fields

- uint32_t divSlow: 4
  Slow clock divider, see scg_sys_clk_div_t.
- uint32_t __pad0__: 4
  Reserved.
- uint32_t __pad1__: 4
  Reserved.
- uint32_t __pad2__: 4
  Reserved.
- uint32_t divCore: 4
  Core clock divider, see scg_sys_clk_div_t.
- uint32_t __pad3__: 4
  Reserved.
- uint32_t src: 4
  System clock source, see scg_sys_clk_src_t.
- uint32_t __pad4__: 4
  reserved.

Field Documentation

(1) uint32_t scg_sys_clk_config_t::divSlow

(2) uint32_t scg_sys_clk_config_t::__pad0__

(3) uint32_t scg_sys_clk_config_t::__pad1__

(4) uint32_t scg_sys_clk_config_t::__pad2__

(5) uint32_t scg_sys_clk_config_t::divCore

(6) uint32_t scg_sys_clk_config_t::__pad3__

(7) uint32_t scg_sys_clk_config_t::src

(8) uint32_t scg_sys_clk_config_t::__pad4__

28.2.2 struct scg_sosc_config_t

Data Fields

- uint32_t freq
  System OSC frequency.
- scg_sosc_monitor_mode_t monitorMode
  Clock monitor mode selected.
- uint8_t enableMode
  Enable mode, OR'ed value of _scg_sosc_enable_mode.
- scg_async_clk_div_t div2
  SOSCDIV2 value.
- scg_sosc_mode_t workMode
  OSC work mode.

Field Documentation

(1) uint32_t scg_sosc_config_t::freq

(2) scg_sosc_monitor_mode_t scg_sosc_config_t::monitorMode

(3) uint8_t scg_sosc_config_t::enableMode

(4) scg_async_clk_div_t scg_sosc_config_t::div2

(5) scg_sosc_mode_t scg_sosc_config_t::workMode

28.2.3 struct scg_sirc_config_t

Data Fields

- uint32_t enableMode
  Enable mode, OR'ed value of _scg_sirc_enable_mode.
- scg_async_clk_div_t div2
  SIRCDIV2 value.
- scg_sirc_range_t range
  Slow IRC frequency range.

Field Documentation

(1) uint32_t scg_sirc_config_t::enableMode

(2) scg_async_clk_div_t scg_sirc_config_t::div2

(3) scg_sirc_range_t scg_sirc_config_t::range

28.2.4 struct scg_firc_trim_config_t

Data Fields

- scg_firc_trim_mode_t trimMode
  FIRC trim mode.
- scg_firc_trim_src_t trimSrc
  Trim source.
- scg_firc_trim_div_t trimDiv
  Trim predivided value for the system OSC.
- uint8_t trimCoar
  Trim coarse value; Irrelevant if trimMode is kSCG_FircTrimUpdate.
- uint8_t trimFine
  Trim fine value; Irrelevant if trimMode is kSCG_FircTrimUpdate.

Field Documentation

(1) scg_firc_trim_mode_t scg_firc_trim_config_t::trimMode

(2) scg_firc_trim_src_t scg_firc_trim_config_t::trimSrc

(3) scg_firc_trim_div_t scg_firc_trim_config_t::trimDiv

(4) uint8_t scg_firc_trim_config_t::trimCoar

(5) uint8_t scg_firc_trim_config_t::trimFine

28.2.5 struct scg_firc_config_t

Data Fields

- uint32_t enableMode
  Enable mode, OR'ed value of _scg_firc_enable_mode.
- scg_async_clk_div_t div2
  FIRCDIV2 value.
- scg_firc_range_t range
  Fast IRC frequency range.
- const scg_firc_trim_config_t * trimConfig
  Pointer to the FIRC trim configuration; set NULL to disable trim.

Field Documentation

(1) uint32_t scg_firc_config_t::enableMode

(2) scg_async_clk_div_t scg_firc_config_t::div2

(3) scg_firc_range_t scg_firc_config_t::range

(4) const scg_firc_trim_config_t* scg_firc_config_t::trimConfig

28.2.6 struct scg_lpfll_trim_config_t

Data Fields

- scg_lpfll_trim_mode_t trimMode
  Trim mode.
- scg_lpfll_lock_mode_t lockMode
  Lock mode; Irrelevant if the trimMode is kSCG_LpFllTrimNonUpdate.
- scg_lpfll_trim_src_t trimSrc
  Trim source.
- uint8_t trimDiv
  Trim predivideds value, which can be 0 $\sim$ 31.
- uint8_t trimValue
  Trim value; Irrelevant if trimMode is the kSCG_LpFllTrimUpdate.

Field Documentation

(1) scg_lpfll_trim_mode_t scg_lpfll_trim_config_t::trimMode

(2) scg_lpfll_lock_mode_t scg_lpfll_trim_config_t::lockMode

(3) scg_lpfll_trim_src_t scg_lpfll_trim_config_t::trimSrc

(4) uint8_t scg_lpfll_trim_config_t::trimDiv

[Trim source frequency / (trimDiv + 1)] must be 2 MHz or 32768 Hz.

(5) uint8_t scg_lpfll_trim_config_t::trimValue

28.2.7 struct scg_lpfll_config_t

Data Fields

- uint8_t enableMode
  Enable mode, OR'ed value of _scg_lpfll_enable_mode.
- scg_async_clk_div_t div2
  LPFLLDIV2 value.
- scg_lpfll_range_t range
  LPFLL frequency range.
- const scg_lpfll_trim_config_t * trimConfig
  Trim configuration; set NULL to disable trim.

Field Documentation

(1) scg_async_clk_div_t scg_lpfll_config_t::div2

(2) scg_lpfll_range_t scg_lpfll_config_t::range

(3) const scg_lpfll_trim_config_t* scg_lpfll_config_t::trimConfig

28.3 Macro Definition Documentation

28.3.1 #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0

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

Note

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

28.3.2 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

28.3.3 #define RTC_CLOCKS

Value:

```
{               \
        kCLOCK_Rtc0 \
    }
```

28.3.4 #define PORT_CLOCKS

Value:

```
{                                                                         \
        kCLOCK_PortA, kCLOCK_PortB, kCLOCK_PortC, kCLOCK_PortD, kCLOCK_PortE \
    }
```

28.3.5 #define LPI2C_CLOCKS

Value:

```
{                 \
        kCLOCK_Lpi2c0 \
    }
```

28.3.6 #define TSI_CLOCKS

Value:

```
{                \
        kCLOCK_Tsi0 \
    }
```

28.3.7 #define LPUART_CLOCKS

Value:

```
{                                                   \
        kCLOCK_Lpuart0, kCLOCK_Lpuart1, kCLOCK_Lpuart2 \
    }
```

28.3.8 #define LPTMR_CLOCKS

Value:

```
{                  \
        kCLOCK_Lptmr0 \
    }
```

28.3.9 #define ADC12_CLOCKS

Value:

```
{                 \
        kCLOCK_Adc0   \
    }
```

28.3.10 #define LPSPI_CLOCKS

Value:

```
{                  \
        kCLOCK_Lpspi0 \
    }
```

28.3.11 #define LPIT_CLOCKS

Value:

```
{                  \
        kCLOCK_Lpit0 \
    }
```

28.3.12 #define CRC_CLOCKS

Value:

```
{                 \
        kCLOCK_Crc0 \
    }
```

28.3.13 #define CMP_CLOCKS

Value:

```
{                 \
        kCLOCK_Cmp0 \
    }
```

28.3.14 #define FLASH_CLOCKS

Value:

```
{                   \
        kCLOCK_Flash0 \
    }
```

28.3.15 #define EWM_CLOCKS

Value:

```
{                 \
        kCLOCK_Ewm0 \
    }
```

28.3.16 #define FTM_CLOCKS

Value:

```
{                            \
        kCLOCK_Ftm0, kCLOCK_Ftm1 \
    }
```

28.3.17 #define PDB_CLOCKS

Value:

```
{               \
        kCLOCK_Pdb0 \
    }
```

28.3.18 #define PWT_CLOCKS

Value:

```
{               \
        kCLOCK_Pwt0 \
    }
```

28.3.19 #define MSCAN_CLOCKS

Value:

```
{                 \
        kCLOCK_Mscan0 \
    }
```

28.3.20 #define CLOCK_GetOsc0ErClkFreq CLOCK_GetErClkFreq

28.4 Enumeration Type Documentation

28.4.1 enum clock_name_t

Enumerator

kCLOCK_CoreSysClk Core/system clock.

kCLOCK_BusClk Bus clock.
kCLOCK_FlashClk Flash clock.
kCLOCK_ScgSysOscClk SCG system OSC clock. (SYSOSC)
kCLOCK_ScgSircClk SCG SIRC clock.
kCLOCK_ScgFircClk SCG FIRC clock.
kCLOCK_ScgLpFllClk SCG low power FLL clock. (LPFLL)
kCLOCK_ScgSysOscAsyncDiv2Clk SOSCDIV2_CLK.
kCLOCK_ScgSircAsyncDiv2Clk SIRCDIV2_CLK.
kCLOCK_ScgFircAsyncDiv2Clk FIRCDIV2_CLK.
kCLOCK_ScgLpFllAsyncDiv2Clk LPFLLDIV2_CLK.
kCLOCK_LpoClk LPO clock.
kCLOCK_ErClk ERCLK. The external reference clock from SCG.

28.4.2 enum clock_ip_src_t

Enumerator

kCLOCK_IpSrcNoneOrExt Clock is off or external clock is used.
kCLOCK_IpSrcSysOscAsync System Oscillator async clock.
kCLOCK_IpSrcSircAsync Slow IRC async clock.
kCLOCK_IpSrcFircAsync Fast IRC async clock.
kCLOCK_IpSrcLpFllAsync LPFLL async clock.

28.4.3 enum clock_ip_name_t

It is defined as the corresponding register address.

28.4.4 anonymous enum

Enumerator

kStatus_SCG_Busy Clock is busy.
kStatus_SCG_InvalidSrc Invalid source.

28.4.5 enum scg_sys_clk_t

Enumerator

kSCG_SysClkSlow System slow clock.
kSCG_SysClkCore Core clock.

28.4.6 enum scg_sys_clk_src_t

Enumerator

kSCG_SysClkSrcSysOsc System OSC.
kSCG_SysClkSrcSirc Slow IRC.
kSCG_SysClkSrcFirc Fast IRC.
kSCG_SysClkSrcLpFll Low power FLL.

28.4.7 enum scg_sys_clk_div_t

Enumerator

kSCG_SysClkDivBy1 Divided by 1.
kSCG_SysClkDivBy2 Divided by 2.
kSCG_SysClkDivBy3 Divided by 3.
kSCG_SysClkDivBy4 Divided by 4.
kSCG_SysClkDivBy5 Divided by 5.
kSCG_SysClkDivBy6 Divided by 6.
kSCG_SysClkDivBy7 Divided by 7.
kSCG_SysClkDivBy8 Divided by 8.
kSCG_SysClkDivBy9 Divided by 9.
kSCG_SysClkDivBy10 Divided by 10.
kSCG_SysClkDivBy11 Divided by 11.
kSCG_SysClkDivBy12 Divided by 12.
kSCG_SysClkDivBy13 Divided by 13.
kSCG_SysClkDivBy14 Divided by 14.
kSCG_SysClkDivBy15 Divided by 15.
kSCG_SysClkDivBy16 Divided by 16.

28.4.8 enum clock_clkout_src_t

Enumerator

kClockClkoutSelScgSlow SCG slow clock.
kClockClkoutSelSysOsc System OSC.
kClockClkoutSelSirc Slow IRC.
kClockClkoutSelFirc Fast IRC.
kClockClkoutSelLpFll Low power FLL.

28.4.9 enum scg_async_clk_t

Enumerator

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

28.4.10 enum scg_async_clk_div_t

Enumerator

kSCG_AsyncClkDisable Clock output is disabled.
kSCG_AsyncClkDivBy1 Divided by 1.
kSCG_AsyncClkDivBy2 Divided by 2.
kSCG_AsyncClkDivBy4 Divided by 4.
kSCG_AsyncClkDivBy8 Divided by 8.
kSCG_AsyncClkDivBy16 Divided by 16.
kSCG_AsyncClkDivBy32 Divided by 32.
kSCG_AsyncClkDivBy64 Divided by 64.

28.4.11 enum scg_sosc_monitor_mode_t

Enumerator

kSCG_SysOscMonitorDisable Monitor disabled.
kSCG_SysOscMonitorInt Interrupt when the system OSC error is detected.
kSCG_SysOscMonitorReset Reset when the system OSC error is detected.

28.4.12 enum scg_sosc_mode_t

Enumerator

kSCG_SysOscModeExt Use external clock.
kSCG_SysOscModeOscLowPower Oscillator low power.
kSCG_SysOscModeOscHighGain Oscillator high gain.

28.4.13 anonymous enum

Enumerator

kSCG_SysOscEnable Enable OSC clock.
kSCG_SysOscEnableInStop Enable OSC in stop mode.

kSCG_SysOscEnableInLowPower Enable OSC in low power mode.
kSCG_SysOscEnableErClk Enable OSCERCLK.

28.4.14 enum scg_sirc_range_t

Enumerator

kSCG_SircRangeLow Slow IRC low range clock (2 MHz, 4 MHz for i.MX 7 ULP).
kSCG_SircRangeHigh Slow IRC high range clock (8 MHz, 16 MHz for i.MX 7 ULP).

28.4.15 anonymous enum

Enumerator

kSCG_SircEnable Enable SIRC clock.
kSCG_SircEnableInStop Enable SIRC in stop mode.
kSCG_SircEnableInLowPower Enable SIRC in low power mode.

28.4.16 enum scg_firc_trim_mode_t

Enumerator

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

28.4.17 enum scg_firc_trim_div_t

Enumerator

kSCG_FircTrimDivBy1 Divided by 1.
kSCG_FircTrimDivBy128 Divided by 128.
kSCG_FircTrimDivBy256 Divided by 256.
kSCG_FircTrimDivBy512 Divided by 512.
kSCG_FircTrimDivBy1024 Divided by 1024.
kSCG_FircTrimDivBy2048 Divided by 2048.

28.4.18 enum scg_firc_trim_src_t

Enumerator

kSCG_FircTrimSrcSysOsc System OSC.

28.4.19 enum scg_firc_range_t

Enumerator

kSCG_FircRange48M Fast IRC is trimmed to 48 MHz.

28.4.20 anonymous enum

Enumerator

kSCG_FircEnable Enable FIRC clock.
kSCG_FircEnableInStop Enable FIRC in stop mode.
kSCG_FircEnableInLowPower Enable FIRC in low power mode.
kSCG_FircDisableRegulator Disable regulator.

28.4.21 anonymous enum

Enumerator

kSCG_LpFllEnable Enable LPFLL clock.

28.4.22 enum scg_lpfll_range_t

Enumerator

kSCG_LpFllRange48M LPFLL is trimmed to 48MHz.

28.4.23 enum scg_lpfll_trim_mode_t

Enumerator

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

28.4.24 enum scg_lpfll_trim_src_t

Enumerator

kSCG_LpFllTrimSrcSirc SIRC.
kSCG_LpFllTrimSrcFirc FIRC.
kSCG_LpFllTrimSrcSysOsc System OSC.
kSCG_LpFllTrimSrcRtcOsc RTC OSC (32.768 kHz).

28.4.25 enum scg_lpfll_lock_mode_t

Enumerator

kSCG_LpFllLock1Lsb Lock with 1 LSB.
kSCG_LpFllLock2Lsb Lock with 2 LSB.

28.5 Function Documentation

28.5.1 static void CLOCK_EnableClock (clock_ip_name_t *name*) [inline], [static]

Parameters

name	Which clock to enable, see clock_ip_name_t.

28.5.2 static void CLOCK_DisableClock (clock_ip_name_t *name*) [inline], [static]

Parameters

name	Which clock to disable, see clock_ip_name_t.

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

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

Parameters

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

28.5.4 uint32_t CLOCK_GetFreq (clock_name_t *clockName*)

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

Parameters

clockName	Clock names defined in clock_name_t

Returns

Clock frequency value in hertz

28.5.5 uint32_t CLOCK_GetCoreSysClkFreq (void)

Returns

Clock frequency in Hz.

28.5.6 uint32_t CLOCK_GetBusClkFreq (void)

Returns

Clock frequency in Hz.

28.5.7 uint32_t CLOCK_GetFlashClkFreq (void)

Returns

Clock frequency in Hz.

28.5.8 uint32_t CLOCK_GetErClkFreq (void)

Returns

Clock frequency in Hz.

28.5.9 uint32_t CLOCK_GetIpFreq (clock_ip_name_t *name*)

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

Parameters

name	Which peripheral to get, see clock_ip_name_t.

Returns

Clock frequency value in hertz

28.5.10 uint32_t CLOCK_GetSysClkFreq (scg_sys_clk_t *type*)

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

Parameters

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

Returns

Clock frequency.

28.5.11 static void CLOCK_SetVlprModeSysClkConfig (const scg_sys_clk_config_t * *config*) [inline], [static]

This function sets the system clock configuration for VLPR mode.

Parameters

config	Pointer to the configuration.

28.5.12 static void CLOCK_SetRunModeSysClkConfig (const scg_sys_clk_config_t * *config*) [inline], [static]

This function sets the system clock configuration for RUN mode.

Parameters

config	Pointer to the configuration.

28.5.13 static void CLOCK_GetCurSysClkConfig (scg_sys_clk_config_t * *config*) [inline], [static]

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

Parameters

config	Pointer to the configuration.

28.5.14 static void CLOCK_SetClkOutSel (clock_clkout_src_t *setting*) [inline], [static]

This function sets the clock out selection (CLKOUTSEL).

Parameters

setting	The selection to set.

Returns

The current clock out selection.

28.5.15 status_t CLOCK_InitSysOsc (const scg_sosc_config_t * *config*)

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

Parameters

config	Pointer to the configuration structure.

Return values

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

Note

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

28.5.16 status_t CLOCK_DeinitSysOsc (void)

This function disables the SCG system OSC clock.

Return values

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

Note

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

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

Parameters

asyncClk	Which asynchronous clock to configure.

divider	The divider value to set.

Note

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

28.5.18 uint32_t CLOCK_GetSysOscFreq (void)

Returns

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

28.5.19 uint32_t CLOCK_GetSysOscAsyncFreq (scg_async_clk_t *type*)

Parameters

type	The asynchronous clock type.

Returns

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

28.5.20 static bool CLOCK_IsSysOscErr (void) [inline], [static]

Returns

True if the error occurs, false if not.

28.5.21 static void CLOCK_SetSysOscMonitorMode (scg_sosc_monitor_mode_t *mode*) [inline], [static]

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

Parameters

mode	Monitor mode to set.

28.5.22 static bool CLOCK_IsSysOscValid (void) [inline], [static]

Returns

True if clock is valid, false if not.

28.5.23 status_t CLOCK_InitSirc (const scg_sirc_config_t * *config*)

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

Parameters

config	Pointer to the configuration structure.

Return values

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

Note

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

28.5.24 status_t CLOCK_DeinitSirc (void)

This function disables the SCG slow IRC.

Return values

kStatus_Success	SIRC is deinitialized.

kStatus_SCG_Busy	SIRC is used by system clock.
kStatus_ReadOnly	SIRC control register is locked.

Note

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

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

Parameters

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

Note

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

28.5.26 uint32_t CLOCK_GetSircFreq (void)

Returns

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

28.5.27 uint32_t CLOCK_GetSircAsyncFreq (scg_async_clk_t *type*)

Parameters

type	The asynchronous clock type.

Returns

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

28.5.28 static bool CLOCK_IsSircValid (void) [inline], [static]

Returns

True if clock is valid, false if not.

28.5.29 status_t CLOCK_InitFirc (const scg_firc_config_t * *config*)

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

Parameters

config	Pointer to the configuration structure.

Return values

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

Note

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

28.5.30 status_t CLOCK_DeinitFirc (void)

This function disables the SCG fast IRC.

Return values

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

Note

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

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

Parameters

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

Note

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

28.5.32 uint32_t CLOCK_GetFircFreq (void)

Returns

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

28.5.33 uint32_t CLOCK_GetFircAsyncFreq (scg_async_clk_t *type*)

Parameters

type	The asynchronous clock type.

Returns

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

28.5.34 static bool CLOCK_IsFircErr (void) [inline], [static]

Returns

True if the error occurs, false if not.

28.5.35 static bool CLOCK_IsFircValid (void) [inline], [static]

Returns

True if clock is valid, false if not.

28.5.36 status_t CLOCK_InitLpFll (const scg_lpfll_config_t * *config*)

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

Parameters

config	Pointer to the configuration structure.

Return values

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

Note

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

28.5.37 status_t CLOCK_DeinitLpFll (void)

This function disables the SCG LPFLL.

Return values

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

Note

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

28.5.38 static void CLOCK_SetLpFllAsyncClkDiv (scg_async_clk_t *asyncClk*, scg_async_clk_div_t *divider*) [inline], [static]

Parameters

asyncClk	Which asynchronous clock to configure.

divider	The divider value to set.

Note

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

28.5.39 uint32_t CLOCK_GetLpFllFreq (void)

Returns

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

28.5.40 uint32_t CLOCK_GetLpFllAsyncFreq (scg_async_clk_t *type*)

Parameters

type	The asynchronous clock type.

Returns

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

28.5.41 static bool CLOCK_IsLpFllValid (void) [inline], [static]

Returns

True if the clock is valid, false if not.

28.5.42 static void CLOCK_SetXtal0Freq (uint32_t *freq*) [inline], [static]

Parameters

freq	The XTAL0/EXTAL0 input clock frequency in Hz.

28.6 Variable Documentation

28.6.1 volatile uint32_t g_xtal0Freq

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

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

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

28.7 System Clock Generator (SCG)

The MCUXpresso SDK provides a peripheral driver for the System Clock Generator (SCG) module of MCUXpresso SDK devices.

28.7.1 Function description

The SCG module contains the system PLL (SPLL), a slow internal reference clock (SIRC), a fast internal reference clock (FIRC), a low power FLL, and the system oscillator clock (SOSC). They can be configured separately as the source of MCU system clocks. Accordingly, the SCG driver provides these functions:

- MCU system clock configuration.
- SCG SOSC configuration.
- SCG SIRC configuration.
- SCG FIRC configuration.
- SCG SPLL configuration.
- SCG LPFLL configuration.

28.7.1.1 MCU System Clock

MCU system clock configurations include the clock source selection and the clock dividers. The configurations for VLPR, RUN, and HSRUN modes are set separately using the CLOCK_SetVlprModeSysClkConfig(), CLOCK_SetRunModeSysClkConfig(), and the CLOCK_SetHsrunModeSysClkConfig() functions to configure the MCU system clock.

The current MCU system clock configuration can be obtained with the function CLOCK_GetCurSysClkConfig(). The current MCU system clock frequency can be obtained with the CLOCK_GetSysClkFreq() function.

28.7.1.2 SCG System OSC Clock

The functions CLOCK_InitSysOsc()/CLOCK_DeinitSysOsc() are used for the SOSC clock initialization. The function CLOCK_InitSysOsc disables the SOSC internally and re-configures it. As a result, ensure that the SOSC is not used while calling these functions.

The SOSC clock can be used directly as the MCU system clock source. The SOSCDIV1_CLK, SOSCDIV2_CLK, and SOSCDIV3_CLK can be used as the peripheral clock source. The clocks frequencies can be obtained by functions CLOCK_GetSysOscFreq() and CLOCK_GetSysOscAsyncFreq().

To configure the SOSC monitor mode, use the function CLOCK_SetSysOscMonitorMode(). The clock error status can be received and cleared with the CLOCK_IsSysOscErr() and CLOCK_ClearSysOscErr() functions.

28.7.1.3 SCG Slow IRC Clock

The functions CLOCK_InitSirc()/CLOCK_DeinitSirc() are used for the SIRC clock initialization. The function CLOCK_InitSirc disables the SIRC internally and re-configures it. Ensure that the SIRC is not used while calling these functions.

The SIRC clock can be used directly as the MCU system clock source. The SIRCDIV1_CLK, SIRCDIV2_CLK, and SIRCDIV3_CLK can be used as the peripheral clock source. The clocks frequencies can be received with functions CLOCK_GetSircFreq() and CLOCK_GetSircAsyncFreq().

28.7.1.4 SCG Fast IRC Clock

The functions CLOCK_InitFirc()/CLOCK_DeinitFirc() are used for the FIRC clock initialization. The function CLOCK_InitFirc disables the FIRC internally and re-configures it. Ensure that the FIRC is not used while calling these functions.

The FIRC clock can be used directly as the MCU system clock source. The FIRCDIV1_CLK, FIRCDIV2_CLK, and FIRCDIV3_CLK can be used as the peripheral clock source. The clocks frequencies could be obtained by functions CLOCK_GetFircFreq() and CLOCK_GetFircAsyncFreq().

The FIRC can be trimmed by the external clock. See the Section "Typical use case" to enable the FIRC trim.

28.7.1.5 SCG Low Power FLL Clock

The functions CLOCK_InitLpFll()/CLOCK_DeinitLpFll() are used for the LPFLL clock initialization. The function CLOCK_InitLpFll disables the LPFLL internally and re-configures it. Ensure that the LPFLL is not used while calling these functions.

The LPFLL clock can be used directly as the MCU system clock source. The LPFLLDIV1_CLK, LPFLLDIV2_CLK, and LPFLLDIV3_CLK can be used as the peripheral clock source. The clocks frequencies could be obtained by functions CLOCK_GetLpFllFreq() and CLOCK_GetLpFllAsyncFreq().

The LPFLL can be trimmed by the external clock, specific the trimConfig in scg_lpfll_config_t to enable the clock trim.

28.7.1.6 SCG System PLL Clock

The functions CLOCK_InitSysPll()/CLOCK_DeinitSysPll() are used for the SPLL clock initialization. The function CLOCK_InitSysPll disables the SPLL internally and re-configures it. Ensure that the SPLL is not used while calling these functions.

To generate the desired SPLL frequency, PREDIV and MULT value must be set properly while initializing the SPLL. The function CLOCK_GetSysPllMultDiv() calculates the PREDIV and MULT. Passing in the reference clock frequency and the desired output frequency, the function returns the PREDIV and MULT which generate the frequency closest to the desired frequency.

Because the SPLL is based on the FIRC or SOSC, the FIRC or SOSC must be enabled first before the SPLL initialization. Also, when re-configuring the FIRC or SOSC, be careful with the SPLL.

The SPLL clock can be used directly as the MCU system clock source. The SPLLDIV1_CLK, SPLLDIV2_CLK, and SPLLDIV3_CLK can be used as the peripheral clock source. The clocks frequencies can be obtained with functions CLOCK_GetSysPllFreq() and CLOCK_GetSysPllAsyncFreq().

To configure the SPLL monitor mode, use the function CLOCK_SetSysPllMonitorMode(). The clock error status can be received and cleared by the CLOCK_IsSysPllErr() and CLOCK_ClearSysPllErr().

28.7.1.7 SCG clock valid check

The functions such as the CLOCK_IsFircValid() are used to check whether a specific clock is valid or not. See "Typical use case" for details.

The clocks are valid after the initialization functions such as the CLOCK_InitFirc(). As a result, it is not necessary to call the CLOCK_IsFircValid() after the CLOCK_InitFirc().

28.7.2 Typical use case

28.7.2.1 FIRC clock trim

During the FIRC initialization, applications can choose whether to enable trim or not.

1. Trim is not enabled. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/scg
2. Trim is enabled. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/scg

28.7.2.2 SPLL initialization

The following code shows how to set up the SCG SPLL. The SPLL uses the SOSC as a reference clock. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/scg

28.7.2.3 System clock configuration

While changing the system clock configuration, the actual system clock does not change until the target clock source is valid. Ensure that the clock source is valid before using it. The functions such as CLOCK_IsSircValid() are used for this purpose.

The SCG has a dedicated system clock configuration registers for VLPR, RUN, and HSRUN modes. During the power mode change, the system clock configuration may change too. In this case, check whether the clock source is valid during the power mode change.

In the following example, the SIRC is used as the system clock source in VLPR mode, the FIRC is used as a system clock source in RUN mode, and the SPLL is used as a system clock source in HSRUN mode.

The example work flow:

1. SIRC, FIRC, and SPLL are all enabled in RUN mode.
2. MCU enters VLPR mode. In VLPR mode, FIRC, and SPLL are disabled automatically.
3. MCU enters RUN mode. Wait for the FIRC to become valid.
4. MCU enters HSRUN mode. In step 3, the SPLL is already enabled, but may not be valid. Wait for it to become valid when entering HSRUN mode. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/scg

Chapter 29
Debug Console

29.1 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.

debug_console_overview.jpg

Debug console overview

29.2 Function groups

29.2.1 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,
} 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);
```

29.2.2 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 *
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 == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```
      version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
      toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */
```

29.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART added to configure PRINTF and low level output perihperal.

- The macro SDK_DEBUGCONSOLE is used for forntend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The fucntion can be set by the macro SDK_DEBUGCONSOLE.
- The macro SDK_DEBUGCONSOLE_UART is used for backend. It is use to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCU-Xpresso, 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.The macro does not specifically refer to the perihpheral "UART". It refers to the external perihperal similar to UART, like as USB CDC, UART, SWO, etc. So if the macro SDK_DEBUGCONSOLE_UART is not defined when tool-chain printf is calling, the semihosting will be used.

The following the matrix show the effects of SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_-UART on PRINTF and printf. The green mark is the default setting of the debug console.

SDK_DEBUGCONSOLE	SDK_DEBUGCONSOLE_UART	PRINTF	printf
DEBUGCONSOLE_-REDIRECT_TO_SDK	defined	Low level peripheral∗	Low level peripheral
DEBUGCONSOLE_-REDIRECT_TO_SDK	undefined	Low level peripheral∗	semihost
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	defined	Low level peripheral∗	Low level peripheral
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	undefined	semihost	semihost
DEBUGCONSOLE_-DISABLE	defined	No ouput	Low level peripheral
DEBUGCONSOLE_-DISABLE	undefined	No ouput	semihost

∗ the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

29.3 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.

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 DEBUGCONSOLE_REDIRECT_TO_SDK
  Definition to select sdk or toolchain printf, 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.
- status_t DbgConsole_EnterLowpower (void)
  Prepares to enter low power consumption.
- status_t DbgConsole_ExitLowpower (void)
  Restores from low power consumption.
- int DbgConsole_Printf (const char ∗fmt_s,...)
  Writes formatted output to the standard output stream.
- int DbgConsole_Vprintf (const char ∗fmt_s, va_list formatStringArg)
  Writes formatted output to the standard output stream.
- int DbgConsole_Putchar (int ch)
  Writes a character to stdout.
- int DbgConsole_Scanf (char ∗fmt_s,...)
  Reads formatted data from the standard input stream.
- int DbgConsole_Getchar (void)
  Reads a character from standard input.

- int DbgConsole_BlockingPrintf (const char *fmt_s,...)
  Writes formatted output to the standard output stream with the blocking mode.
- int DbgConsole_BlockingVprintf (const char *fmt_s, va_list formatStringArg)
  Writes formatted output to the standard output stream with the blocking mode.
- status_t DbgConsole_Flush (void)
  Debug console flush.

29.4 Macro Definition Documentation

29.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

29.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

29.4.3 #define DEBUGCONSOLE_DISABLE 2U

29.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

29.4.5 #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.

29.5 Function Documentation

29.5.1 status_t DbgConsole_Init (uint8_t *instance*, uint32_t *baudRate*, serial_port_type_t *device*, uint32_t *clkSrcFreq*)

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

instance	The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UAR-T adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART periheral is LPUART1. If the uart_adapter.c is added to the current project, the UART periheral is UART1.
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
clkSrcFreq	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

kStatus_Success	Execution successfully

29.5.2 status_t DbgConsole_Deinit (void)

Call this function to disable debug log messages to be output via the specified peripheral initialized by the serial manager module.

Returns

Indicates whether de-initialization was successful or not.

29.5.3 status_t DbgConsole_EnterLowpower (void)

This function is used to prepare to enter low power consumption.

Returns

Indicates whether de-initialization was successful or not.

29.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

29.5.5 int DbgConsole_Printf (const char * *fmt_s*, ...)

Call this function to write a formatted output to the standard output stream.

Parameters

fmt_s	Format control string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

29.5.6 int DbgConsole_Vprintf (const char * *fmt_s*, va_list *formatStringArg*)

Call this function to write a formatted output to the standard output stream.

Parameters

fmt_s	Format control string.
formatString-Arg	Format arguments.

Returns

Returns the number of characters printed or a negative value if an error occurs.

29.5.7 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.

29.5.8 int DbgConsole_Scanf (char * *fmt_s*, ...)

Call this function to read formatted data from the standard input stream.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Parameters

fmt_s	Format control string.

Returns

Returns the number of fields successfully converted and assigned.

29.5.9 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Returns

Returns the character read.

29.5.10 int DbgConsole_BlockingPrintf (const char * *fmt_s*, ...)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

fmt_s	Format control string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

29.5.11 int DbgConsole_BlockingVprintf (const char * *fmt_s*, va_list *formatStringArg*)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

fmt_s	Format control string.
formatStringArg	Format arguments.

Returns

Returns the number of characters printed or a negative value if an error occurs.

29.5.12 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.

29.5.13 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

29.5.14 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 "".

Chapter 30
Notification Framework

30.1 Overview

This section describes the programming interface of the Notifier driver.

30.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.
   The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{

    status_t ret = kStatus_Success;

    ...
    ...
    ...

    return ret;
}
// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *userData)
{
```

```
    ...
    ...
    ...
}
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
                kNOTIFIER_CallbackBeforeAfter,
                (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
      APP_PowerModeSwitch, NULL);
    ...
    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
      kNOTIFIER_PolicyAgreement);
}
```

Data Structures

- struct notifier_notification_block_t
  notification block passed to the registered callback function. More...
- struct notifier_callback_config_t
  Callback configuration structure. More...
- struct notifier_handle_t
  Notifier handle structure. More...

Typedefs

- typedef void notifier_user_config_t
  Notifier user configuration type.
- typedef status_t(∗ notifier_user_function_t)(notifier_user_config_t ∗targetConfig, void ∗userData)
  Notifier user function prototype Use this function to execute specific operations in configuration switch.

- typedef status_t(∗ notifier_callback_t)(notifier_notification_block_t ∗notify, void ∗data)
  Callback prototype.

Enumerations

- enum _notifier_status {
  kStatus_NOTIFIER_ErrorNotificationBefore,
  kStatus_NOTIFIER_ErrorNotificationAfter }
  Notifier error codes.
- enum notifier_policy_t {
  kNOTIFIER_PolicyAgreement,
  kNOTIFIER_PolicyForcible }
  Notifier policies.
- enum notifier_notification_type_t {
  kNOTIFIER_NotifyRecover = 0x00U,
  kNOTIFIER_NotifyBefore = 0x01U,
  kNOTIFIER_NotifyAfter = 0x02U }
  Notification type.
- enum notifier_callback_type_t {
  kNOTIFIER_CallbackBefore = 0x01U,
  kNOTIFIER_CallbackAfter = 0x02U,
  kNOTIFIER_CallbackBeforeAfter = 0x03U }
  The callback type, which indicates kinds of notification the callback handles.

Functions

- status_t NOTIFIER_CreateHandle (notifier_handle_t ∗notifierHandle, notifier_user_config_t ∗∗configs, uint8_t configsNumber, notifier_callback_config_t ∗callbacks, uint8_t callbacksNumber, notifier_user_function_t userFunction, void ∗userData)
  Creates a Notifier handle.
- status_t NOTIFIER_SwitchConfig (notifier_handle_t ∗notifierHandle, uint8_t configIndex, notifier_policy_t policy)
  Switches the configuration according to a pre-defined structure.
- uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t ∗notifierHandle)
  This function returns the last failed notification callback.

30.3 Data Structure Documentation

30.3.1 struct notifier_notification_block_t

Data Fields

- notifier_user_config_t ∗ targetConfig
  Pointer to target configuration.
- notifier_policy_t policy
  Configure transition policy.
- notifier_notification_type_t notifyType
  Configure notification type.

Field Documentation

(1) notifier_user_config_t* notifier_notification_block_t::targetConfig

(2) notifier_policy_t notifier_notification_block_t::policy

(3) notifier_notification_type_t notifier_notification_block_t::notifyType

30.3.2 struct notifier_callback_config_t

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback.

Data Fields

- notifier_callback_t callback
  Pointer to the callback function.
- notifier_callback_type_t callbackType
  Callback type.
- void * callbackData
  Pointer to the data passed to the callback.

Field Documentation

(1) notifier_callback_t notifier_callback_config_t::callback

(2) notifier_callback_type_t notifier_callback_config_t::callbackType

(3) void* notifier_callback_config_t::callbackData

30.3.3 struct notifier_handle_t

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. NOTIFIER_CreateHandle() must be called to initialize this handle.

Data Fields

- notifier_user_config_t ** configsTable
  Pointer to configure table.
- uint8_t configsNumber
  Number of configurations.
- notifier_callback_config_t * callbacksTable
  Pointer to callback table.

- uint8_t callbacksNumber
  Maximum number of callback configurations.
- uint8_t errorCallbackIndex
  Index of callback returns error.
- uint8_t currentConfigIndex
  Index of current configuration.
- notifier_user_function_t userFunction
  User function.
- void * userData
  User data passed to user function.

Field Documentation

(1) notifier_user_config_t** notifier_handle_t::configsTable

(2) uint8_t notifier_handle_t::configsNumber

(3) notifier_callback_config_t* notifier_handle_t::callbacksTable

(4) uint8_t notifier_handle_t::callbacksNumber

(5) uint8_t notifier_handle_t::errorCallbackIndex

(6) uint8_t notifier_handle_t::currentConfigIndex

(7) notifier_user_function_t notifier_handle_t::userFunction

(8) void* notifier_handle_t::userData

30.4 Typedef Documentation

30.4.1 typedef void notifier_user_config_t

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

30.4.2 typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, NOTIFIER_SwitchConfig() exits.

Parameters

targetConfig	target Configuration.
userData	Refers to other specific data passed to user function.

Returns

An error code or kStatus_Success.

30.4.3 typedef status_t(∗ notifier_callback_t)(notifier_notification_block_t ∗notify, void ∗data)

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the notifier_callback_config_t callback configuration structure. Depending on callback type, function of this prototype is called (see NOTIFIER_SwitchConfig()) before configuration switch, after it or in both use cases to notify about the switch progress (see notifier_callback_type_t). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see notifier_notification_block_t) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see notifier_policy_t), the callback may deny the execution of the user function by returning an error code different than kStatus_Success (see NOTIFIER_SwitchConfig()).

Parameters

notify	Notification block.
data	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or kStatus_Success.

30.5 Enumeration Type Documentation

30.5.1 enum _notifier_status

Used as return value of Notifier functions.

Enumerator

kStatus_NOTIFIER_ErrorNotificationBefore An error occurs during send "BEFORE" notification.

kStatus_NOTIFIER_ErrorNotificationAfter An error occurs during send "AFTER" notification.

30.5.2 enum notifier_policy_t

Defines whether the user function execution is forced or not. For kNOTIFIER_PolicyForcible, the user function is executed regardless of the callback results, while kNOTIFIER_PolicyAgreement policy is used to exit NOTIFIER_SwitchConfig() when any of the callbacks returns error code. See also NOTIFIER_-SwitchConfig() description.

Enumerator

kNOTIFIER_PolicyAgreement NOTIFIER_SwitchConfig() method is exited when any of the callbacks returns error code.
kNOTIFIER_PolicyForcible The user function is executed regardless of the results.

30.5.3 enum notifier_notification_type_t

Used to notify registered callbacks

Enumerator

kNOTIFIER_NotifyRecover Notify IP to recover to previous work state.
kNOTIFIER_NotifyBefore Notify IP that configuration setting is going to change.
kNOTIFIER_NotifyAfter Notify IP that configuration setting has been changed.

30.5.4 enum notifier_callback_type_t

Used in the callback configuration structure (notifier_callback_config_t) to specify when the registered callback is called during configuration switch initiated by the NOTIFIER_SwitchConfig(). Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect NOTIFIER_SwitchConfig() execution. See the NOTIFIER_SwitchConfig() and notifier_policy_t documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

kNOTIFIER_CallbackBefore Callback handles BEFORE notification.
kNOTIFIER_CallbackAfter Callback handles AFTER notification.
kNOTIFIER_CallbackBeforeAfter Callback handles BEFORE and AFTER notification.

30.6 Function Documentation

30.6.1 status_t NOTIFIER_CreateHandle (notifier_handle_t * *notifierHandle,* notifier_user_config_t ** *configs,* uint8_t *configsNumber,* notifier_callback-_config_t * *callbacks,* uint8_t *callbacksNumber,* notifier_user_function_t *userFunction,* void * *userData*)

Parameters

notifierHandle	A pointer to the notifier handle.
configs	A pointer to an array with references to all configurations which is handled by the Notifier.
configsNumber	Number of configurations. Size of the configuration array.
callbacks	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
callbacks-Number	Number of registered callbacks. Size of the callbacks array.
userFunction	User function.
userData	User data passed to user function.

Returns

An error Code or kStatus_Success.

30.6.2 status_t NOTIFIER_SwitchConfig (notifier_handle_t ∗ *notifierHandle*, uint8_t *configIndex*, notifier_policy_t *policy*)

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_Get-ErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER_SwitchConfig() exits.

Parameters

notifierHandle	pointer to notifier handle
configIndex	Index of the target configuration.
policy	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus_Success.

30.6.3 uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t * *notifierHandle*)

This function returns an index of the last callback that failed during the configuration switch while the last NOTIFIER_SwitchConfig() was called. If the last NOTIFIER_SwitchConfig() call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

notifierHandle	Pointer to the notifier handle

Returns

Callback Index of the last failed callback or value equal to callbacks count.

Chapter 31
Shell

31.1 Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

31.2 Function groups

31.2.1 Initialization

To initialize the Shell middleware, call the SHELL_Init() function with these parameters. This function automatically enables the middleware.

```
shell_status_t SHELL_Init(shell_handle_t shellHandle,
      serial_handle_t serialHandle, char *prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

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

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
```

31.2.2 Advanced Feature

- Support to get a character from standard input devices.

  ```
  static shell_status_t SHELL_GetChar(shell_context_handle_t *shellContextHandle, uint8_t *ch);
  ```

Commands	Description
help	List all the registered commands.
exit	Exit program.

31.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
SHELL_Task((s_shellHandle);
```

Data Structures

- struct shell_command_t
  User command data configuration structure. More...

Macros

- #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE
  Whether use non-blocking mode.
- #define SHELL_AUTO_COMPLETE (1U)
  Macro to set on/off auto-complete feature.
- #define SHELL_BUFFER_SIZE (64U)
  Macro to set console buffer size.
- #define SHELL_MAX_ARGS (8U)
  Macro to set maximum arguments in command.
- #define SHELL_HISTORY_COUNT (3U)
  Macro to set maximum count of history commands.
- #define SHELL_IGNORE_PARAMETER_COUNT (0xFF)
  Macro to bypass arguments check.
- #define SHELL_HANDLE_SIZE
  The handle size of the shell module.
- #define SHELL_USE_COMMON_TASK (0U)
  Macro to determine whether use common task.
- #define SHELL_TASK_PRIORITY (2U)
  Macro to set shell task priority.
- #define SHELL_TASK_STACK_SIZE (1000U)
  Macro to set shell task stack size.
- #define SHELL_HANDLE_DEFINE(name) uint32_t name[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the shell handle.
- #define SHELL_COMMAND_DEFINE(command, descriptor, callback, paramCount)
  Defines the shell command structure.
- #define SHELL_COMMAND(command) &g_shellCommand##command
  Gets the shell command pointer.

Typedefs

- typedef void ∗ shell_handle_t
  The handle of the shell module.
- typedef shell_status_t(∗ cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char ∗∗argv)
  User command function prototype.

Enumerations

- enum shell_status_t {
  kStatus_SHELL_Success = kStatus_Success,
  kStatus_SHELL_Error = MAKE_STATUS(kStatusGroup_SHELL, 1),
  kStatus_SHELL_OpenWriteHandleFailed = MAKE_STATUS(kStatusGroup_SHELL, 2),
  kStatus_SHELL_OpenReadHandleFailed = MAKE_STATUS(kStatusGroup_SHELL, 3) }
  Shell status.

Shell functional operation

- shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char ∗prompt)
  Initializes the shell module.
- shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t ∗shellCommand)
  Registers the shell command.
- shell_status_t SHELL_UnregisterCommand (shell_command_t ∗shellCommand)
  Unregisters the shell command.
- shell_status_t SHELL_Write (shell_handle_t shellHandle, const char ∗buffer, uint32_t length)
  Sends data to the shell output stream.
- int SHELL_Printf (shell_handle_t shellHandle, const char ∗formatString,...)
  Writes formatted output to the shell output stream.
- shell_status_t SHELL_WriteSynchronization (shell_handle_t shellHandle, const char ∗buffer, uint32_t length)
  Sends data to the shell output stream with OS synchronization.
- int SHELL_PrintfSynchronization (shell_handle_t shellHandle, const char ∗formatString,...)
  Writes formatted output to the shell output stream with OS synchronization.
- void SHELL_ChangePrompt (shell_handle_t shellHandle, char ∗prompt)
  Change shell prompt.
- void SHELL_PrintPrompt (shell_handle_t shellHandle)
  Print shell prompt.
- void SHELL_Task (shell_handle_t shellHandle)
  The task function for Shell.
- static bool SHELL_checkRunningInIsr (void)
  Check if code is running in ISR.

31.3 Data Structure Documentation

31.3.1 struct shell_command_t

Data Fields

- const char ∗ pcCommand
  The command that is executed.
- char ∗ pcHelpString
  String that describes how to use the command.
- const cmd_function_t pFuncCallBack
  A pointer to the callback function that returns the output generated by the command.
- uint8_t cExpectedNumberOfParameters
  Commands expect a fixed number of parameters, which may be zero.
- list_element_t link
  link of the element

Field Documentation

(1) const char∗ shell_command_t::pcCommand

For example "help". It must be all lower case.

(2) char∗ shell_command_t::pcHelpString

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

(3) const cmd_function_t shell_command_t::pFuncCallBack

(4) uint8_t shell_command_t::cExpectedNumberOfParameters

31.4 Macro Definition Documentation

31.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

31.4.2 #define SHELL_AUTO_COMPLETE (1U)

31.4.3 #define SHELL_BUFFER_SIZE (64U)

31.4.4 #define SHELL_MAX_ARGS (8U)

31.4.5 #define SHELL_HISTORY_COUNT (3U)

31.4.6 #define SHELL_HANDLE_SIZE

Value:

```
(160U + SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE +
      SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + \
     SERIAL_MANAGER_WRITE_HANDLE_SIZE)
```

It is the sum of the SHELL_HISTORY_COUNT ∗ SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE

31.4.7 #define SHELL_USE_COMMON_TASK (0U)

31.4.8 #define SHELL_TASK_PRIORITY (2U)

31.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

31.4.10 #define SHELL_HANDLE_DEFINE(*name*) uint32_t name[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned shell handle. Then use "(shell_handle_t)name" to get the shell handle.

The macro should be global and could be optional. You could also define shell handle by yourself.

This is an example,

```
* SHELL_HANDLE_DEFINE(shellHandle);
*
```

Parameters

name	The name string of the shell handle.

31.4.11 #define SHELL_COMMAND_DEFINE(*command, descriptor, callback, paramCount*)

Value:

```
\
    shell_command_t g_shellCommand##command = {                          \
        (#command), (descriptor), (callback), (paramCount), {0},         \
    }
```

This macro is used to define the shell command structure shell_command_t. And then uses the macro SHELL_COMMAND to get the command structure pointer. The macro should not be used in any function.

This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

command	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".

descriptor	The description of the command is used for showing the command usage when "help" is typing.
callback	The callback of the command is used to handle the command line when the input command is matched.
paramCount	The max parameter count of the current command.

31.4.12 #define SHELL_COMMAND(*command*) &g_shellCommand##command

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Parameters

command	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".

31.5 Typedef Documentation

31.5.1 typedef shell_status_t(∗ cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char ∗∗argv)

31.6 Enumeration Type Documentation

31.6.1 enum shell_status_t

Enumerator

kStatus_SHELL_Success Success.
kStatus_SHELL_Error Failed.
kStatus_SHELL_OpenWriteHandleFailed Open write handle failed.
kStatus_SHELL_OpenReadHandleFailed Open read handle failed.

31.7 Function Documentation

31.7.1 shell_status_t SHELL_Init (shell_handle_t *shellHandle,* serial_handle_t *serialHandle,* char ∗ *prompt*)

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL-_Init function by passing in these parameters. This is an example.

```
*    static SHELL_HANDLE_DEFINE(s_shellHandle);
*    SHELL_Init((shell_handle_t)s_shellHandle, (
       serial_handle_t)s_serialHandle, "Test@SHELL>");
*
```

Parameters

shellHandle	Pointer to point to a memory space of size SHELL_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SHELL_HANDLE_DEFINE(shellHandle); or uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];
serialHandle	The serial manager module handle pointer.
prompt	The string prompt pointer of Shell. Only the global variable can be passed.

Return values

kStatus_SHELL_Success	The shell initialization succeed.
kStatus_SHELL_Error	An error occurred when the shell is initialized.
kStatus_SHELL_OpenWriteHandleFailed	Open the write handle failed.
kStatus_SHELL_OpenReadHandleFailed	Open the read handle failed.

31.7.2 shell_status_t SHELL_RegisterCommand (shell_handle_t *shellHandle*, shell_command_t ∗ *shellCommand*)

This function is used to register the shell command by using the command configuration shell_command_config_t. This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

shellHandle	The shell module handle pointer.
shellCommand	The command element.

Return values

kStatus_SHELL_Success	Successfully register the command.
kStatus_SHELL_Error	An error occurred.

31.7.3 shell_status_t SHELL_UnregisterCommand (shell_command_t * *shellCommand*)

This function is used to unregister the shell command.

Parameters

shellCommand	The command element.

Return values

kStatus_SHELL_Success	Successfully unregister the command.

31.7.4 shell_status_t SHELL_Write (shell_handle_t *shellHandle*, const char * *buffer*, uint32_t *length*)

This function is used to send data to the shell output stream.

Parameters

shellHandle	The shell module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SHELL_Success	Successfully send data.
kStatus_SHELL_Error	An error occurred.

31.7.5 int SHELL_Printf (shell_handle_t *shellHandle*, const char * *formatString*, ...)

Call this function to write a formatted output to the shell output stream.

Parameters

shellHandle	The shell module handle pointer.
formatString	Format string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

31.7.6 shell_status_t SHELL_WriteSynchronization (shell_handle_t *shellHandle,* const char * *buffer,* uint32_t *length*)

This function is used to send data to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

shellHandle	The shell module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SHELL_Success	Successfully send data.
kStatus_SHELL_Error	An error occurred.

31.7.7 int SHELL_PrintfSynchronization (shell_handle_t *shellHandle,* const char * *formatString,* ...)

Call this function to write a formatted output to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

shellHandle	The shell module handle pointer.

formatString	Format string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

31.7.8 void SHELL_ChangePrompt (shell_handle_t *shellHandle,* char * *prompt*)

Call this function to change shell prompt.

Parameters

shellHandle	The shell module handle pointer.
prompt	The string which will be used for command prompt

Returns

NULL.

31.7.9 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

shellHandle	The shell module handle pointer.

Returns

NULL.

31.7.10 void SHELL_Task (shell_handle_t *shellHandle*)

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

shellHandle	The shell module handle pointer.

31.7.11 static bool SHELL_checkRunningInIsr (void) [inline], [static]

This function is used to check if code running in ISR.

Return values

TRUE	if code runing in ISR.

Chapter 32
Serial Manager

32.1 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_rpmsg
- Serial_port_swo
- Serial_port_uart
- Serial_port_usb
- Serial_port_virtual

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_MANAGER_NON_BLOCKING_MODE (0U)
  Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)
- #define SERIAL_MANAGER_RING_BUFFER_FLOWCONTROL (0U)
  Enable or ring buffer flow control (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_UART (0U)
  Enable or disable uart port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_UART_DMA (0U)
  Enable or disable uart dma 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_VIRTUAL (0U)
  Enable or disable USB CDC virtual port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_RPMSG (0U)
  Enable or disable rPMSG port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SPI_MASTER (0U)
  Enable or disable SPI Master port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SPI_SLAVE (0U)
  Enable or disable SPI Slave port (1 - enable, 0 - disable)

- #define SERIAL_MANAGER_TASK_HANDLE_TX (0U)
  Enable or disable SerialManager_Task() handle TX to prevent recursive calling.
- #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)
  Set the default delay time in ms used by SerialManager_WriteTimeDelay().
- #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)
  Set the default delay time in ms used by SerialManager_ReadTimeDelay().
- #define SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY (0U)
  Enable or disable SerialManager_Task() handle RX data available notify.
- #define SERIAL_MANAGER_WRITE_HANDLE_SIZE (4U)
  Set serial manager write handle size.
- #define SERIAL_MANAGER_USE_COMMON_TASK (0U)
  SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.
- #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)
  Definition of serial manager handle size.
- #define SERIAL_MANAGER_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager handle.
- #define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager write handle.
- #define SERIAL_MANAGER_READ_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager read handle.
- #define SERIAL_MANAGER_TASK_PRIORITY (2U)
  Macro to set serial manager task priority.
- #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)
  Macro to set serial manager task stack size.

Typedefs

- typedef void ∗ serial_handle_t
  The handle of the serial manager module.
- typedef void ∗ serial_write_handle_t
  The write handle of the serial manager module.
- typedef void ∗ serial_read_handle_t
  The read handle of the serial manager module.
- typedef void(∗ serial_manager_callback_t)(void ∗callbackParam, serial_manager_callback_message_t ∗message, serial_manager_status_t status)
  serial manager callback function
- typedef void(∗ serial_manager_lowpower_critical_callback_t)(void)
  serial manager Lowpower Critical callback function

Enumerations

- enum serial_port_type_t {
  kSerialPort_None = 0U,
  kSerialPort_Uart = 1U,
  kSerialPort_UsbCdc,
  kSerialPort_Swo,
  kSerialPort_Virtual,
  kSerialPort_Rpmsg,
  kSerialPort_UartDma,
  kSerialPort_SpiMaster,
  kSerialPort_SpiSlave }

  serial port type
- enum serial_manager_type_t {
  kSerialManager_NonBlocking = 0x0U,
  kSerialManager_Blocking = 0x8F41U }

  serial manager 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,
  kStatus_SerialManager_NotConnected = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7) }

  serial manager error code

Functions

- serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const 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.
- void SerialManager_SetLowpowerCriticalCb (const serial_manager_lowpower_critical_CBs_t *pf-Callback)
  This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.

32.2 Data Structure Documentation

32.2.1 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.
- serial_manager_type_t blockType
  Serial manager port type.
- void * portConfig
  Serial port configuration.

Field Documentation

(1) uint8_t* serial_manager_config_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

32.2.2 struct serial_manager_callback_message_t

Data Fields

- uint8_t * buffer
  Transferred buffer.
- uint32_t length
  Transferred data length.

32.3 Macro Definition Documentation

32.3.1 #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)

32.3.2 #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)

32.3.3 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

Macro to determine whether use common task.

32.3.4 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_-SIZE_TEMP + 12U)

32.3.5 #define SERIAL_MANAGER_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial_handle_t)name" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);
*
```

Parameters

name	The name string of the serial manager handle.

32.3.6 #define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial_write_handle-_t)name" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);
*
```

Parameters

name	The name string of the serial manager write handle.

32.3.7 #define SERIAL_MANAGER_READ_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial_read_handle-_t)name" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```

Parameters

name	The name string of the serial manager read handle.

32.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

32.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

32.4 Enumeration Type Documentation

32.4.1 enum serial_port_type_t

Enumerator

kSerialPort_None Serial port is none.
kSerialPort_Uart Serial port UART.
kSerialPort_UsbCdc Serial port USB CDC.
kSerialPort_Swo Serial port SWO.
kSerialPort_Virtual Serial port Virtual.
kSerialPort_Rpmsg Serial port RPMSG.
kSerialPort_UartDma Serial port UART DMA.

kSerialPort_SpiMaster Serial port SPIMASTER.
kSerialPort_SpiSlave Serial port SPISLAVE.

32.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.
kSerialManager_Blocking Blocking handle.

32.4.3 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.
kStatus_SerialManager_NotConnected The host is not connected.

32.5 Function Documentation

32.5.1 serial_manager_status_t SerialManager_Init (serial_handle_t *serialHandle*, const 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 three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to serial_port_type_t for serial port setting. These three 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 SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   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;
*   uartConfig.enableRxRTS = 0;
*   uartConfig.enableTxCTS = 0;
*   config.portConfig = &uartConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

For USB CDC,

```
*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   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((serial_handle_t)s_serialHandle, &config);
*
```

Parameters

serialHandle	Pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_HANDLE_DEFINE(serialHandle); or uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];
config	Pointer to user-defined configuration structure.

Return values

kStatus_SerialManager_Error	An error occurred.
kStatus_SerialManager_Success	The Serial Manager module initialization succeed.

32.5.2 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.

32.5.3 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. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
writeHandle	The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_WRITE_HAN-DLE_DEFINE(writeHandle); or uint32_t writeHandle[((SERIAL_MANAGER_W-RITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

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 SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
*   static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
*   SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
     , (serial_write_handle_t)s_serialWriteHandle1);
*   SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle1,
*                                    Task1_SerialManagerTxCallback,
*                                    s_serialWriteHandle1);
*   SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle1,
*                                   s_nonBlockingWelcome1,
*                                   sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
*   static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
*   static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
*   SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
     , (serial_write_handle_t)s_serialWriteHandle2);
*   SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle2,
*                                    Task2_SerialManagerTxCallback,
*                                    s_serialWriteHandle2);
*   SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle2,
*                                   s_nonBlockingWelcome2,
*                                   sizeof(s_nonBlockingWelcome2) - 1U);
*
```

32.5.4 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.

32.5.5 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. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
readHandle	The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle); or uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

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 SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
*   SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
      (serial_read_handle_t)s_serialReadHandle);
*   static uint8_t s_nonBlockingBuffer[64];
*   SerialManager_InstallRxCallback((serial_read_handle_t)s_serialReadHandle,
*                                    APP_SerialManagerRxCallback,
*                                    s_serialReadHandle);
*   SerialManager_ReadNonBlocking((serial_read_handle_t)s_serialReadHandle,
*                                  s_nonBlockingBuffer,
*                                  sizeof(s_nonBlockingBuffer));
*
```

32.5.6 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.

32.5.7 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.

32.5.8 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.

32.5.9 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.

32.5.10 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.

32.5.11 void SerialManager_SetLowpowerCriticalCb (const serial_manager_-lowpower_critical_CBs_t * *pfCallback*)

Parameters

pfCallback	Pointer to the function structure used to allow/disable lowpower.

Chapter 33
LPSPI FreeRTOS Driver

33.1 Overview

Driver version

- #define FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  LPSPI FreeRTOS driver version 2.0.5.

LPSPI RTOS Operation

- status_t LPSPI_RTOS_Init (lpspi_rtos_handle_t ∗handle, LPSPI_Type ∗base, const lpspi_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes LPSPI.
- status_t LPSPI_RTOS_Deinit (lpspi_rtos_handle_t ∗handle)
  Deinitializes the LPSPI.
- status_t LPSPI_RTOS_Transfer (lpspi_rtos_handle_t ∗handle, lpspi_transfer_t ∗transfer)
  Performs SPI transfer.

33.2 Macro Definition Documentation

33.2.1 #define FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

33.3 Function Documentation

33.3.1 status_t LPSPI_RTOS_Init (lpspi_rtos_handle_t ∗ *handle,* LPSPI_Type ∗ *base,* const lpspi_master_config_t ∗ *masterConfig,* uint32_t *srcClock_Hz*)

This function initializes the LPSPI module and related RTOS context.

Parameters

handle	The RTOS LPSPI handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the LPSPI instance to initialize.
masterConfig	Configuration structure to set-up LPSPI in master mode.

srcClock_Hz	Frequency of input clock of the LPSPI module.

Returns

status of the operation.

33.3.2 status_t LPSPI_RTOS_Deinit (lpspi_rtos_handle_t * *handle*)

This function deinitializes the LPSPI module and related RTOS context.

Parameters

handle	The RTOS LPSPI handle.

33.3.3 status_t LPSPI_RTOS_Transfer (lpspi_rtos_handle_t * *handle*, lpspi_transfer_t * *transfer*)

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

handle	The RTOS LPSPI handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

Chapter 34
Lpuart_freertos_driver

34.1 Overview

Data Structures

- struct lpuart_rtos_config_t
  LPUART RTOS configuration structure. More...

Driver version

- #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  LPUART FreeRTOS driver version.

LPUART RTOS Operation

- int LPUART_RTOS_Init (lpuart_rtos_handle_t ∗handle, lpuart_handle_t ∗t_handle, const lpuart_rtos_config_t ∗cfg)
  Initializes an LPUART instance for operation in RTOS.
- int LPUART_RTOS_Deinit (lpuart_rtos_handle_t ∗handle)
  Deinitializes an LPUART instance for operation.

LPUART transactional Operation

- int LPUART_RTOS_Send (lpuart_rtos_handle_t ∗handle, uint8_t ∗buffer, uint32_t length)
  Sends data in the background.
- int LPUART_RTOS_Receive (lpuart_rtos_handle_t ∗handle, uint8_t ∗buffer, uint32_t length, size_t ∗received)
  Receives data.
- int LPUART_RTOS_SetRxTimeout (lpuart_rtos_handle_t ∗handle, uint32_t rx_timeout_constant_ms, uint32_t rx_timeout_multiplier_ms)
  Set RX timeout in runtime.
- int LPUART_RTOS_SetTxTimeout (lpuart_rtos_handle_t ∗handle, uint32_t tx_timeout_constant_ms, uint32_t tx_timeout_multiplier_ms)
  Set TX timeout in runtime.

34.2 Data Structure Documentation

34.2.1 struct lpuart_rtos_config_t

Data Fields

- LPUART_Type ∗ base
  UART base address.

- uint32_t srcclk
  UART source clock in Hz.
- uint32_t baudrate
  Desired communication speed.
- lpuart_parity_mode_t parity
  Parity setting.
- lpuart_stop_bit_count_t stopbits
  Number of stop bits to use.
- uint8_t * buffer
  Buffer for background reception.
- uint32_t buffer_size
  Size of buffer for background reception.
- uint32_t rx_timeout_constant_ms
  RX timeout applied per receive.
- uint32_t rx_timeout_multiplier_ms
  RX timeout added for each byte of the receive.
- uint32_t tx_timeout_constant_ms
  TX timeout applied per transmition.
- uint32_t tx_timeout_multiplier_ms
  TX timeout added for each byte of the transmition.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- lpuart_transmit_cts_source_t txCtsSource
  TX CTS source.
- lpuart_transmit_cts_config_t txCtsConfig
  TX CTS configure.

Field Documentation

(1) uint32_t lpuart_rtos_config_t::rx_timeout_multiplier_ms

(2) uint32_t lpuart_rtos_config_t::tx_timeout_multiplier_ms

34.3 Macro Definition Documentation

34.3.1 #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

34.4 Function Documentation

34.4.1 int LPUART_RTOS_Init (lpuart_rtos_handle_t * *handle*, lpuart_handle_t * *t_handle*, const lpuart_rtos_config_t * *cfg*)

Parameters

handle	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
t_handle	The pointer to an allocated space to store the transactional layer internal state.
cfg	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

34.4.2 int LPUART_RTOS_Deinit (lpuart_rtos_handle_t * *handle*)

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

handle	The RTOS LPUART handle.

34.4.3 int LPUART_RTOS_Send (lpuart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length*)

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

handle	The RTOS LPUART handle.
buffer	The pointer to buffer to send.
length	The number of bytes to send.

34.4.4 int LPUART_RTOS_Receive (lpuart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length,* size_t * *received*)

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

handle	The RTOS LPUART handle.
buffer	The pointer to buffer where to write received data.
length	The number of bytes to receive.
received	The pointer to a variable of size_t where the number of received data is filled.

34.4.5 int LPUART_RTOS_SetRxTimeout (lpuart_rtos_handle_t * *handle*, uint32_t *rx_timeout_constant_ms*, uint32_t *rx_timeout_multiplier_ms*)

This function can modify RX timeout between initialization and receive.

param handle The RTOS LPUART handle. param rx_timeout_constant_ms RX timeout applied per receive. param rx_timeout_multiplier_ms RX timeout added for each byte of the receive.

34.4.6 int LPUART_RTOS_SetTxTimeout (lpuart_rtos_handle_t * *handle*, uint32_t *tx_timeout_constant_ms*, uint32_t *tx_timeout_multiplier_ms*)

This function can modify TX timeout between initialization and send.

param handle The RTOS LPUART handle. param tx_timeout_constant_ms TX timeout applied per transmition. param tx_timeout_multiplier_ms TX timeout added for each byte of the transmition.

Chapter 35
Tsi_v5_driver

35.1 Overview

Data Structures

- struct tsi_calibration_data_t
  TSI calibration data storage. More...
- struct tsi_common_config_t
  TSI common configuration structure. More...
- struct tsi_selfCap_config_t
  TSI configuration structure for self-cap mode. More...
- struct tsi_mutualCap_config_t
  TSI configuration structure for mutual-cap mode. More...

Macros

- #define FSL_TSI_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  TSI driver version.
- #define ALL_FLAGS_MASK (TSI_GENCS_EOSF_MASK | TSI_GENCS_OUTRGF_MASK)
  TSI status flags macro collection.

Enumerations

- enum tsi_main_clock_selection_t {
  kTSI_MainClockSlection_0 = 0U,
  kTSI_MainClockSlection_1 = 1U,
  kTSI_MainClockSlection_2 = 2U,
  kTSI_MainClockSlection_3 = 3U }
  TSI main clock selection.
- enum tsi_sensing_mode_selection_t {
  kTSI_SensingModeSlection_Self = 0U,
  kTSI_SensingModeSlection_Mutual = 1U }
  TSI sensing mode selection.
- enum tsi_dvolt_option_t {
  kTSI_DvoltOption_0 = 0U,
  kTSI_DvoltOption_1 = 1U,
  kTSI_DvoltOption_2 = 2U,
  kTSI_DvoltOption_3 = 3U }
  TSI DVOLT settings.
- enum tsi_sensitivity_xdn_option_t {

kTSI_SensitivityXdnOption_0 = 0U,
kTSI_SensitivityXdnOption_1 = 1U,
kTSI_SensitivityXdnOption_2 = 2U,
kTSI_SensitivityXdnOption_3 = 3U,
kTSI_SensitivityXdnOption_4 = 4U,
kTSI_SensitivityXdnOption_5 = 5U,
kTSI_SensitivityXdnOption_6 = 6U,
kTSI_SensitivityXdnOption_7 = 7U }
TSI sensitivity ajustment (XDN option).
- enum tsi_shield_t {
kTSI_shieldAllOff = 0U,
kTSI_shield0On = 1U,
kTSI_shield1On = 2U,
kTSI_shield1and0On = 3U,
kTSI_shield2On = 4U,
kTSI_shield2and0On = 5U,
kTSI_shield2and1On = 6U,
kTSI_shieldAllOn = 7U }
TSI Shield setting (S_W_SHIELD option).
- enum tsi_sensitivity_ctrim_option_t {
kTSI_SensitivityCtrimOption_0 = 0U,
kTSI_SensitivityCtrimOption_1 = 1U,
kTSI_SensitivityCtrimOption_2 = 2U,
kTSI_SensitivityCtrimOption_3 = 3U,
kTSI_SensitivityCtrimOption_4 = 4U,
kTSI_SensitivityCtrimOption_5 = 5U,
kTSI_SensitivityCtrimOption_6 = 6U,
kTSI_SensitivityCtrimOption_7 = 7U }
TSI sensitivity ajustment (CTRIM option).
- enum tsi_current_multiple_input_t {
kTSI_CurrentMultipleInputValue_0 = 0U,
kTSI_CurrentMultipleInputValue_1 = 1U }
TSI current ajustment (Input current multiple).
- enum tsi_current_multiple_charge_t {
kTSI_CurrentMultipleChargeValue_0 = 0U,
kTSI_CurrentMultipleChargeValue_1 = 1U,
kTSI_CurrentMultipleChargeValue_2 = 2U,
kTSI_CurrentMultipleChargeValue_3 = 3U,
kTSI_CurrentMultipleChargeValue_4 = 4U,
kTSI_CurrentMultipleChargeValue_5 = 5U,
kTSI_CurrentMultipleChargeValue_6 = 6U,
kTSI_CurrentMultipleChargeValue_7 = 7U }
TSI current ajustment (Charge/Discharge current multiple).
- enum tsi_mutual_pre_current_t {

kTSI_MutualPreCurrent_1uA = 0U,
kTSI_MutualPreCurrent_2uA = 1U,
kTSI_MutualPreCurrent_3uA = 2U,
kTSI_MutualPreCurrent_4uA = 3U,
kTSI_MutualPreCurrent_5uA = 4U,
kTSI_MutualPreCurrent_6uA = 5U,
kTSI_MutualPreCurrent_7uA = 6U,
kTSI_MutualPreCurrent_8uA = 7U }

TSI current used in vref generator.

- enum tsi_mutual_pre_resistor_t {
kTSI_MutualPreResistor_1k = 0U,
kTSI_MutualPreResistor_2k = 1U,
kTSI_MutualPreResistor_3k = 2U,
kTSI_MutualPreResistor_4k = 3U,
kTSI_MutualPreResistor_5k = 4U,
kTSI_MutualPreResistor_6k = 5U,
kTSI_MutualPreResistor_7k = 6U,
kTSI_MutualPreResistor_8k = 7U }

TSI resistor used in pre-charge.

- enum tsi_mutual_sense_resistor_t {
kTSI_MutualSenseResistor_2k5 = 0U,
kTSI_MutualSenseResistor_5k = 1U,
kTSI_MutualSenseResistor_7k5 = 2U,
kTSI_MutualSenseResistor_10k = 3U,
kTSI_MutualSenseResistor_12k5 = 4U,
kTSI_MutualSenseResistor_15k = 5U,
kTSI_MutualSenseResistor_17k5 = 6U,
kTSI_MutualSenseResistor_20k = 7U,
kTSI_MutualSenseResistor_22k5 = 8U,
kTSI_MutualSenseResistor_25k = 9U,
kTSI_MutualSenseResistor_27k5 = 10U,
kTSI_MutualSenseResistor_30k = 11U,
kTSI_MutualSenseResistor_32k5 = 12U,
kTSI_MutualSenseResistor_35k = 13U,
kTSI_MutualSenseResistor_37k5 = 14U,
kTSI_MutualSenseResistor_40k = 15U }

TSI resistor used in I-sense generator.

- enum tsi_mutual_tx_channel_t {
kTSI_MutualTxChannel_0 = 0U,
kTSI_MutualTxChannel_1 = 1U,
kTSI_MutualTxChannel_2 = 2U,
kTSI_MutualTxChannel_3 = 3U,
kTSI_MutualTxChannel_4 = 4U,
kTSI_MutualTxChannel_5 = 5U }

TSI TX channel selection in mutual-cap mode.

- enum tsi_mutual_rx_channel_t {

kTSI_MutualRxChannel_6 = 0U,
kTSI_MutualRxChannel_7 = 1U,
kTSI_MutualRxChannel_8 = 2U,
kTSI_MutualRxChannel_9 = 3U,
kTSI_MutualRxChannel_10 = 4U,
kTSI_MutualRxChannel_11 = 5U }
TSI RX channel selection in mutual-cap mode.
- enum tsi_mutual_sense_boost_current_t {
kTSI_MutualSenseBoostCurrent_0uA = 0U,
kTSI_MutualSenseBoostCurrent_2uA = 1U,
kTSI_MutualSenseBoostCurrent_4uA = 2U,
kTSI_MutualSenseBoostCurrent_6uA = 3U,
kTSI_MutualSenseBoostCurrent_8uA = 4U,
kTSI_MutualSenseBoostCurrent_10uA = 5U,
kTSI_MutualSenseBoostCurrent_12uA = 6U,
kTSI_MutualSenseBoostCurrent_14uA = 7U,
kTSI_MutualSenseBoostCurrent_16uA = 8U,
kTSI_MutualSenseBoostCurrent_18uA = 9U,
kTSI_MutualSenseBoostCurrent_20uA = 10U,
kTSI_MutualSenseBoostCurrent_22uA = 11U,
kTSI_MutualSenseBoostCurrent_24uA = 12U,
kTSI_MutualSenseBoostCurrent_26uA = 13U,
kTSI_MutualSenseBoostCurrent_28uA = 14U,
kTSI_MutualSenseBoostCurrent_30uA = 15U,
kTSI_MutualSenseBoostCurrent_32uA = 16U,
kTSI_MutualSenseBoostCurrent_34uA = 17U,
kTSI_MutualSenseBoostCurrent_36uA = 18U,
kTSI_MutualSenseBoostCurrent_38uA = 19U,
kTSI_MutualSenseBoostCurrent_40uA = 20U,
kTSI_MutualSenseBoostCurrent_42uA = 21U,
kTSI_MutualSenseBoostCurrent_44uA = 22U,
kTSI_MutualSenseBoostCurrent_46uA = 23U,
kTSI_MutualSenseBoostCurrent_48uA = 24U,
kTSI_MutualSenseBoostCurrent_50uA = 25U,
kTSI_MutualSenseBoostCurrent_52uA = 26U,
kTSI_MutualSenseBoostCurrent_54uA = 27U,
kTSI_MutualSenseBoostCurrent_56uA = 28U,
kTSI_MutualSenseBoostCurrent_58uA = 29U,
kTSI_MutualSenseBoostCurrent_60uA = 30U,
kTSI_MutualSenseBoostCurrent_62uA = 31U }
TSI sensitivity boost current settings.
- enum tsi_mutual_tx_drive_mode_t {
kTSI_MutualTxDriveModeOption_0 = 0U,
kTSI_MutualTxDriveModeOption_1 = 1U }
TSI TX drive mode control.

- enum tsi_mutual_pmos_current_left_t {
kTSI_MutualPmosCurrentMirrorLeft_4 = 0U,
kTSI_MutualPmosCurrentMirrorLeft_8 = 1U,
kTSI_MutualPmosCurrentMirrorLeft_12 = 2U,
kTSI_MutualPmosCurrentMirrorLeft_16 = 3U,
kTSI_MutualPmosCurrentMirrorLeft_20 = 4U,
kTSI_MutualPmosCurrentMirrorLeft_24 = 5U,
kTSI_MutualPmosCurrentMirrorLeft_28 = 6U,
kTSI_MutualPmosCurrentMirrorLeft_32 = 7U }
TSI Pmos current mirror selection on the left side.
- enum tsi_mutual_pmos_current_right_t {
kTSI_MutualPmosCurrentMirrorRight_1 = 0U,
kTSI_MutualPmosCurrentMirrorRight_2 = 1U,
kTSI_MutualPmosCurrentMirrorRight_3 = 2U,
kTSI_MutualPmosCurrentMirrorRight_4 = 3U }
TSI Pmos current mirror selection on the right side.
- enum tsi_mutual_nmos_current_t {
kTSI_MutualNmosCurrentMirror_1 = 0U,
kTSI_MutualNmosCurrentMirror_2 = 1U,
kTSI_MutualNmosCurrentMirror_3 = 2U,
kTSI_MutualNmosCurrentMirror_4 = 3U }
TSI Nmos current mirror selection.
- enum tsi_sinc_cutoff_div_t {
kTSI_SincCutoffDiv_1 = 0U,
kTSI_SincCutoffDiv_2 = 1U,
kTSI_SincCutoffDiv_4 = 2U,
kTSI_SincCutoffDiv_8 = 3U,
kTSI_SincCutoffDiv_16 = 4U,
kTSI_SincCutoffDiv_32 = 5U,
kTSI_SincCutoffDiv_64 = 6U,
kTSI_SincCutoffDiv_128 = 7U }
TSI SINC cutoff divider setting.
- enum tsi_sinc_filter_order_t {
kTSI_SincFilterOrder_1 = 0U,
kTSI_SincFilterOrder_2 = 1U }
TSI SINC filter order setting.
- enum tsi_sinc_decimation_value_t {

kTSI_SincDecimationValue_1 = 0U,
kTSI_SincDecimationValue_2 = 1U,
kTSI_SincDecimationValue_3 = 2U,
kTSI_SincDecimationValue_4 = 3U,
kTSI_SincDecimationValue_5 = 4U,
kTSI_SincDecimationValue_6 = 5U,
kTSI_SincDecimationValue_7 = 6U,
kTSI_SincDecimationValue_8 = 7U,
kTSI_SincDecimationValue_9 = 8U,
kTSI_SincDecimationValue_10 = 9U,
kTSI_SincDecimationValue_11 = 10U,
kTSI_SincDecimationValue_12 = 11U,
kTSI_SincDecimationValue_13 = 12U,
kTSI_SincDecimationValue_14 = 13U,
kTSI_SincDecimationValue_15 = 14U,
kTSI_SincDecimationValue_16 = 15U,
kTSI_SincDecimationValue_17 = 16U,
kTSI_SincDecimationValue_18 = 17U,
kTSI_SincDecimationValue_19 = 18U,
kTSI_SincDecimationValue_20 = 19U,
kTSI_SincDecimationValue_21 = 20U,
kTSI_SincDecimationValue_22 = 21U,
kTSI_SincDecimationValue_23 = 22U,
kTSI_SincDecimationValue_24 = 23U,
kTSI_SincDecimationValue_25 = 24U,
kTSI_SincDecimationValue_26 = 25U,
kTSI_SincDecimationValue_27 = 26U,
kTSI_SincDecimationValue_28 = 27U,
kTSI_SincDecimationValue_29 = 28U,
kTSI_SincDecimationValue_30 = 29U,
kTSI_SincDecimationValue_31 = 30U,
kTSI_SincDecimationValue_32 = 31U }

TSI SINC decimation value setting.

- enum tsi_ssc_charge_num_t {

kTSI_SscChargeNumValue_1 = 0U,
kTSI_SscChargeNumValue_2 = 1U,
kTSI_SscChargeNumValue_3 = 2U,
kTSI_SscChargeNumValue_4 = 3U,
kTSI_SscChargeNumValue_5 = 4U,
kTSI_SscChargeNumValue_6 = 5U,
kTSI_SscChargeNumValue_7 = 6U,
kTSI_SscChargeNumValue_8 = 7U,
kTSI_SscChargeNumValue_9 = 8U,
kTSI_SscChargeNumValue_10 = 9U,
kTSI_SscChargeNumValue_11 = 10U,
kTSI_SscChargeNumValue_12 = 11U,
kTSI_SscChargeNumValue_13 = 12U,
kTSI_SscChargeNumValue_14 = 13U,
kTSI_SscChargeNumValue_15 = 14U,
kTSI_SscChargeNumValue_16 = 15U }
TSI SSC output bit0's period setting(SSC0[CHARGE_NUM])
- enum tsi_ssc_nocharge_num_t {
kTSI_SscNoChargeNumValue_1 = 0U,
kTSI_SscNoChargeNumValue_2 = 1U,
kTSI_SscNoChargeNumValue_3 = 2U,
kTSI_SscNoChargeNumValue_4 = 3U,
kTSI_SscNoChargeNumValue_5 = 4U,
kTSI_SscNoChargeNumValue_6 = 5U,
kTSI_SscNoChargeNumValue_7 = 6U,
kTSI_SscNoChargeNumValue_8 = 7U,
kTSI_SscNoChargeNumValue_9 = 8U,
kTSI_SscNoChargeNumValue_10,
kTSI_SscNoChargeNumValue_11,
kTSI_SscNoChargeNumValue_12,
kTSI_SscNoChargeNumValue_13,
kTSI_SscNoChargeNumValue_14,
kTSI_SscNoChargeNumValue_15,
kTSI_SscNoChargeNumValue_16 }
TSI SSC output bit1's period setting(SSC0[BASE_NOCHARGE_NUM])
- enum tsi_ssc_prbs_outsel_t {

kTSI_SscPrbsOutsel_2 = 2U,
kTSI_SscPrbsOutsel_3 = 3U,
kTSI_SscPrbsOutsel_4 = 4U,
kTSI_SscPrbsOutsel_5 = 5U,
kTSI_SscPrbsOutsel_6 = 6U,
kTSI_SscPrbsOutsel_7 = 7U,
kTSI_SscPrbsOutsel_8 = 8U,
kTSI_SscPrbsOutsel_9 = 9U,
kTSI_SscPrbsOutsel_10 = 10U,
kTSI_SscPrbsOutsel_11 = 11U,
kTSI_SscPrbsOutsel_12 = 12U,
kTSI_SscPrbsOutsel_13 = 13U,
kTSI_SscPrbsOutsel_14 = 14U,
kTSI_SscPrbsOutsel_15 = 15U }
TSI SSC outsel choosing the length of the PRBS (Pseudo-RandomBinarySequence) method setting(SSC0[-TSI_SSC0_PRBS_OUTSEL])

- enum tsi_status_flags_t {
kTSI_EndOfScanFlag = TSI_GENCS_EOSF_MASK,
kTSI_OutOfRangeFlag = (int)TSI_GENCS_OUTRGF_MASK }
TSI status flags.
- enum tsi_interrupt_enable_t {
kTSI_GlobalInterruptEnable = 1U,
kTSI_OutOfRangeInterruptEnable = 2U,
kTSI_EndOfScanInterruptEnable = 4U }
TSI feature interrupt source.
- enum tsi_ssc_mode_t {
kTSI_ssc_prbs_method = 0U,
kTSI_ssc_up_down_counter = 1U,
kTSI_ssc_dissable = 2U }
TSI SSC mode selection.
- enum tsi_ssc_prescaler_t {
kTSI_ssc_div_by_1 = 0x0U,
kTSI_ssc_div_by_2 = 0x1U,
kTSI_ssc_div_by_4 = 0x3U,
kTSI_ssc_div_by_8 = 0x7U,
kTSI_ssc_div_by_16 = 0xfU,
kTSI_ssc_div_by_32 = 0x1fU,
kTSI_ssc_div_by_64 = 0x3fU,
kTSI_ssc_div_by_128 = 0x7fU,
kTSI_ssc_div_by_256 = 0xffU }
TSI main clock selection.

Functions

- uint32_t TSI_GetInstance (TSI_Type ∗base)
Get the TSI instance from peripheral base address.
- void TSI_InitSelfCapMode (TSI_Type ∗base, const tsi_selfCap_config_t ∗config)

Initialize hardware to Self-cap mode.
- void TSI_InitMutualCapMode (TSI_Type ∗base, const tsi_mutualCap_config_t ∗config)
  Initialize hardware to Mutual-cap mode.
- void TSI_Deinit (TSI_Type ∗base)
  De-initialize hardware.
- void TSI_GetSelfCapModeDefaultConfig (tsi_selfCap_config_t ∗userConfig)
  Get TSI self-cap mode user configure structure.
- void TSI_GetMutualCapModeDefaultConfig (tsi_mutualCap_config_t ∗userConfig)
  Get TSI mutual-cap mode default user configure structure.
- void TSI_SelfCapCalibrate (TSI_Type ∗base, tsi_calibration_data_t ∗calBuff)
  Hardware base counter value for calibration.
- void TSI_EnableInterrupts (TSI_Type ∗base, uint32_t mask)
  Enables TSI interrupt requests.
- void TSI_DisableInterrupts (TSI_Type ∗base, uint32_t mask)
  Disables TSI interrupt requests.
- static uint32_t TSI_GetStatusFlags (TSI_Type ∗base)
  Get interrupt flag.
- void TSI_ClearStatusFlags (TSI_Type ∗base, uint32_t mask)
  Clear interrupt flag.
- static uint32_t TSI_GetScanTriggerMode (TSI_Type ∗base)
  Get TSI scan trigger mode.
- static bool TSI_IsScanInProgress (TSI_Type ∗base)
  Get scan in progress flag.
- static void TSI_EnableModule (TSI_Type ∗base, bool enable)
  Enables the TSI Module or not.
- static void TSI_EnableLowPower (TSI_Type ∗base, bool enable)
  Sets the TSI low power STOP mode enable or not.
- static void TSI_EnableHardwareTriggerScan (TSI_Type ∗base, bool enable)
  Enable the hardware trigger scan or not.
- static void TSI_StartSoftwareTrigger (TSI_Type ∗base)
  Start one sotware trigger measurement (trigger a new measurement).
- static void TSI_SetSelfCapMeasuredChannel (TSI_Type ∗base, uint8_t channel)
  Set the measured channel number for self-cap mode.
- static uint8_t TSI_GetSelfCapMeasuredChannel (TSI_Type ∗base)
  Get the current measured channel number, in self-cap mode.
- static void TSI_EnableDmaTransfer (TSI_Type ∗base, bool enable)
  Enable DMA transfer or not.
- static void TSI_EnableEndOfScanDmaTransferOnly (TSI_Type ∗base, bool enable)
  Decide whether to enable End of Scan DMA transfer request only.
- static uint16_t TSI_GetCounter (TSI_Type ∗base)
  Gets the conversion counter value.
- static void TSI_SetLowThreshold (TSI_Type ∗base, uint16_t low_threshold)
  Set the TSI wake-up channel low threshold.
- static void TSI_SetHighThreshold (TSI_Type ∗base, uint16_t high_threshold)
  Set the TSI wake-up channel high threshold.
- static void TSI_SetMainClock (TSI_Type ∗base, tsi_main_clock_selection_t mainClock)
  Set the main clock of the TSI module.
- static void TSI_SetSensingMode (TSI_Type ∗base, tsi_sensing_mode_selection_t mode)
  Set the sensing mode of the TSI module.
- static tsi_sensing_mode_selection_t TSI_GetSensingMode (TSI_Type ∗base)
  Get the sensing mode of the TSI module.

- static void TSI_SetDvolt (TSI_Type ∗base, tsi_dvolt_option_t dvolt)
  Set the DVOLT settings.
- static void TSI_EnableNoiseCancellation (TSI_Type ∗base, bool enableCancellation)
  Enable self-cap mode noise cancellation function or not.
- static void TSI_SetMutualCapTxChannel (TSI_Type ∗base, tsi_mutual_tx_channel_t txChannel)
  Set the mutual-cap mode TX channel.
- static tsi_mutual_tx_channel_t TSI_GetTxMutualCapMeasuredChannel (TSI_Type ∗base)
  Get the current measured TX channel number, in mutual-cap mode.
- static void TSI_SetMutualCapRxChannel (TSI_Type ∗base, tsi_mutual_rx_channel_t rxChannel)
  Set the mutual-cap mode RX channel.
- static tsi_mutual_rx_channel_t TSI_GetRxMutualCapMeasuredChannel (TSI_Type ∗base)
  Get the current measured RX channel number, in mutual-cap mode.
- static void TSI_SetSscMode (TSI_Type ∗base, tsi_ssc_mode_t mode)
  Set the SSC clock mode of the TSI module.
- static void TSI_SetSscPrescaler (TSI_Type ∗base, tsi_ssc_prescaler_t prescaler)
  Set the SSC prescaler of the TSI module.
- static void TSI_SetUsedTxChannel (TSI_Type ∗base, tsi_mutual_tx_channel_t txChannel)
  Set used mutual-cap TX channel.
- static void TSI_ClearUsedTxChannel (TSI_Type ∗base, tsi_mutual_tx_channel_t txChannel)
  Clear used mutual-cap TX channel.

35.2 Data Structure Documentation

35.2.1 struct tsi_calibration_data_t

Data Fields

- uint16_t calibratedData [FSL_FEATURE_TSI_CHANNEL_COUNT]
  TSI calibration data storage buffer.

35.2.2 struct tsi_common_config_t

This structure contains the common settings for TSI self-cap or mutual-cap mode, configurations including the TSI module main clock, sensing mode, DVOLT options, SINC and SSC configurations.

Data Fields

- tsi_main_clock_selection_t mainClock
  Set main clock.
- tsi_sensing_mode_selection_t mode
  Choose sensing mode.
- tsi_dvolt_option_t dvolt
  DVOLT option value.
- tsi_sinc_cutoff_div_t cutoff
  Cutoff divider.
- tsi_sinc_filter_order_t order
  SINC filter order.

- tsi_sinc_decimation_value_t decimation
  SINC decimation value.
- tsi_ssc_charge_num_t chargeNum
  SSC High Width (t1), SSC output bit0's period setting.
- tsi_ssc_prbs_outsel_t prbsOutsel
  SSC High Random Width (t2), length of PRBS(Pseudo-RandomBinarySequence),SSC output bit2's period setting.
- tsi_ssc_nocharge_num_t noChargeNum
  SSC Low Width (t3), SSC output bit1's period setting.
- tsi_ssc_mode_t ssc_mode
  Clock mode selection (basic - from main clock by divider,advanced - using SSC(Switching Speed Clock) by three configurable intervals.
- tsi_ssc_prescaler_t ssc_prescaler
  Set clock divider for basic mode.

Field Documentation

(1) tsi_main_clock_selection_t tsi_common_config_t::mainClock

(2) tsi_sensing_mode_selection_t tsi_common_config_t::mode

(3) tsi_dvolt_option_t tsi_common_config_t::dvolt

(4) tsi_sinc_cutoff_div_t tsi_common_config_t::cutoff

(5) tsi_sinc_filter_order_t tsi_common_config_t::order

(6) tsi_sinc_decimation_value_t tsi_common_config_t::decimation

(7) tsi_ssc_charge_num_t tsi_common_config_t::chargeNum

(8) tsi_ssc_prbs_outsel_t tsi_common_config_t::prbsOutsel

(9) tsi_ssc_nocharge_num_t tsi_common_config_t::noChargeNum

(10) tsi_ssc_mode_t tsi_common_config_t::ssc_mode

(11) tsi_ssc_prescaler_t tsi_common_config_t::ssc_prescaler

35.2.3 struct tsi_selfCap_config_t

This structure contains the settings for the most common TSI self-cap configurations including the TSI module charge currents, sensitivity configuration and so on.

Data Fields

- tsi_common_config_t commonConfig
  Common settings.
- bool enableSensitivity

Enable sensitivity boost of self-cap or not.
- tsi_shield_t enableShield
  Enable shield of self-cap mode or not.
- tsi_sensitivity_xdn_option_t xdn
  Sensitivity XDN option.
- tsi_sensitivity_ctrim_option_t ctrim
  Sensitivity CTRIM option.
- tsi_current_multiple_input_t inputCurrent
  Input current multiple.
- tsi_current_multiple_charge_t chargeCurrent
  Charge/Discharge current multiple.

Field Documentation

(1) tsi_common_config_t tsi_selfCap_config_t::commonConfig

(2) bool tsi_selfCap_config_t::enableSensitivity

(3) tsi_shield_t tsi_selfCap_config_t::enableShield

(4) tsi_sensitivity_xdn_option_t tsi_selfCap_config_t::xdn

(5) tsi_sensitivity_ctrim_option_t tsi_selfCap_config_t::ctrim

(6) tsi_current_multiple_input_t tsi_selfCap_config_t::inputCurrent

(7) tsi_current_multiple_charge_t tsi_selfCap_config_t::chargeCurrent

35.2.4 struct tsi_mutualCap_config_t

This structure contains the settings for the most common TSI mutual-cap configurations including the TSI module generator settings, sensitivity related current settings and so on.

Data Fields

- tsi_common_config_t commonConfig
  Common settings.
- tsi_mutual_pre_current_t preCurrent
  Vref generator current.
- tsi_mutual_pre_resistor_t preResistor
  Vref generator resistor.
- tsi_mutual_sense_resistor_t senseResistor
  I-sense generator resistor.
- tsi_mutual_sense_boost_current_t boostCurrent
  Sensitivity boost current setting.
- tsi_mutual_tx_drive_mode_t txDriveMode
  TX drive mode control setting.
- tsi_mutual_pmos_current_left_t pmosLeftCurrent
  Pmos current mirror on the left side.

- tsi_mutual_pmos_current_right_t pmosRightCurrent
  Pmos current mirror on the right side.
- bool enableNmosMirror
  Enable Nmos current mirror setting or not.
- tsi_mutual_nmos_current_t nmosCurrent
  Nmos current mirror setting.

Field Documentation

(1) tsi_common_config_t tsi_mutualCap_config_t::commonConfig

(2) tsi_mutual_pre_current_t tsi_mutualCap_config_t::preCurrent

(3) tsi_mutual_pre_resistor_t tsi_mutualCap_config_t::preResistor

(4) tsi_mutual_sense_resistor_t tsi_mutualCap_config_t::senseResistor

(5) tsi_mutual_sense_boost_current_t tsi_mutualCap_config_t::boostCurrent

(6) tsi_mutual_tx_drive_mode_t tsi_mutualCap_config_t::txDriveMode

(7) tsi_mutual_pmos_current_left_t tsi_mutualCap_config_t::pmosLeftCurrent

(8) tsi_mutual_pmos_current_right_t tsi_mutualCap_config_t::pmosRightCurrent

(9) bool tsi_mutualCap_config_t::enableNmosMirror

(10) tsi_mutual_nmos_current_t tsi_mutualCap_config_t::nmosCurrent

35.3 Enumeration Type Documentation

35.3.1 enum tsi_main_clock_selection_t

These constants set the tsi main clock.

Enumerator

kTSI_MainClockSlection_0 Set TSI main clock frequency to 20.72MHz.
kTSI_MainClockSlection_1 Set TSI main clock frequency to 16.65MHz.
kTSI_MainClockSlection_2 Set TSI main clock frequency to 13.87MHz.
kTSI_MainClockSlection_3 Set TSI main clock frequency to 11.91MHz.

35.3.2 enum tsi_sensing_mode_selection_t

These constants set the tsi sensing mode.

Enumerator

kTSI_SensingModeSlection_Self Set TSI sensing mode to self-cap mode.

kTSI_SensingModeSlection_Mutual Set TSI sensing mode to mutual-cap mode.

35.3.3 enum tsi_dvolt_option_t

These bits indicate the comparator vp, vm and dvolt voltage.

Enumerator

kTSI_DvoltOption_0 DVOLT value option 0, the value may differ on different platforms.
kTSI_DvoltOption_1 DVOLT value option 1, the value may differ on different platforms.
kTSI_DvoltOption_2 DVOLT value option 2, the value may differ on different platforms.
kTSI_DvoltOption_3 DVOLT value option 3, the value may differ on different platforms.

35.3.4 enum tsi_sensitivity_xdn_option_t

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

Enumerator

kTSI_SensitivityXdnOption_0 Adjust sensitivity in self-cap mode, 1/16.
kTSI_SensitivityXdnOption_1 Adjust sensitivity in self-cap mode, 1/8.
kTSI_SensitivityXdnOption_2 Adjust sensitivity in self-cap mode, 1/4.
kTSI_SensitivityXdnOption_3 Adjust sensitivity in self-cap mode, 1/2.
kTSI_SensitivityXdnOption_4 Adjust sensitivity in self-cap mode, 1/1.
kTSI_SensitivityXdnOption_5 Adjust sensitivity in self-cap mode, 2/1.
kTSI_SensitivityXdnOption_6 Adjust sensitivity in self-cap mode, 4/1.
kTSI_SensitivityXdnOption_7 Adjust sensitivity in self-cap mode, 8/1.

35.3.5 enum tsi_shield_t

These constants define the shield pin used for HW shielding functionality. One or more shield pin can be selected. The involved bitfield is not fix can change from device to device (KE16Z7 and KE17Z7 support 3 shield pins, other KE serials only support 1 shield pin).

Enumerator

kTSI_shieldAllOff No pin used.
kTSI_shield0On Shield 0 pin used.
kTSI_shield1On Shield 1 pin used.
kTSI_shield1and0On Shield 0,1 pins used.
kTSI_shield2On Shield 2 pin used.
kTSI_shield2and0On Shield 2,0 pins used.

kTSI_shield2and1On Shield 2,1 pins used.
kTSI_shieldAllOn Shield 2,1,0 pins used.

35.3.6 enum tsi_sensitivity_ctrim_option_t

These constants define the tsi sensitivity ajustment in self-cap mode, when TSI_MODE[S_SEN] = 1.

Enumerator

kTSI_SensitivityCtrimOption_0 Adjust sensitivity in self-cap mode, 2.5p.
kTSI_SensitivityCtrimOption_1 Adjust sensitivity in self-cap mode, 5.0p.
kTSI_SensitivityCtrimOption_2 Adjust sensitivity in self-cap mode, 7.5p.
kTSI_SensitivityCtrimOption_3 Adjust sensitivity in self-cap mode, 10.0p.
kTSI_SensitivityCtrimOption_4 Adjust sensitivity in self-cap mode, 12.5p.
kTSI_SensitivityCtrimOption_5 Adjust sensitivity in self-cap mode, 15.0p.
kTSI_SensitivityCtrimOption_6 Adjust sensitivity in self-cap mode, 17.5p.
kTSI_SensitivityCtrimOption_7 Adjust sensitivity in self-cap mode, 20.0p.

35.3.7 enum tsi_current_multiple_input_t

These constants set the tsi input current multiple in self-cap mode.

Enumerator

kTSI_CurrentMultipleInputValue_0 Adjust input current multiple in self-cap mode, 1/8.
kTSI_CurrentMultipleInputValue_1 Adjust input current multiple in self-cap mode, 1/4.

35.3.8 enum tsi_current_multiple_charge_t

These constants set the tsi charge/discharge current multiple in self-cap mode.

Enumerator

kTSI_CurrentMultipleChargeValue_0 Adjust charge/discharge current multiple in self-cap mode, 1/16.
kTSI_CurrentMultipleChargeValue_1 Adjust charge/discharge current multiple in self-cap mode, 1/8.
kTSI_CurrentMultipleChargeValue_2 Adjust charge/discharge current multiple in self-cap mode, 1/4.
kTSI_CurrentMultipleChargeValue_3 Adjust charge/discharge current multiple in self-cap mode, 1/2.

kTSI_CurrentMultipleChargeValue_4 Adjust charge/discharge current multiple in self-cap mode, 1/1.
kTSI_CurrentMultipleChargeValue_5 Adjust charge/discharge current multiple in self-cap mode, 2/1.
kTSI_CurrentMultipleChargeValue_6 Adjust charge/discharge current multiple in self-cap mode, 4/1.
kTSI_CurrentMultipleChargeValue_7 Adjust charge/discharge current multiple in self-cap mode, 8/1.

35.3.9 enum tsi_mutual_pre_current_t

These constants Choose the current used in vref generator.

Enumerator

kTSI_MutualPreCurrent_1uA Vref generator current is 1uA, used in mutual-cap mode.
kTSI_MutualPreCurrent_2uA Vref generator current is 2uA, used in mutual-cap mode.
kTSI_MutualPreCurrent_3uA Vref generator current is 3uA, used in mutual-cap mode.
kTSI_MutualPreCurrent_4uA Vref generator current is 4uA, used in mutual-cap mode.
kTSI_MutualPreCurrent_5uA Vref generator current is 5uA, used in mutual-cap mode.
kTSI_MutualPreCurrent_6uA Vref generator current is 6uA, used in mutual-cap mode.
kTSI_MutualPreCurrent_7uA Vref generator current is 7uA, used in mutual-cap mode.
kTSI_MutualPreCurrent_8uA Vref generator current is 8uA, used in mutual-cap mode.

35.3.10 enum tsi_mutual_pre_resistor_t

These constants Choose the resistor used in pre-charge.

Enumerator

kTSI_MutualPreResistor_1k Vref generator resistor is 1k, used in mutual-cap mode.
kTSI_MutualPreResistor_2k Vref generator resistor is 2k, used in mutual-cap mode.
kTSI_MutualPreResistor_3k Vref generator resistor is 3k, used in mutual-cap mode.
kTSI_MutualPreResistor_4k Vref generator resistor is 4k, used in mutual-cap mode.
kTSI_MutualPreResistor_5k Vref generator resistor is 5k, used in mutual-cap mode.
kTSI_MutualPreResistor_6k Vref generator resistor is 6k, used in mutual-cap mode.
kTSI_MutualPreResistor_7k Vref generator resistor is 7k, used in mutual-cap mode.
kTSI_MutualPreResistor_8k Vref generator resistor is 8k, used in mutual-cap mode.

35.3.11 enum tsi_mutual_sense_resistor_t

These constants Choose the resistor used in I-sense generator.

Enumerator

kTSI_MutualSenseResistor_2k5 I-sense resistor is 2.5k , used in mutual-cap mode.
kTSI_MutualSenseResistor_5k I-sense resistor is 5.0k , used in mutual-cap mode.
kTSI_MutualSenseResistor_7k5 I-sense resistor is 7.5k , used in mutual-cap mode.
kTSI_MutualSenseResistor_10k I-sense resistor is 10.0k, used in mutual-cap mode.
kTSI_MutualSenseResistor_12k5 I-sense resistor is 12.5k, used in mutual-cap mode.
kTSI_MutualSenseResistor_15k I-sense resistor is 15.0k, used in mutual-cap mode.
kTSI_MutualSenseResistor_17k5 I-sense resistor is 17.5k, used in mutual-cap mode.
kTSI_MutualSenseResistor_20k I-sense resistor is 20.0k, used in mutual-cap mode.
kTSI_MutualSenseResistor_22k5 I-sense resistor is 22.5k, used in mutual-cap mode.
kTSI_MutualSenseResistor_25k I-sense resistor is 25.0k, used in mutual-cap mode.
kTSI_MutualSenseResistor_27k5 I-sense resistor is 27.5k, used in mutual-cap mode.
kTSI_MutualSenseResistor_30k I-sense resistor is 30.0k, used in mutual-cap mode.
kTSI_MutualSenseResistor_32k5 I-sense resistor is 32.5k, used in mutual-cap mode.
kTSI_MutualSenseResistor_35k I-sense resistor is 35.0k, used in mutual-cap mode.
kTSI_MutualSenseResistor_37k5 I-sense resistor is 37.5k, used in mutual-cap mode.
kTSI_MutualSenseResistor_40k I-sense resistor is 40.0k, used in mutual-cap mode.

35.3.12 enum tsi_mutual_tx_channel_t

These constants Choose the TX channel used in mutual-cap mode.

Enumerator

kTSI_MutualTxChannel_0 Select channel 0 as tx0, used in mutual-cap mode.
kTSI_MutualTxChannel_1 Select channel 1 as tx1, used in mutual-cap mode.
kTSI_MutualTxChannel_2 Select channel 2 as tx2, used in mutual-cap mode.
kTSI_MutualTxChannel_3 Select channel 3 as tx3, used in mutual-cap mode.
kTSI_MutualTxChannel_4 Select channel 4 as tx4, used in mutual-cap mode.
kTSI_MutualTxChannel_5 Select channel 5 as tx5, used in mutual-cap mode.

35.3.13 enum tsi_mutual_rx_channel_t

These constants Choose the RX channel used in mutual-cap mode.

Enumerator

kTSI_MutualRxChannel_6 Select channel 6 as rx6, used in mutual-cap mode.
kTSI_MutualRxChannel_7 Select channel 7 as rx7, used in mutual-cap mode.
kTSI_MutualRxChannel_8 Select channel 8 as rx8, used in mutual-cap mode.
kTSI_MutualRxChannel_9 Select channel 9 as rx9, used in mutual-cap mode.
kTSI_MutualRxChannel_10 Select channel 10 as rx10, used in mutual-cap mode.
kTSI_MutualRxChannel_11 Select channel 11 as rx11, used in mutual-cap mode.

35.3.14 enum tsi_mutual_sense_boost_current_t

These constants set the sensitivity boost current.

Enumerator

kTSI_MutualSenseBoostCurrent_0uA Sensitivity boost current is 0uA , used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_2uA Sensitivity boost current is 2uA , used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_4uA Sensitivity boost current is 4uA , used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_6uA Sensitivity boost current is 6uA , used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_8uA Sensitivity boost current is 8uA , used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_10uA Sensitivity boost current is 10uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_12uA Sensitivity boost current is 12uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_14uA Sensitivity boost current is 14uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_16uA Sensitivity boost current is 16uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_18uA Sensitivity boost current is 18uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_20uA Sensitivity boost current is 20uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_22uA Sensitivity boost current is 22uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_24uA Sensitivity boost current is 24uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_26uA Sensitivity boost current is 26uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_28uA Sensitivity boost current is 28uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_30uA Sensitivity boost current is 30uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_32uA Sensitivity boost current is 32uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_34uA Sensitivity boost current is 34uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_36uA Sensitivity boost current is 36uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_38uA Sensitivity boost current is 38uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_40uA Sensitivity boost current is 40uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_42uA Sensitivity boost current is 42uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_44uA Sensitivity boost current is 44uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_46uA Sensitivity boost current is 46uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_48uA Sensitivity boost current is 48uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_50uA Sensitivity boost current is 50uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_52uA Sensitivity boost current is 52uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_54uA Sensitivity boost current is 54uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_56uA Sensitivity boost current is 56uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_58uA Sensitivity boost current is 58uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_60uA Sensitivity boost current is 60uA, used in mutual-cap mode.

kTSI_MutualSenseBoostCurrent_62uA Sensitivity boost current is 62uA, used in mutual-cap mode.

35.3.15 enum tsi_mutual_tx_drive_mode_t

These constants Choose the TX drive mode control setting.

Enumerator

kTSI_MutualTxDriveModeOption_0 TX drive mode is -5v $\sim$ +5v, used in mutual-cap mode.

kTSI_MutualTxDriveModeOption_1 TX drive mode is 0v $\sim$ +5v, used in mutual-cap mode.

35.3.16 enum tsi_mutual_pmos_current_left_t

These constants set the Pmos current mirror on the left side used in mutual-cap mode.

Enumerator

kTSI_MutualPmosCurrentMirrorLeft_4 Set Pmos current mirror left value as 4, used in mutual-cap mode.

kTSI_MutualPmosCurrentMirrorLeft_8 Set Pmos current mirror left value as 8, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorLeft_12 Set Pmos current mirror left value as 12, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorLeft_16 Set Pmos current mirror left value as 16, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorLeft_20 Set Pmos current mirror left value as 20, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorLeft_24 Set Pmos current mirror left value as 24, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorLeft_28 Set Pmos current mirror left value as 28, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorLeft_32 Set Pmos current mirror left value as 32, used in mutual-cap mode.

35.3.17 enum tsi_mutual_pmos_current_right_t

These constants set the Pmos current mirror on the right side used in mutual-cap mode.

Enumerator

kTSI_MutualPmosCurrentMirrorRight_1 Set Pmos current mirror right value as 1, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorRight_2 Set Pmos current mirror right value as 2, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorRight_3 Set Pmos current mirror right value as 3, used in mutual-cap mode.
kTSI_MutualPmosCurrentMirrorRight_4 Set Pmos current mirror right value as 4, used in mutual-cap mode.

35.3.18 enum tsi_mutual_nmos_current_t

These constants set the Nmos current mirror used in mutual-cap mode.

Enumerator

kTSI_MutualNmosCurrentMirror_1 Set Nmos current mirror value as 1, used in mutual-cap mode.

kTSI_MutualNmosCurrentMirror_2 Set Nmos current mirror value as 2, used in mutual-cap mode.

kTSI_MutualNmosCurrentMirror_3 Set Nmos current mirror value as 3, used in mutual-cap mode.

kTSI_MutualNmosCurrentMirror_4 Set Nmos current mirror value as 4, used in mutual-cap mode.

35.3.19 enum tsi_sinc_cutoff_div_t

These bits set the SINC cutoff divider.

Enumerator

kTSI_SincCutoffDiv_1 Set SINC cutoff divider as 1.
kTSI_SincCutoffDiv_2 Set SINC cutoff divider as 2.
kTSI_SincCutoffDiv_4 Set SINC cutoff divider as 4.
kTSI_SincCutoffDiv_8 Set SINC cutoff divider as 8.
kTSI_SincCutoffDiv_16 Set SINC cutoff divider as 16.
kTSI_SincCutoffDiv_32 Set SINC cutoff divider as 32.
kTSI_SincCutoffDiv_64 Set SINC cutoff divider as 64.
kTSI_SincCutoffDiv_128 Set SINC cutoff divider as 128.

35.3.20 enum tsi_sinc_filter_order_t

These bits set the SINC filter order.

Enumerator

kTSI_SincFilterOrder_1 Use 1 order SINC filter.
kTSI_SincFilterOrder_2 Use 1 order SINC filter.

35.3.21 enum tsi_sinc_decimation_value_t

These bits set the SINC decimation value.

Enumerator

kTSI_SincDecimationValue_1 The TSI_DATA[TSICH] bits is the counter value of 1 triger period.

kTSI_SincDecimationValue_2 The TSI_DATA[TSICH] bits is the counter value of 2 triger period.

kTSI_SincDecimationValue_3 The TSI_DATA[TSICH] bits is the counter value of 3 triger period.

kTSI_SincDecimationValue_4 The TSI_DATA[TSICH] bits is the counter value of 4 triger period.

kTSI_SincDecimationValue_5 The TSI_DATA[TSICH] bits is the counter value of 5 triger period.

kTSI_SincDecimationValue_6 The TSI_DATA[TSICH] bits is the counter value of 6 triger period.

kTSI_SincDecimationValue_7 The TSI_DATA[TSICH] bits is the counter value of 7 triger period.

kTSI_SincDecimationValue_8 The TSI_DATA[TSICH] bits is the counter value of 8 triger period.

kTSI_SincDecimationValue_9 The TSI_DATA[TSICH] bits is the counter value of 9 triger period.

kTSI_SincDecimationValue_10 The TSI_DATA[TSICH] bits is the counter value of 10 triger period.

kTSI_SincDecimationValue_11 The TSI_DATA[TSICH] bits is the counter value of 11 triger period.

kTSI_SincDecimationValue_12 The TSI_DATA[TSICH] bits is the counter value of 12 triger period.

kTSI_SincDecimationValue_13 The TSI_DATA[TSICH] bits is the counter value of 13 triger period.

kTSI_SincDecimationValue_14 The TSI_DATA[TSICH] bits is the counter value of 14 triger period.

kTSI_SincDecimationValue_15 The TSI_DATA[TSICH] bits is the counter value of 15 triger period.

kTSI_SincDecimationValue_16 The TSI_DATA[TSICH] bits is the counter value of 16 triger period.

kTSI_SincDecimationValue_17 The TSI_DATA[TSICH] bits is the counter value of 17 triger period.

kTSI_SincDecimationValue_18 The TSI_DATA[TSICH] bits is the counter value of 18 triger period.

kTSI_SincDecimationValue_19 The TSI_DATA[TSICH] bits is the counter value of 19 triger period.

kTSI_SincDecimationValue_20 The TSI_DATA[TSICH] bits is the counter value of 20 triger period.

kTSI_SincDecimationValue_21 The TSI_DATA[TSICH] bits is the counter value of 21 triger period.

kTSI_SincDecimationValue_22 The TSI_DATA[TSICH] bits is the counter value of 22 triger period.

kTSI_SincDecimationValue_23 The TSI_DATA[TSICH] bits is the counter value of 23 triger period.

kTSI_SincDecimationValue_24 The TSI_DATA[TSICH] bits is the counter value of 24 triger period.

kTSI_SincDecimationValue_25 The TSI_DATA[TSICH] bits is the counter value of 25 triger period.

kTSI_SincDecimationValue_26 The TSI_DATA[TSICH] bits is the counter value of 26 triger period.

kTSI_SincDecimationValue_27 The TSI_DATA[TSICH] bits is the counter value of 27 triger period.

kTSI_SincDecimationValue_28 The TSI_DATA[TSICH] bits is the counter value of 28 triger period.

kTSI_SincDecimationValue_29 The TSI_DATA[TSICH] bits is the counter value of 29 triger period.

kTSI_SincDecimationValue_30 The TSI_DATA[TSICH] bits is the counter value of 30 triger period.

kTSI_SincDecimationValue_31 The TSI_DATA[TSICH] bits is the counter value of 31 triger period.

kTSI_SincDecimationValue_32 The TSI_DATA[TSICH] bits is the counter value of 32 triger period.

35.3.22 enum tsi_ssc_charge_num_t

These bits set the SSC output bit0's period setting.

Enumerator

kTSI_SscChargeNumValue_1 The SSC output bit 0's period will be 1 clock cycle of system clock.

kTSI_SscChargeNumValue_2 The SSC output bit 0's period will be 2 clock cycle of system clock.

kTSI_SscChargeNumValue_3 The SSC output bit 0's period will be 3 clock cycle of system clock.

kTSI_SscChargeNumValue_4 The SSC output bit 0's period will be 4 clock cycle of system clock.

kTSI_SscChargeNumValue_5 The SSC output bit 0's period will be 5 clock cycle of system clock.

kTSI_SscChargeNumValue_6 The SSC output bit 0's period will be 6 clock cycle of system clock.

kTSI_SscChargeNumValue_7 The SSC output bit 0's period will be 7 clock cycle of system clock.

kTSI_SscChargeNumValue_8 The SSC output bit 0's period will be 8 clock cycle of system clock.

kTSI_SscChargeNumValue_9 The SSC output bit 0's period will be 9 clock cycle of system clock.

kTSI_SscChargeNumValue_10 The SSC output bit 0's period will be 10 clock cycle of system clock.

kTSI_SscChargeNumValue_11 The SSC output bit 0's period will be 11 clock cycle of system clock.

kTSI_SscChargeNumValue_12 The SSC output bit 0's period will be 12 clock cycle of system clock.

kTSI_SscChargeNumValue_13 The SSC output bit 0's period will be 13 clock cycle of system clock.

kTSI_SscChargeNumValue_14 The SSC output bit 0's period will be 14 clock cycle of system clock.

kTSI_SscChargeNumValue_15 The SSC output bit 0's period will be 15 clock cycle of system clock.

kTSI_SscChargeNumValue_16 The SSC output bit 0's period will be 16 clock cycle of system clock.

35.3.23 enum tsi_ssc_nocharge_num_t

These bits set the SSC output bit1's period setting.

Enumerator

kTSI_SscNoChargeNumValue_1 The SSC output bit 1's basic period will be 1 clock cycle of system clock.

kTSI_SscNoChargeNumValue_2 The SSC output bit 1's basic period will be 2 clock cycle of system clock.
kTSI_SscNoChargeNumValue_3 The SSC output bit 1's basic period will be 3 clock cycle of system clock.
kTSI_SscNoChargeNumValue_4 The SSC output bit 1's basic period will be 4 clock cycle of system clock.
kTSI_SscNoChargeNumValue_5 The SSC output bit 1's basic period will be 5 clock cycle of system clock.
kTSI_SscNoChargeNumValue_6 The SSC output bit 1's basic period will be 6 clock cycle of system clock.
kTSI_SscNoChargeNumValue_7 The SSC output bit 1's basic period will be 7 clock cycle of system clock.
kTSI_SscNoChargeNumValue_8 The SSC output bit 1's basic period will be 8 clock cycle of system clock.
kTSI_SscNoChargeNumValue_9 The SSC output bit 1's basic period will be 9 clock cycle of system clock.
kTSI_SscNoChargeNumValue_10 The SSC output bit 1's basic period will be 10 clock cycle of system clock.
kTSI_SscNoChargeNumValue_11 The SSC output bit 1's basic period will be 11 clock cycle of system clock.
kTSI_SscNoChargeNumValue_12 The SSC output bit 1's basic period will be 12 clock cycle of system clock.
kTSI_SscNoChargeNumValue_13 The SSC output bit 1's basic period will be 13 clock cycle of system clock.
kTSI_SscNoChargeNumValue_14 The SSC output bit 1's basic period will be 14 clock cycle of system clock.
kTSI_SscNoChargeNumValue_15 The SSC output bit 1's basic period will be 15 clock cycle of system clock.
kTSI_SscNoChargeNumValue_16 The SSC output bit 1's basic period will be 16 clock cycle of system clock.

35.3.24 enum tsi_ssc_prbs_outsel_t

These bits set the SSC PRBS length.

Enumerator

kTSI_SscPrbsOutsel_2 The length of the PRBS is 2.
kTSI_SscPrbsOutsel_3 The length of the PRBS is 3.
kTSI_SscPrbsOutsel_4 The length of the PRBS is 4.
kTSI_SscPrbsOutsel_5 The length of the PRBS is 5.
kTSI_SscPrbsOutsel_6 The length of the PRBS is 6.
kTSI_SscPrbsOutsel_7 The length of the PRBS is 7.
kTSI_SscPrbsOutsel_8 The length of the PRBS is 8.

kTSI_SscPrbsOutsel_9 The length of the PRBS is 9.
kTSI_SscPrbsOutsel_10 The length of the PRBS is 10.
kTSI_SscPrbsOutsel_11 The length of the PRBS is 11.
kTSI_SscPrbsOutsel_12 The length of the PRBS is 12.
kTSI_SscPrbsOutsel_13 The length of the PRBS is 13.
kTSI_SscPrbsOutsel_14 The length of the PRBS is 14.
kTSI_SscPrbsOutsel_15 The length of the PRBS is 15.

35.3.25 enum tsi_status_flags_t

Enumerator

kTSI_EndOfScanFlag End-Of-Scan flag.
kTSI_OutOfRangeFlag Out-Of-Range flag.

35.3.26 enum tsi_interrupt_enable_t

Enumerator

kTSI_GlobalInterruptEnable TSI module global interrupt.
kTSI_OutOfRangeInterruptEnable Out-Of-Range interrupt.
kTSI_EndOfScanInterruptEnable End-Of-Scan interrupt.

35.3.27 enum tsi_ssc_mode_t

These constants set the SSC mode.

Enumerator

kTSI_ssc_prbs_method Using PRBS method generating SSC output bit.
kTSI_ssc_up_down_counter Using up-down counter generating SSC output bit.
kTSI_ssc_dissable SSC function is disabled.

35.3.28 enum tsi_ssc_prescaler_t

These constants set select the divider ratio for the clock used for generating the SSC output bit.

Enumerator

kTSI_ssc_div_by_1 Set SSC divider to 00000000 div1(2^0)

kTSI_ssc_div_by_2 Set SSC divider to 00000001 div2(2^1)
kTSI_ssc_div_by_4 Set SSC divider to 00000011 div4(2^2)
kTSI_ssc_div_by_8 Set SSC divider to 00000111 div8(2^3)
kTSI_ssc_div_by_16 Set SSC divider to 00001111 div16(2^4)
kTSI_ssc_div_by_32 Set SSC divider to 00011111 div32(2^5)
kTSI_ssc_div_by_64 Set SSC divider to 00111111 div64(2^6)
kTSI_ssc_div_by_128 Set SSC divider to 01111111 div128(2^7)
kTSI_ssc_div_by_256 Set SSC divider to 11111111 div256(2^8)

35.4 Function Documentation

35.4.1 uint32_t TSI_GetInstance (TSI_Type * *base*)

Parameters

base	TSI peripheral base address.

Returns

TSI instance.

35.4.2 void TSI_InitSelfCapMode (TSI_Type * *base,* const tsi_selfCap_config_t * *config*)

Initialize the peripheral to the targeted state specified by parameter config, such as sets sensitivity adjustment, current settings.

Parameters

base	TSI peripheral base address.
config	Pointer to TSI self-cap configuration structure.

Returns

none

35.4.3 void TSI_InitMutualCapMode (TSI_Type * *base,* const tsi_mutualCap_config_t * *config*)

Initialize the peripheral to the targeted state specified by parameter config, such as sets Vref generator setting, sensitivity boost settings, Pmos/Nmos settings.

Parameters

base	TSI peripheral base address.
config	Pointer to TSI mutual-cap configuration structure.

Returns

none

35.4.4 void TSI_Deinit (TSI_Type * *base*)

De-initialize the peripheral to default state.

Parameters

base	TSI peripheral base address.

Returns

none

35.4.5 void TSI_GetSelfCapModeDefaultConfig (tsi_selfCap_config_t * *userConfig*)

This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to a value:

```
userConfig->commonConfig.mainClock      = kTSI_MainClockSlection_0;
userConfig->commonConfig.mode           = kTSI_SensingModeSlection_Self;
userConfig->commonConfig.dvolt          = kTSI_DvoltOption_2;
userConfig->commonConfig.cutoff         = kTSI_SincCutoffDiv_1;
userConfig->commonConfig.order          = kTSI_SincFilterOrder_1;
userConfig->commonConfig.decimation     = kTSI_SincDecimationValue_8;
userConfig->commonConfig.chargeNum      = kTSI_SscChargeNumValue_3;
userConfig->commonConfig.prbsOutsel     = kTSI_SscPrbsOutsel_2;
userConfig->commonConfig.noChargeNum    = kTSI_SscNoChargeNumValue_2;
userConfig->commonConfig.ssc_mode       = kTSI_ssc_prbs_method;
userConfig->commonConfig.ssc_prescaler = kTSI_ssc_div_by_1;
userConfig->enableSensitivity           = true;
userConfig->enableShield                = false;
userConfig->xdn                         = kTSI_SensitivityXdnOption_1;
userConfig->ctrim                       = kTSI_SensitivityCtrimOption_7;
userConfig->inputCurrent                = kTSI_CurrentMultipleInputValue_0;
userConfig->chargeCurrent               = kTSI_CurrentMultipleChargeValue_1
      ;
```

Parameters

userConfig	Pointer to TSI user configure structure.

35.4.6 void TSI_GetMutualCapModeDefaultConfig (tsi_mutualCap_config_t * *userConfig*)

This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to a value:

```
userConfig->commonConfig.mainClock      = kTSI_MainClockSlection_1;
userConfig->commonConfig.mode           = kTSI_SensingModeSlection_Mutual;
userConfig->commonConfig.dvolt          = kTSI_DvoltOption_0;
userConfig->commonConfig.cutoff         = kTSI_SincCutoffDiv_1;
userConfig->commonConfig.order          = kTSI_SincFilterOrder_1;
userConfig->commonConfig.decimation     = kTSI_SincDecimationValue_8;
userConfig->commonConfig.chargeNum      = kTSI_SscChargeNumValue_4;
userConfig->commonConfig.prbsOutsel     = kTSI_SscPrbsOutsel_2;
userConfig->commonConfig.noChargeNum    = kTSI_SscNoChargeNumValue_5;
userConfig->commonConfig.ssc_mode       = kTSI_ssc_prbs_method;
userConfig->commonConfig.ssc_prescaler = kTSI_ssc_div_by_1;
userConfig->preCurrent                  = kTSI_MutualPreCurrent_4uA;
userConfig->preResistor                 = kTSI_MutualPreResistor_4k;
userConfig->senseResistor               = kTSI_MutualSenseResistor_10k;
userConfig->boostCurrent                = kTSI_MutualSenseBoostCurrent_0uA;
userConfig->txDriveMode                 = kTSI_MutualTxDriveModeOption_0;
userConfig->pmosLeftCurrent             = kTSI_MutualPmosCurrentMirrorLeft_32
      ;
userConfig->pmosRightCurrent            = kTSI_MutualPmosCurrentMirrorRight_1
      ;
userConfig->enableNmosMirror            = true;
userConfig->nmosCurrent                 = kTSI_MutualNmosCurrentMirror_1;
```

Parameters

userConfig	Pointer to TSI user configure structure.

35.4.7 void TSI_SelfCapCalibrate (TSI_Type * *base*, tsi_calibration_data_t * *calBuff*)

Calibrate the peripheral to fetch the initial counter value of the enabled channels. This API is mostly used at initial application setup, it shall be called after the TSI_Init API, then user can use the calibrated counter values to setup applications(such as to determine under which counter value we can confirm a touch event occurs).

Parameters

base	TSI peripheral base address.
calBuff	Data buffer that store the calibrated counter value.

Returns

none

Note

This API is mainly used for self-cap mode;
The calibration work in mutual-cap mode shall be done in applications due to different board layout.

35.4.8 void TSI_EnableInterrupts (TSI_Type * *base*, uint32_t *mask*)

Parameters

base	TSI peripheral base address.
mask	interrupt source The parameter can be combination of the following source if defined: • kTSI_GlobalInterruptEnable • kTSI_EndOfScanInterruptEnable • kTSI_OutOfRangeInterruptEnable

35.4.9 void TSI_DisableInterrupts (TSI_Type * *base*, uint32_t *mask*)

Parameters

base	TSI peripheral base address.
mask	interrupt source The parameter can be combination of the following source if defined: • kTSI_GlobalInterruptEnable • kTSI_EndOfScanInterruptEnable • kTSI_OutOfRangeInterruptEnable

35.4.10 static uint32_t TSI_GetStatusFlags (TSI_Type * *base*) [inline], [static]

This function get tsi interrupt flags.

Parameters

base	TSI peripheral base address.

Returns

The mask of these status flags combination.

35.4.11 void TSI_ClearStatusFlags (TSI_Type * *base,* uint32_t *mask*)

This function clear tsi interrupt flag, automatically cleared flags can not be cleared by this function.

Parameters

base	TSI peripheral base address.
mask	The status flags to clear.

35.4.12 static uint32_t TSI_GetScanTriggerMode (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

Scan trigger mode.

35.4.13 static bool TSI_IsScanInProgress (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

True - scan is in progress. False - scan is not in progress.

35.4.14 static void TSI_EnableModule (TSI_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose whether to enable or disable module; • true Enable TSI module; • false Disable TSI module;

Returns

none.

35.4.15 static void TSI_EnableLowPower (TSI_Type ∗ *base*, bool *enable*) [inline], [static]

```
This enables TSI module function in low power modes.
```

Parameters

base	TSI peripheral base address.
enable	Choose to enable or disable STOP mode. • true Enable module in STOP mode; • false Disable module in STOP mode;

Returns

none.

35.4.16 static void TSI_EnableHardwareTriggerScan (TSI_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose to enable hardware trigger or software trigger scan. • true Enable hardware trigger scan; • false Enable software trigger scan;

Returns

none.

35.4.17 static void TSI_StartSoftwareTrigger (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

none.

35.4.18 static void TSI_SetSelfCapMeasuredChannel (TSI_Type * *base,* uint8_t *channel*) [inline], [static]

Parameters

base	TSI peripheral base address.
channel	Channel number 0 ... 24.

Returns

none.

Note

This API can only be used in self-cap mode!

35.4.19 static uint8_t TSI_GetSelfCapMeasuredChannel (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

uint8_t Channel number 0 ... 24.

Note

This API can only be used in self-cap mode!

35.4.20 static void TSI_EnableDmaTransfer (TSI_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose to enable DMA transfer or not. • true Enable DMA transfer; • false Disable DMA transfer;

Returns

none.

35.4.21 static void TSI_EnableEndOfScanDmaTransferOnly (TSI_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose whether to enable End of Scan DMA transfer request only. • true Enable End of Scan DMA transfer request only; • false Both End-of-Scan and Out-of-Range can generate DMA transfer request.

Returns

none.

35.4.22 static uint16_t TSI_GetCounter (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

Accumulated scan counter value ticked by the reference clock.

35.4.23 static void TSI_SetLowThreshold (TSI_Type ∗ *base*, uint16_t *low_threshold*) [inline], [static]

Parameters

base	TSI peripheral base address.
low_threshold	Low counter threshold.

Returns

none.

35.4.24 static void TSI_SetHighThreshold (TSI_Type ∗ *base*, uint16_t *high_threshold*) [inline], [static]

Parameters

base	TSI peripheral base address.
high_threshold	High counter threshold.

Returns

none.

35.4.25 static void TSI_SetMainClock (TSI_Type ∗ *base*, tsi_main_clock_selection_t *mainClock*) [inline], [static]

Parameters

base	TSI peripheral base address.
mainClock	clock option value.

Returns

none.

35.4.26 static void TSI_SetSensingMode (TSI_Type * *base*, tsi_sensing_mode_selection_t *mode*) [inline], [static]

Parameters

base	TSI peripheral base address.
mode	Mode value.

Returns

none.

35.4.27 static tsi_sensing_mode_selection_t TSI_GetSensingMode (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

Currently selected sensing mode.

35.4.28 static void TSI_SetDvolt (TSI_Type * *base*, tsi_dvolt_option_t *dvolt*) [inline], [static]

Parameters

base	TSI peripheral base address.
dvolt	The voltage rails.

Returns

none.

35.4.29 static void TSI_EnableNoiseCancellation (TSI_Type * *base*, bool *enableCancellation*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable-Cancellation	Choose whether to enable noise cancellation in self-cap mode • true Enable noise cancellation; • false Disable noise cancellation;

Returns

none.

35.4.30 static void TSI_SetMutualCapTxChannel (TSI_Type * *base*, tsi_mutual_tx_channel_t *txChannel*) [inline], [static]

Parameters

base	TSI peripheral base address.
txChannel	Mutual-cap mode TX channel number

Returns

none.

35.4.31 static tsi_mutual_tx_channel_t TSI_GetTxMutualCapMeasuredChannel (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address;

Returns

Tx Channel number 0 ... 5;

Note

This API can only be used in mutual-cap mode!

35.4.32 static void TSI_SetMutualCapRxChannel (TSI_Type ∗ *base*, tsi_mutual_rx_channel_t *rxChannel*) [inline], [static]

Parameters

base	TSI peripheral base address.
rxChannel	Mutual-cap mode RX channel number

Returns

none.

35.4.33 static tsi_mutual_rx_channel_t TSI_GetRxMutualCapMeasuredChannel (TSI_Type ∗ *base*) [inline], [static]

Parameters

base	TSI peripheral base address;

Returns

Rx Channel number 6 ... 11;

Note

This API can only be used in mutual-cap mode!

35.4.34 static void TSI_SetSscMode (TSI_Type ∗ *base*, tsi_ssc_mode_t *mode*) [inline], [static]

Parameters

base	TSI peripheral base address.
mode	SSC mode option value.

Returns

none.

35.4.35 static void TSI_SetSscPrescaler (TSI_Type * *base*, tsi_ssc_prescaler_t *prescaler*) [inline], [static]

Parameters

base	TSI peripheral base address.
prescaler	SSC prescaler option value.

Returns

none.

35.4.36 static void TSI_SetUsedTxChannel (TSI_Type * *base*, tsi_mutual_tx_channel_t *txChannel*) [inline], [static]

Parameters

base	TSI peripheral base address.
txChannel	Mutual-cap mode TX channel number

Returns

none.

35.4.37 static void TSI_ClearUsedTxChannel (TSI_Type * *base*, tsi_mutual_tx_channel_t *txChannel*) [inline], [static]

Parameters

base	TSI peripheral base address.
txChannel	Mutual-cap mode TX channel number

Returns

none.

Chapter 36
GenericList

36.1 Overview

Data Structures

- struct list_handle_t
  The list structure. More...
- struct list_element_handle_t
  The list element. More...

Macros

- #define GENERIC_LIST_LIGHT (1)
  Definition to determine whether use list light.
- #define GENERIC_LIST_DUPLICATED_CHECKING (0)
  Definition to determine whether enable list duplicated checking.

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),
  kLIST_NotSupport = MAKE_STATUS(kStatusGroup_LIST, 5) }
  The list status.

Functions

- void LIST_Init (list_handle_t list, uint32_t max)
  Initialize the list.
- 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.

36.2 Data Structure Documentation

36.2.1 struct list_label_t

Data Fields

- struct list_element_tag * head
  list head
- struct list_element_tag * tail
  list tail
- uint32_t size
  list size
- uint32_t max
  list max number of elements

36.2.2 struct list_element_t

Data Fields

- struct list_element_tag * next
  next list element
- struct list_label * list
  pointer to the list

36.3 Macro Definition Documentation

36.3.1 #define GENERIC_LIST_LIGHT (1)

36.3.2 #define GENERIC_LIST_DUPLICATED_CHECKING (0)

36.4 Enumeration Type Documentation

36.4.1 enum list_status_t

Enumerator

kLIST_Ok Success.
kLIST_DuplicateError Duplicate Error.
kLIST_Full FULL.
kLIST_Empty Empty.
kLIST_OrphanElement Orphan Element.
kLIST_NotSupport Not Support.

36.5 Function Documentation

36.5.1 void LIST_Init (list_handle_t *list*, uint32_t *max*)

This function initialize the list.

Parameters

list	- List handle to initialize.
max	- Maximum number of elements in list. 0 for unlimited.

36.5.2 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.

36.5.3 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.

36.5.4 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.

36.5.5 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.

36.5.6 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.

36.5.7 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.

36.5.8 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.

36.5.9 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.

36.5.10 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.

36.5.11 uint32_t LIST_GetSize (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

Current	size of the list.

36.5.12 uint32_t LIST_GetAvailableSize (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

Available	size of the list.

36.6 Serial_port_rpmsg

36.6.1 Overview

Macros

- #define SERIAL_PORT_RPMSG_HANDLE_SIZE (HAL_RPMSG_HANDLE_SIZE + 32U)
  serial port uart handle size

36.7 Serial_port_uart

36.7.1 Overview

Data Structures

- struct serial_spi_master_config_t
  spi master user configure structure. More...
- struct serial_spi_slave_config_t
  spi slave user configure structure. More...
- struct serial_spi_transfer_t
  spi transfer structure More...

Macros

- #define SERIAL_PORT_SPI_MASTER_HANDLE_SIZE (HAL_SPI_MASTER_HANDLE_SIZ-E)
  serial port uart handle size
- #define SERIAL_USE_CONFIGURE_STRUCTURE (0U)
  Enable or disable the confgure structure pointer.
- #define SERIAL_PORT_UART_DMA_RECEIVE_DATA_LENGTH (64U)
  serial port uart handle size
- #define SERIAL_USE_CONFIGURE_STRUCTURE (0U)
  Enable or disable the confgure structure pointer.

Enumerations

- enum serial_spi_clock_polarity_t {
  kSerial_SpiClockPolarityActiveHigh = 0x0U,
  kSerial_SpiClockPolarityActiveLow }
  spi clock polarity configuration.
- enum serial_spi_clock_phase_t {
  kSerial_SpiClockPhaseFirstEdge = 0x0U,
  kSerial_SpiClockPhaseSecondEdge }
  spi clock phase configuration.
- enum serial_spi_shift_direction_t {
  kSerial_SpiMsbFirst = 0x0U,
  kSerial_SpiLsbFirst }
  spi data shifter direction options.
- enum serial_port_uart_parity_mode_t {
  kSerialManager_UartParityDisabled = 0x0U,
  kSerialManager_UartParityEven = 0x2U,
  kSerialManager_UartParityOdd = 0x3U }
  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

36.7.2 Data Structure Documentation

36.7.2.1 struct serial_spi_master_config_t

Data Fields

- uint32_t srcClock_Hz
  Clock source for spi in Hz.
- uint32_t baudRate_Bps
  Baud Rate for spi in Hz.
- serial_spi_clock_polarity_t polarity
  Clock polarity.
- serial_spi_clock_phase_t phase
  Clock phase.
- serial_spi_shift_direction_t direction
  MSB or LSB.
- uint8_t instance
  Instance of the spi.
- bool enableMaster
  Enable spi at initialization time.
- uint32_t configFlags
  Transfer config Flags.

36.7.2.2 struct serial_spi_slave_config_t

Data Fields

- hal_spi_clock_polarity_t polarity
  Clock polarity.
- hal_spi_clock_phase_t phase
  Clock phase.
- hal_spi_shift_direction_t direction
  MSB or LSB.
- uint8_t instance
  Instance of the spi.
- bool enableSlave
  Enable spi at initialization time.
- uint32_t configFlags
  Transfer config Flags.

36.7.2.3 struct serial_spi_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- size_t dataSize
  Transfer bytes.
- uint32_t flags
  spi control flag.

Field Documentation

(1) uint32_t serial_spi_transfer_t::flags

36.7.3 Enumeration Type Documentation

36.7.3.1 enum serial_spi_clock_polarity_t

Enumerator

kSerial_SpiClockPolarityActiveHigh Active-high spi clock (idles low).
kSerial_SpiClockPolarityActiveLow Active-low spi clock (idles high).

36.7.3.2 enum serial_spi_clock_phase_t

Enumerator

kSerial_SpiClockPhaseFirstEdge First edge on SPSCK occurs at the middle of the first cycle of a data transfer.
kSerial_SpiClockPhaseSecondEdge First edge on SPSCK occurs at the start of the first cycle of a data transfer.

36.7.3.3 enum serial_spi_shift_direction_t

Enumerator

kSerial_SpiMsbFirst Data transfers start with most significant bit.
kSerial_SpiLsbFirst Data transfers start with least significant bit.

36.7.3.4 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.

kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

36.7.3.5 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

36.8 Serial_port_swo

36.8.1 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

36.8.2 Data Structure Documentation

36.8.2.1 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.

36.8.3 Enumeration Type Documentation

36.8.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

36.9 Serial_port_usb

36.9.1 Overview

Data Structures

- struct serial_port_usb_cdc_config_t
  serial port usb config struct More...

Macros

- #define SERIAL_PORT_USB_CDC_HANDLE_SIZE (72U)
  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.

36.9.2 Data Structure Documentation

36.9.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- serial_port_usb_cdc_controller_index_t controllerIndex
  controller index

36.9.3 Enumeration Type Documentation

36.9.3.1 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.

36.10 Serial_port_virtual

36.10.1 Overview

Data Structures

- struct serial_port_virtual_config_t
  serial port usb config struct More...

Macros

- #define SERIAL_PORT_VIRTUAL_HANDLE_SIZE (40U)
  serial port USB handle size

Enumerations

- enum serial_port_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.

36.10.2 Data Structure Documentation

36.10.2.1 struct serial_port_virtual_config_t

Data Fields

- serial_port_virtual_controller_index_t controllerIndex
  controller index

36.10.3 Enumeration Type Documentation

36.10.3.1 enum serial_port_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 37
Usb_device_configuration

37.1 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_CDC_RNDIS** (0U)
- #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.

37.2 Macro Definition Documentation

37.2.1 #define USB_DEVICE_CONFIG_SELF_POWER (1U)

1U supported, 0U not supported

37.2.2 #define USB_DEVICE_CONFIG_ENDPOINTS (4U)

37.2.3 #define USB_DEVICE_CONFIG_USE_TASK (0U)

37.2.4 #define USB_DEVICE_CONFIG_MAX_MESSAGES (8U)

37.2.5 #define USB_DEVICE_CONFIG_USB20_TEST_MODE (0U)

37.2.6 #define USB_DEVICE_CONFIG_CV_TEST (0U)

37.2.7 #define USB_DEVICE_CONFIG_COMPLIANCE_TEST (0U)

If the macro is enabled, the test mode and CV test macroes will be set.

37.2.8 #define USB_DEVICE_CONFIG_KEEP_ALIVE_MODE (0U)

37.2.9 #define USB_DEVICE_CONFIG_BUFFER_PROPERTY_CACHEABLE (0U)

37.2.10 #define USB_DEVICE_CONFIG_LOW_POWER_MODE (0U)

37.2.11 #define USB_DEVICE_CONFIG_REMOTE_WAKEUP (0U)

37.2.12 #define USB_DEVICE_CONFIG_DETACH_ENABLE (0U)

37.2.13 #define USB_DEVICE_CONFIG_ERROR_HANDLING (0U)

37.2.14 #define USB_DEVICE_CHARGER_DETECT_ENABLE (0U)

Chapter 38
UART_Adapter

38.1 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_DMA_INIT_ENABLE (0U)
  Enable or disable master SPI DMA adapter int mode (1 - enable, 0 - disable)
- #define HAL_UART_DMA_IDLELINE_TIMEOUT (1U)
  Definition of uart dma adapter software idleline detection timeout value in ms.
- #define HAL_UART_HANDLE_SIZE (8U + HAL_UART_ADAPTER_LOWPOWER ∗ 16U + HAL_UART_DMA_ENABLE ∗ 4U)
  Definition of uart adapter handle size.
- #define UART_HANDLE_DEFINE(name) uint32_t name[((HAL_UART_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Definition of uart dma adapter handle size.
- #define HAL_UART_TRANSFER_MODE (0U)
  Whether enable transactional function of the UART.

Typedefs

- typedef void ∗ hal_uart_handle_t
  The handle of uart adapter.
- typedef void ∗ hal_uart_dma_handle_t
  The handle of uart dma adapter.
- 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 = 0x2U,
  kHAL_UartParityOdd = 0x3U }
  UART parity mode.
- enum hal_uart_stop_bit_count_t {
  kHAL_UartOneStopBit = 0U,
  kHAL_UartTwoStopBit = 1U }
  UART stop bit count.

Functions

- hal_uart_status_t HAL_UartEnterLowpower (hal_uart_handle_t handle)
  Prepares to enter low power consumption.
- hal_uart_status_t HAL_UartExitLowpower (hal_uart_handle_t handle)
  Restores from low power consumption.

Initialization and deinitialization

- hal_uart_status_t HAL_UartInit (hal_uart_handle_t handle, const 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.

38.2 Data Structure Documentation

38.2.1 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 enableRxRTS
  Enable RX RTS.
- uint8_t enableTxCTS
  Enable TX CTS.
- uint8_t instance
  Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.

Field Documentation

(1) uint8_t hal_uart_config_t::instance

Invalid instance value will cause initialization failure.

38.2.2 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.

Field Documentation

(1) uint8_t* hal_uart_transfer_t::data

(2) size_t hal_uart_transfer_t::dataSize

38.3 Macro Definition Documentation

38.3.1 #define HAL_UART_DMA_IDLELINE_TIMEOUT (1U)

38.3.2 #define HAL_UART_HANDLE_SIZE (8U + HAL_UART_ADAPTER_LOWPOWER * 16U + HAL_UART_DMA_ENABLE * 4U)

38.3.3 #define UART_HANDLE_DEFINE(*name*) uint32_t name[((HAL_UART_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

Defines the uart handle

This macro is used to define a 4 byte aligned uart handle. Then use "(hal_uart_handle_t)name" to get the uart handle.

The macro should be global and could be optional. You could also define uart handle by yourself.

This is an example,

```
* UART_HANDLE_DEFINE(uartHandle);
*
```

Parameters

name	The name string of the uart handle.

38.3.4 #define HAL_UART_TRANSFER_MODE (0U)

(0 - disable, 1 - enable)

38.4 Typedef Documentation

38.4.1 typedef void* hal_uart_handle_t

38.4.2 typedef void* hal_uart_dma_handle_t

38.4.3 typedef void(* hal_uart_transfer_callback_t)(hal_uart_handle_t handle, hal_uart_status_t status, void *callbackParam)

38.5 Enumeration Type Documentation

38.5.1 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.

38.5.2 enum hal_uart_parity_mode_t

Enumerator

kHAL_UartParityDisabled Parity disabled.
kHAL_UartParityEven Parity even enabled.
kHAL_UartParityOdd Parity odd enabled.

38.5.3 enum hal_uart_stop_bit_count_t

Enumerator

kHAL_UartOneStopBit One stop bit.
kHAL_UartTwoStopBit Two stop bits.

38.6 Function Documentation

38.6.1 hal_uart_status_t HAL_UartInit (hal_uart_handle_t *handle*, const 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.

```
*   UART_HANDLE_DEFINE(g_UartHandle);
*   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.enableRxRTS = 0;
*   config.enableTxCTS = 0;
*   config.instance = 0;
*   HAL_UartInit((hal_uart_handle_t)g_UartHandle, &config);
*
```

Parameters

handle	Pointer to point to a memory space of size HAL_UART_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: UART_HANDLE_DEFINE(handle); or uint32_t handle[((HAL_UART_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];
config	Pointer to user-defined configuration structure.

Return values

kStatus_HAL_UartBaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_HAL_UartSuccess	UART initialization succeed

38.6.2 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

38.6.3 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_UartParity-Error	A parity error occurred while receiving data.
kStatus_HAL_Uart-Success	Successfully received all data.

38.6.4 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.

38.6.5 hal_uart_status_t HAL_UartEnterLowpower (hal_uart_handle_t *handle*)

This function is used to prepare to enter low power consumption.

Parameters

handle	UART handle pointer.

Return values

kStatus_HAL_UartSuccess	Successful operation.
kStatus_HAL_UartError	An error occurred.

38.6.6 hal_uart_status_t HAL_UartExitLowpower (hal_uart_handle_t *handle*)

This function is used to restore from low power consumption.

Parameters

handle	UART handle pointer.

Return values

kStatus_HAL_UartSuccess	Successful operation.
kStatus_HAL_UartError	An error occurred.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support