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.

Chapter 1
Introduction

The MCUXpresso Software Development Kit (MCUXpresso SDK) is a collection of software enablement for NXP Microcontrollers that includes peripheral drivers, multicore support and integrated RTOS support for FreeRTOS^{TM} . In addition to the base enablement, the MCUXpresso SDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support provided by MCUXpresso SDK. The MCUXpresso SDK Web Builder is available to provide access to all MCUXpresso SDK packages. See the *MCUXpresso Software Development Kit (SD-K) 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 RT-OS 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 RT-OS 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.

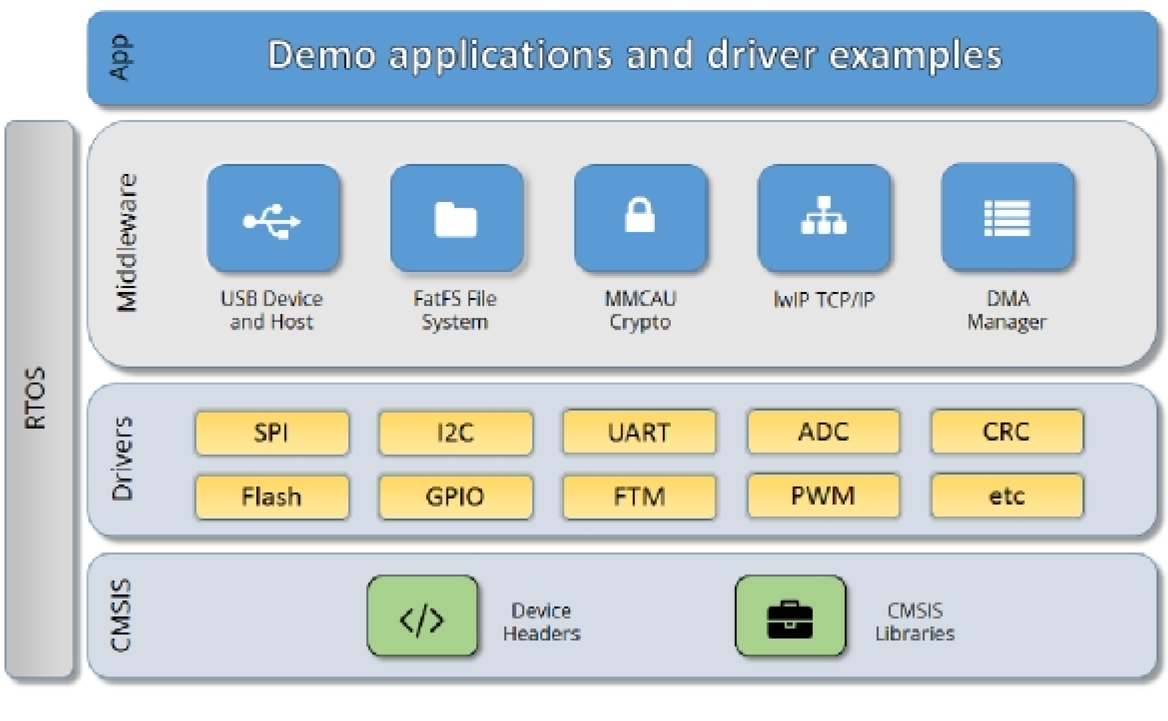
Chapter 3
Architectural Overview

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



MCUXpresso SDK Block Diagram

MCU header files

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

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

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

4.1 Overview

The MCUXpresso SDK provides APIs for MCUXpresso SDK devices' clock operation.

The clock driver supports:

- Clock generator (PLL, FLL, and so on) configuration
- Clock mux and divider configuration
- Getting clock frequency

Modules

- Multipurpose Clock Generator (MCG)

Files

- file fsl_clock.h

Data Structures

- struct sim_clock_config_t
  SIM configuration structure for clock setting. More...
- struct oscer_config_t
  OSC configuration for OSCERCLK. More...
- struct osc_config_t
  OSC Initialization Configuration Structure. More...
- struct mcg_pll_config_t
  MCG PLL configuration. More...
- struct mcg_config_t
  MCG mode change configuration structure. More...

Macros

- #define MCG_CONFIG_CHECK_PARAM 0U
  Configures whether to check a parameter in a function.
- #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0
  Configure whether driver controls clock.
- #define DMAMUX_CLOCKS
  Clock ip name array for DMAMUX.
- #define RTC_CLOCKS
  Clock ip name array for RTC.
- #define SPI_CLOCKS
  Clock ip name array for SPI.
- #define SLCD_CLOCKS

Clock ip name array for SLCD.
- #define EWM_CLOCKS
  Clock ip name array for EWM.
- #define AFE_CLOCKS
  Clock ip name array for AFE.
- #define ADC16_CLOCKS
  Clock ip name array for ADC16.
- #define XBAR_CLOCKS
  Clock ip name array for XBAR.
- #define SYSMPU_CLOCKS
  Clock ip name array for MPU.
- #define VREF_CLOCKS
  Clock ip name array for VREF.
- #define DMA_CLOCKS
  Clock ip name array for DMA.
- #define PORT_CLOCKS
  Clock ip name array for PORT.
- #define UART_CLOCKS
  Clock ip name array for UART.
- #define PIT_CLOCKS
  Clock ip name array for PIT.
- #define RNGA_CLOCKS
  Clock ip name array for RNGA.
- #define CRC_CLOCKS
  Clock ip name array for CRC.
- #define I2C_CLOCKS
  Clock ip name array for I2C.
- #define LPTMR_CLOCKS
  Clock ip name array for LPTMR.
- #define TMR_CLOCKS
  Clock ip name array for TMR.
- #define PDB_CLOCKS
  Clock ip name array for PDB.
- #define FTF_CLOCKS
  Clock ip name array for FTF.
- #define CMP_CLOCKS
  Clock ip name array for CMP.
- #define LPO_CLK_FREQ 1000U
  LPO clock frequency.
- #define SYS_CLK kCLOCK_CoreSysClk
  Peripherals clock source definition.

Enumerations

- enum clock_name_t {
kCLOCK_CoreSysClk,
kCLOCK_PlatClk,
kCLOCK_BusClk,
kCLOCK_FlashClk,
kCLOCK_Er32kClk,
kCLOCK_Osc0ErClk,
kCLOCK_McgFixedFreqClk,
kCLOCK_McgInternalRefClk,
kCLOCK_McgFllClk,
kCLOCK_McgPll0Clk,
kCLOCK_McgExtPllClk,
kCLOCK_McgPeriphClk,
kCLOCK_LpoClk }
  Clock name used to get clock frequency.
- enum clock_ip_name_t
  Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
- enum osc_mode_t {
kOSC_ModeExt = 0U,
kOSC_ModeOscLowPower = MCG_C2_EREFS0_MASK,
kOSC_ModeOscHighGain }
  OSC work mode.
- enum _osc_cap_load {
kOSC_Cap2P = OSC_CR_SC2P_MASK,
kOSC_Cap4P = OSC_CR_SC4P_MASK,
kOSC_Cap8P = OSC_CR_SC8P_MASK,
kOSC_Cap16P = OSC_CR_SC16P_MASK }
  Oscillator capacitor load setting.
- enum _oscer_enable_mode {
kOSC_ErClkEnable = OSC_CR_ERCLKEN_MASK,
kOSC_ErClkEnableInStop = OSC_CR_EREFSTEN_MASK }
  OSCERCLK enable mode.
- enum mcg_fll_src_t {
kMCG_FllSrcExternal,
kMCG_FllSrcInternal }
  MCG FLL reference clock source select.
- enum mcg_irc_mode_t {
kMCG_IrcSlow,
kMCG_IrcFast }
  MCG internal reference clock select.
- enum mcg_dmx32_t {
kMCG_Dmx32Default,
kMCG_Dmx32Fine }
  MCG DCO Maximum Frequency with 32.768 kHz Reference.
- enum mcg_drs_t {

kMCG_DrsLow,
kMCG_DrsMid,
kMCG_DrsMidHigh,
kMCG_DrsHigh }
MCG DCO range select.
- enum mcg_pll_ref_src_t {
kMCG_PllRefRtc,
kMCG_PllRefIrc,
kMCG_PllRefFllRef }
MCG PLL reference clock select.
- enum mcg_clkout_src_t {
kMCG_ClkOutSrcOut,
kMCG_ClkOutSrcInternal,
kMCG_ClkOutSrcExternal }
MCGOUT clock source.
- enum mcg_atm_select_t {
kMCG_AtmSel32k,
kMCG_AtmSel4m }
MCG Automatic Trim Machine Select.
- enum mcg_oscsel_t {
kMCG_OscselOsc,
kMCG_OscselRtc }
MCG OSC Clock Select.
- enum mcg_pll_clk_select_t { kMCG_PllClkSelPll0 }
MCG PLLCS select.
- enum mcg_monitor_mode_t {
kMCG_MonitorNone,
kMCG_MonitorInt,
kMCG_MonitorReset }
MCG clock monitor mode.
- enum {
kStatus_MCG_ModeUnreachable = MAKE_STATUS(kStatusGroup_MCG, 0U),
kStatus_MCG_ModeInvalid = MAKE_STATUS(kStatusGroup_MCG, 1U),
kStatus_MCG_AtmBusClockInvalid = MAKE_STATUS(kStatusGroup_MCG, 2U),
kStatus_MCG_AtmDesiredFreqInvalid = MAKE_STATUS(kStatusGroup_MCG, 3U),
kStatus_MCG_AtmIrcUsed = MAKE_STATUS(kStatusGroup_MCG, 4U),
kStatus_MCG_AtmHardwareFail = MAKE_STATUS(kStatusGroup_MCG, 5U),
kStatus_MCG_SourceUsed = MAKE_STATUS(kStatusGroup_MCG, 6U) }
MCG status.
- enum {
kMCG_Osc0LostFlag = (1U << 0U),
kMCG_Osc0InitFlag = (1U << 1U),
kMCG_RtcOscLostFlag = (1U << 4U),
kMCG_Pll0LostFlag = (1U << 5U),
kMCG_Pll0LockFlag = (1U << 6U) }
MCG status flags.
- enum {

kMCG_IrclkEnable = MCG_C1_IRCLKEN_MASK,
kMCG_IrclkEnableInStop = MCG_C1_IREFSTEN_MASK }
MCG internal reference clock (MCGIRCLK) enable mode definition.
- enum {
kMCG_PllEnableIndependent = MCG_C5_PLLCLKEN0_MASK,
kMCG_PllEnableInStop = MCG_C5_PLLSTEN0_MASK }
MCG PLL clock enable mode definition.
- enum mcg_mode_t {
kMCG_ModeFEI = 0U,
kMCG_ModeFBI,
kMCG_ModeBLPI,
kMCG_ModeFEE,
kMCG_ModeFBE,
kMCG_ModeBLPE,
kMCG_ModePBE,
kMCG_ModePEE,
kMCG_ModePEI,
kMCG_ModePBI,
kMCG_ModeError }
MCG mode definitions.

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_SetEr32kClock (uint32_t src)
Set ERCLK32K source.
- static void CLOCK_SetAfeClkSrc (uint32_t src)
Set the clock selection of AFECLKSEL.
- static void CLOCK_SetClkOutClock (uint32_t src)
Set CLKOUT source.
- static void CLOCK_SetAdcTriggerClock (uint32_t src)
Set ADC trigger clock source.
- uint32_t CLOCK_GetAfeFreq (void)
Gets the clock frequency for AFE 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_GetPlatClkFreq (void)
Get the platform 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_GetEr32kClkFreq (void)
Get the external reference 32K clock frequency (ERCLK32K).

- uint32_t CLOCK_GetOsc0ErClkFreq (void)
  Get the OSC0 external reference clock frequency (OSC0ERCLK).
- void CLOCK_SetSimConfig (sim_clock_config_t const *config)
  Set the clock configure in SIM module.
- static void CLOCK_SetSimSafeDivs (void)
  Set the system clock dividers in SIM to safe value.

Variables

- volatile uint32_t g_xtal0Freq
  External XTAL0 (OSC0) clock frequency.
- volatile uint32_t g_xtal32Freq
  External XTAL32/EXTAL32/RTC_CLKIN clock frequency.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  CLOCK driver version 2.0.0.

MCG frequency functions.

- uint32_t CLOCK_GetOutClkFreq (void)
  Gets the MCG output clock (MCGOUTCLK) frequency.
- uint32_t CLOCK_GetFllFreq (void)
  Gets the MCG FLL clock (MCGFLLCLK) frequency.
- uint32_t CLOCK_GetInternalRefClkFreq (void)
  Gets the MCG internal reference clock (MCGIRCLK) frequency.
- uint32_t CLOCK_GetFixedFreqClkFreq (void)
  Gets the MCG fixed frequency clock (MCGFFCLK) frequency.
- uint32_t CLOCK_GetPll0Freq (void)
  Gets the MCG PLL0 clock (MCGPLL0CLK) frequency.

MCG clock configuration.

- static void CLOCK_SetLowPowerEnable (bool enable)
  Enables or disables the MCG low power.
- status_t CLOCK_SetInternalRefClkConfig (uint8_t enableMode, mcg_irc_mode_t ircs, uint8_-t fcrdiv)
  Configures the Internal Reference clock (MCGIRCLK).
- status_t CLOCK_SetExternalRefClkConfig (mcg_oscsel_t oscsel)
  Selects the MCG external reference clock.
- static void CLOCK_SetFllExtRefDiv (uint8_t frdiv)
  Set the FLL external reference clock divider value.
- void CLOCK_EnablePll0 (mcg_pll_config_t const *config)
  Enables the PLL0 in FLL mode.
- static void CLOCK_DisablePll0 (void)
  Disables the PLL0 in FLL mode.

MCG clock lock monitor functions.

- void CLOCK_SetOsc0MonitorMode (mcg_monitor_mode_t mode)

Sets the OSC0 clock monitor mode.

- void CLOCK_SetRtcOscMonitorMode (mcg_monitor_mode_t mode)
  Sets the RTC OSC clock monitor mode.
- void CLOCK_SetPll0MonitorMode (mcg_monitor_mode_t mode)
  Sets the PLL0 clock monitor mode.
- uint32_t CLOCK_GetStatusFlags (void)
  Gets the MCG status flags.
- void CLOCK_ClearStatusFlags (uint32_t mask)
  Clears the MCG status flags.

OSC configuration

- static void OSC_SetExtRefClkConfig (OSC_Type ∗base, oscer_config_t const ∗config)
  Configures the OSC external reference clock (OSCERCLK).
- static void OSC_SetCapLoad (OSC_Type ∗base, uint8_t capLoad)
  Sets the capacitor load configuration for the oscillator.
- void CLOCK_InitOsc0 (osc_config_t const ∗config)
  Initializes the OSC0.
- void CLOCK_DeinitOsc0 (void)
  Deinitializes the OSC0.

External clock frequency

- static void CLOCK_SetXtal0Freq (uint32_t freq)
  Sets the XTAL0 frequency based on board settings.
- static void CLOCK_SetXtal32Freq (uint32_t freq)
  Sets the XTAL32/RTC_CLKIN frequency based on board settings.

IRCs frequency

- void CLOCK_SetSlowIrcFreq (uint32_t freq)
  Set the Slow IRC frequency based on the trimmed value.
- void CLOCK_SetFastIrcFreq (uint32_t freq)
  Set the Fast IRC frequency based on the trimmed value.

MCG auto-trim machine.

- status_t CLOCK_TrimInternalRefClk (uint32_t extFreq, uint32_t desireFreq, uint32_t ∗actualFreq, mcg_atm_select_t atms)
  Auto trims the internal reference clock.

MCG mode functions.

- mcg_mode_t CLOCK_GetMode (void)
  Gets the current MCG mode.
- status_t CLOCK_SetFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(∗fllStableDelay)(void))
  Sets the MCG to FEI mode.
- status_t CLOCK_SetFeeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(∗fllStableDelay)(void))
  Sets the MCG to FEE mode.

- status_t CLOCK_SetFbiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))
  Sets the MCG to FBI mode.
- status_t CLOCK_SetFbeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))
  Sets the MCG to FBE mode.
- status_t CLOCK_SetBlpiMode (void)
  Sets the MCG to BLPI mode.
- status_t CLOCK_SetBlpeMode (void)
  Sets the MCG to BLPE mode.
- status_t CLOCK_SetPbeMode (mcg_pll_clk_select_t pllcs, mcg_pll_config_t const *config)
  Sets the MCG to PBE mode.
- status_t CLOCK_SetPeeMode (void)
  Sets the MCG to PEE mode.
- status_t CLOCK_SetPbiMode (void)
  Sets the MCG to PBI mode.
- status_t CLOCK_SetPeiMode (void)
  Sets the MCG to PEI mode.
- status_t CLOCK_ExternalModeToFbeModeQuick (void)
  Switches the MCG to FBE mode from the external mode.
- status_t CLOCK_InternalModeToFbiModeQuick (void)
  Switches the MCG to FBI mode from internal modes.
- status_t CLOCK_BootToFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))
  Sets the MCG to FEI mode during system boot up.
- status_t CLOCK_BootToFeeMode (mcg_oscsel_t oscsel, uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))
  Sets the MCG to FEE mode during system bootup.
- status_t CLOCK_BootToBlpiMode (uint8_t fcrdiv, mcg_irc_mode_t ircs, uint8_t ircEnableMode)
  Sets the MCG to BLPI mode during system boot up.
- status_t CLOCK_BootToBlpeMode (mcg_oscsel_t oscsel)
  Sets the MCG to BLPE mode during system boot up.
- status_t CLOCK_BootToPeeMode (mcg_oscsel_t oscsel, mcg_pll_clk_select_t pllcs, mcg_pll_config_t const *config)
  Sets the MCG to PEE mode during system boot up.
- status_t CLOCK_BootToPeiMode (void)
  Sets the MCG to PEI mode during system boot up.
- status_t CLOCK_SetMcgConfig (mcg_config_t const *config)
  Sets the MCG to a target mode.

4.2 Data Structure Documentation

4.2.1 struct sim_clock_config_t

Data Fields

- uint8_t er32kSrc
  ERCLK32K source selection.
- uint32_t clkdiv1
  SIM_CLKDIV1.

Field Documentation

(1) uint8_t sim_clock_config_t::er32kSrc

(2) uint32_t sim_clock_config_t::clkdiv1

4.2.2 struct oscer_config_t

Data Fields

- uint8_t enableMode
  OSCERCLK enable mode.

Field Documentation

(1) uint8_t oscer_config_t::enableMode

OR'ed value of _oscer_enable_mode.

4.2.3 struct osc_config_t

Defines the configuration data structure to initialize the OSC. When porting to a new board, set the following members according to the board setting:

1. freq: The external frequency.
2. workMode: The OSC module mode.

Data Fields

- uint32_t freq
  External clock frequency.
- uint8_t capLoad
  Capacitor load setting.
- osc_mode_t workMode
  OSC work mode setting.
- oscer_config_t oscerConfig
  Configuration for OSCERCLK.

Field Documentation

(1) uint32_t osc_config_t::freq

(2) uint8_t osc_config_t::capLoad

(3) osc_mode_t osc_config_t::workMode

(4) oscer_config_t osc_config_t::oscerConfig

4.2.4 struct mcg_pll_config_t

Data Fields

- uint8_t enableMode
  Enable mode.
- mcg_pll_ref_src_t refSrc
  PLL reference clock source.
- uint8_t frdiv
  FLL reference clock divider.

Field Documentation

(1) uint8_t mcg_pll_config_t::enableMode

OR'ed value of enumeration _mcg_pll_enable_mode.

(2) mcg_pll_ref_src_t mcg_pll_config_t::refSrc

(3) uint8_t mcg_pll_config_t::frdiv

4.2.5 struct mcg_config_t

When porting to a new board, set the following members according to the board setting:

1. frdiv: If the FLL uses the external reference clock, set this value to ensure that the external reference clock divided by frdiv is in the 31.25 kHz to 39.0625 kHz range.
2. The PLL reference clock divider PRDIV: PLL reference clock frequency after PRDIV should be in the FSL_FEATURE_MCG_PLL_REF_MIN to FSL_FEATURE_MCG_PLL_REF_MAX range.

Data Fields

- mcg_mode_t mcgMode
  MCG mode.
- uint8_t irclkEnableMode
  MCGIRCLK enable mode.
- mcg_irc_mode_t ircs
  Source, MCG_C2[IRCS].
- uint8_t fcrdiv
  Divider, MCG_SC[FCRDIV].
- uint8_t frdiv
  Divider MCG_C1[FRDIV].
- mcg_drs_t drs
  DCO range MCG_C4[DRST_DRS].
- mcg_dmx32_t dmx32
  MCG_C4[DMX32].
- mcg_oscsel_t oscsel
  OSC select MCG_C7[OSCSEL].

- mcg_pll_config_t pll0Config
  MCGPLL0CLK configuration.

Field Documentation

(1) mcg_mode_t mcg_config_t::mcgMode

(2) uint8_t mcg_config_t::irclkEnableMode

(3) mcg_irc_mode_t mcg_config_t::ircs

(4) uint8_t mcg_config_t::fcrdiv

(5) uint8_t mcg_config_t::frdiv

(6) mcg_drs_t mcg_config_t::drs

(7) mcg_dmx32_t mcg_config_t::dmx32

(8) mcg_oscsel_t mcg_config_t::oscsel

(9) mcg_pll_config_t mcg_config_t::pll0Config

4.3 Macro Definition Documentation

4.3.1 #define MCG_CONFIG_CHECK_PARAM 0U

Some MCG settings must be changed with conditions, for example:

1. MCGIRCLK settings, such as the source, divider, and the trim value should not change when MC-GIRCLK is used as a system clock source.
2. MCG_C7[OSCSEL] should not be changed when the external reference clock is used as a system clock source. For example, in FBE/BLPE/PBE modes.
3. The users should only switch between the supported clock modes.

MCG functions check the parameter and MCG status before setting, if not allowed to change, the functions return error. The parameter checking increases code size, if code size is a critical requirement, change M-CG_CONFIG_CHECK_PARAM to 0 to disable parameter checking.

4.3.2 #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.

4.3.3 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

4.3.4 #define DMAMUX_CLOCKS

Value:

```
{                                                                    \
        kCLOCK_Dmamux0, kCLOCK_Dmamux1, kCLOCK_Dmamux2, kCLOCK_Dmamux3 \
    }
```

4.3.5 #define RTC_CLOCKS

Value:

```
{                \
        kCLOCK_Rtc0 \
    }
```

4.3.6 #define SPI_CLOCKS

Value:

```
{                             \
        kCLOCK_Spi0, kCLOCK_Spi1 \
    }
```

4.3.7 #define SLCD_CLOCKS

Value:

```
{                 \
        kCLOCK_Slcd0 \
    }
```

4.3.8 #define EWM_CLOCKS

Value:

```
{                \
        kCLOCK_Ewm0 \
    }
```

4.3.9 #define AFE_CLOCKS

Value:

```
{                \
        kCLOCK_Afe0 \
    }
```

4.3.10 #define ADC16_CLOCKS

Value:

```
{                 \
        kCLOCK_Adc0   \
    }
```

4.3.11 #define XBAR_CLOCKS

Value:

```
{                \
        kCLOCK_Xbar \
    }
```

4.3.12 #define SYSMPU_CLOCKS

Value:

```
{                   \
        kCLOCK_Sysmpu0 \
    }
```

4.3.13 #define VREF_CLOCKS

Value:

```
{                 \
        kCLOCK_Vref0 \
    }
```

4.3.14 #define DMA_CLOCKS

Value:

```
{                  \
        kCLOCK_Dma0 \
    }
```

4.3.15 #define PORT_CLOCKS

Value:

```
{                                                                                                    \
        kCLOCK_PortA, kCLOCK_PortB, kCLOCK_PortC, kCLOCK_PortD, kCLOCK_PortE, kCLOCK_PortF, kCLOCK_PortG, \
            kCLOCK_PortH, kCLOCK_PortI, kCLOCK_PortJ, kCLOCK_PortK, kCLOCK_PortL, kCLOCK_PortM            \
    }
```

4.3.16 #define UART_CLOCKS

Value:

```
{                                                             \
        kCLOCK_Uart0, kCLOCK_Uart1, kCLOCK_Uart2, kCLOCK_Uart3 \
    }
```

4.3.17 #define PIT_CLOCKS

Value:

```
{                             \
        kCLOCK_Pit0, kCLOCK_Pit1 \
    }
```

4.3.18 #define RNGA_CLOCKS

Value:

```
{                   \
        kCLOCK_Rnga0 \
    }
```

4.3.19 #define CRC_CLOCKS

Value:

```
{                \
        kCLOCK_Crc0 \
    }
```

4.3.20 #define I2C_CLOCKS

Value:

```
{                             \
        kCLOCK_I2c0, kCLOCK_I2c1 \
    }
```

4.3.21 #define LPTMR_CLOCKS

Value:

```
{                  \
        kCLOCK_Lptmr0 \
    }
```

4.3.22 #define TMR_CLOCKS

Value:

```
{                                                     \
        kCLOCK_Tmr0, kCLOCK_Tmr1, kCLOCK_Tmr2, kCLOCK_Tmr3 \
    }
```

4.3.23 #define PDB_CLOCKS

Value:

```
{                \
        kCLOCK_Pdb0 \
    }
```

4.3.24 #define FTF_CLOCKS

Value:

```
{                \
        kCLOCK_Ftf0 \
    }
```

4.3.25 #define CMP_CLOCKS

Value:

```
{                             \
        kCLOCK_Cmp0, kCLOCK_Cmp1 \
    }
```

4.3.26 #define SYS_CLK kCLOCK_CoreSysClk

4.4 Enumeration Type Documentation

4.4.1 enum clock_name_t

Enumerator

kCLOCK_CoreSysClk Core/system clock.
kCLOCK_PlatClk Platform clock.
kCLOCK_BusClk Bus clock.
kCLOCK_FlashClk Flash clock.
kCLOCK_Er32kClk External reference 32K clock (ERCLK32K)
kCLOCK_Osc0ErClk OSC0 external reference clock (OSC0ERCLK)
kCLOCK_McgFixedFreqClk MCG fixed frequency clock (MCGFFCLK)
kCLOCK_McgInternalRefClk MCG internal reference clock (MCGIRCLK)
kCLOCK_McgFllClk MCGFLLCLK.
kCLOCK_McgPll0Clk MCGPLL0CLK.
kCLOCK_McgExtPllClk EXT_PLLCLK.
kCLOCK_McgPeriphClk MCG peripheral clock (MCGPCLK)
kCLOCK_LpoClk LPO clock.

4.4.2 enum clock_ip_name_t

4.4.3 enum osc_mode_t

Enumerator

kOSC_ModeExt Use an external clock.
kOSC_ModeOscLowPower Oscillator low power.
kOSC_ModeOscHighGain Oscillator high gain.

4.4.4 enum _osc_cap_load

Enumerator

kOSC_Cap2P 2 pF capacitor load
kOSC_Cap4P 4 pF capacitor load
kOSC_Cap8P 8 pF capacitor load
kOSC_Cap16P 16 pF capacitor load

4.4.5 enum _oscer_enable_mode

Enumerator

kOSC_ErClkEnable Enable.
kOSC_ErClkEnableInStop Enable in stop mode.

4.4.6 enum mcg_fll_src_t

Enumerator

kMCG_FllSrcExternal External reference clock is selected.
kMCG_FllSrcInternal The slow internal reference clock is selected.

4.4.7 enum mcg_irc_mode_t

Enumerator

kMCG_IrcSlow Slow internal reference clock selected.
kMCG_IrcFast Fast internal reference clock selected.

4.4.8 enum mcg_dmx32_t

Enumerator

kMCG_Dmx32Default DCO has a default range of 25%.
kMCG_Dmx32Fine DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

4.4.9 enum mcg_drs_t

Enumerator

kMCG_DrsLow Low frequency range.
kMCG_DrsMid Mid frequency range.
kMCG_DrsMidHigh Mid-High frequency range.
kMCG_DrsHigh High frequency range.

4.4.10 enum mcg_pll_ref_src_t

Enumerator

kMCG_PllRefRtc Selects 32k RTC oscillator.
kMCG_PllRefIrc Selects 32k IRC.
kMCG_PllRefFllRef Selects FLL reference clock, the clock after FRDIV.

4.4.11 enum mcg_clkout_src_t

Enumerator

kMCG_ClkOutSrcOut Output of the FLL is selected (reset default)
kMCG_ClkOutSrcInternal Internal reference clock is selected.
kMCG_ClkOutSrcExternal External reference clock is selected.

4.4.12 enum mcg_atm_select_t

Enumerator

kMCG_AtmSel32k 32 kHz Internal Reference Clock selected
kMCG_AtmSel4m 4 MHz Internal Reference Clock selected

4.4.13 enum mcg_oscsel_t

Enumerator

kMCG_OscselOsc Selects System Oscillator (OSCCLK)
kMCG_OscselRtc Selects 32 kHz RTC Oscillator.

4.4.14 enum mcg_pll_clk_select_t

Enumerator

kMCG_PllClkSelPll0 PLL0 output clock is selected.

4.4.15 enum mcg_monitor_mode_t

Enumerator

kMCG_MonitorNone Clock monitor is disabled.
kMCG_MonitorInt Trigger interrupt when clock lost.
kMCG_MonitorReset System reset when clock lost.

4.4.16 anonymous enum

Enumeration _mcg_status

Enumerator

kStatus_MCG_ModeUnreachable Can't switch to target mode.
kStatus_MCG_ModeInvalid Current mode invalid for the specific function.
kStatus_MCG_AtmBusClockInvalid Invalid bus clock for ATM.
kStatus_MCG_AtmDesiredFreqInvalid Invalid desired frequency for ATM.
kStatus_MCG_AtmIrcUsed IRC is used when using ATM.
kStatus_MCG_AtmHardwareFail Hardware fail occurs during ATM.
kStatus_MCG_SourceUsed Can't change the clock source because it is in use.

4.4.17 anonymous enum

Enumeration _mcg_status_flags_t

Enumerator

kMCG_Osc0LostFlag OSC0 lost.

kMCG_Osc0InitFlag OSC0 crystal initialized.
kMCG_RtcOscLostFlag RTC OSC lost.
kMCG_Pll0LostFlag PLL0 lost.
kMCG_Pll0LockFlag PLL0 locked.

4.4.18 anonymous enum

Enumeration _mcg_irclk_enable_mode

Enumerator

kMCG_IrclkEnable MCGIRCLK enable.
kMCG_IrclkEnableInStop MCGIRCLK enable in stop mode.

4.4.19 anonymous enum

Enumeration _mcg_pll_enable_mode

Enumerator

kMCG_PllEnableIndependent MCGPLLCLK enable independent of the MCG clock mode. Generally, the PLL is disabled in FLL modes (FEI/FBI/FEE/FBE). Setting the PLL clock enable independent, enables the PLL in the FLL modes.
kMCG_PllEnableInStop MCGPLLCLK enable in STOP mode.

4.4.20 enum mcg_mode_t

Enumerator

kMCG_ModeFEI FEI - FLL Engaged Internal.
kMCG_ModeFBI FBI - FLL Bypassed Internal.
kMCG_ModeBLPI BLPI - Bypassed Low Power Internal.
kMCG_ModeFEE FEE - FLL Engaged External.
kMCG_ModeFBE FBE - FLL Bypassed External.
kMCG_ModeBLPE BLPE - Bypassed Low Power External.
kMCG_ModePBE PBE - PLL Bypassed External.
kMCG_ModePEE PEE - PLL Engaged External.
kMCG_ModePEI PEI - PLL Engaged Internal.
kMCG_ModePBI PBI - PLL Bypassed Internal.
kMCG_ModeError Unknown mode.

4.5 Function Documentation

4.5.1 static void CLOCK_EnableClock (clock_ip_name_t *name*) [inline], [static]

Parameters

name	Which clock to enable, see clock_ip_name_t.

4.5.2 static void CLOCK_DisableClock (clock_ip_name_t *name*) [inline], [static]

Parameters

name	Which clock to disable, see clock_ip_name_t.

4.5.3 static void CLOCK_SetEr32kClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set ERCLK32K clock source.

4.5.4 static void CLOCK_SetAfeClkSrc (uint32_t *src*) [inline], [static]

Parameters

src	The value to set AFECLKSEL clock source.

4.5.5 static void CLOCK_SetClkOutClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set CLKOUT source.

4.5.6 static void CLOCK_SetAdcTriggerClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set ADC trigger clock source.

4.5.7 uint32_t CLOCK_GetAfeFreq (void)

This function checks the current mode configurations in MISC_CTL register.

Returns

Clock frequency value in Hertz

4.5.8 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. The MCG must be properly configured before using this function.

Parameters

clockName	Clock names defined in clock_name_t

Returns

Clock frequency value in Hertz

4.5.9 uint32_t CLOCK_GetCoreSysClkFreq (void)

Returns

Clock frequency in Hz.

4.5.10 uint32_t CLOCK_GetPlatClkFreq (void)

Returns

Clock frequency in Hz.

4.5.11 uint32_t CLOCK_GetBusClkFreq (void)

Returns

Clock frequency in Hz.

4.5.12 uint32_t CLOCK_GetFlashClkFreq (void)

Returns

Clock frequency in Hz.

4.5.13 uint32_t CLOCK_GetEr32kClkFreq (void)

Returns

Clock frequency in Hz.

4.5.14 uint32_t CLOCK_GetOsc0ErClkFreq (void)

Returns

Clock frequency in Hz.

4.5.15 void CLOCK_SetSimConfig (sim_clock_config_t const * *config*)

This function sets system layer clock settings in SIM module.

Parameters

config	Pointer to the configure structure.

4.5.16 static void CLOCK_SetSimSafeDivs (void) [inline], [static]

The system level clocks (core clock, bus clock, flexbus clock and flash clock) must be in allowed ranges. During MCG clock mode switch, the MCG output clock changes then the system level clocks may be out of range. This function could be used before MCG mode change, to make sure system level clocks are in allowed range.

4.5.17 uint32_t CLOCK_GetOutClkFreq (void)

This function gets the MCG output clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGOUTCLK.

4.5.18 uint32_t CLOCK_GetFllFreq (void)

This function gets the MCG FLL clock frequency in Hz based on the current MCG register value. The FLL is enabled in FEI/FBI/FEE/FBE mode and disabled in low power state in other modes.

Returns

The frequency of MCGFLLCLK.

4.5.19 uint32_t CLOCK_GetInternalRefClkFreq (void)

This function gets the MCG internal reference clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGIRCLK.

4.5.20 uint32_t CLOCK_GetFixedFreqClkFreq (void)

This function gets the MCG fixed frequency clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGFFCLK.

4.5.21 uint32_t CLOCK_GetPll0Freq (void)

This function gets the MCG PLL0 clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGPLL0CLK.

4.5.22 static void CLOCK_SetLowPowerEnable (bool *enable*) [inline], [static]

Enabling the MCG low power disables the PLL and FLL in bypass modes. In other words, in FBE and PBE modes, enabling low power sets the MCG to BLPE mode. In FBI and PBI modes, enabling low power sets the MCG to BLPI mode. When disabling the MCG low power, the PLL or FLL are enabled based on MCG settings.

Parameters

enable	True to enable MCG low power, false to disable MCG low power.

4.5.23 status_t CLOCK_SetInternalRefClkConfig (uint8_t *enableMode*, mcg_irc_mode_t *ircs*, uint8_t *fcrdiv*)

This function sets the `MCGIRCLK` base on parameters. It also selects the IRC source. If the fast IRC is used, this function sets the fast IRC divider. This function also sets whether the `MCGIRCLK` is enabled in stop mode. Calling this function in FBI/PBI/BLPI modes may change the system clock. As a result, using the function in these modes it is not allowed.

Parameters

enableMode	MCGIRCLK enable mode, OR'ed value of the enumeration _mcg_irclk_enable_-mode.
ircs	MCGIRCLK clock source, choose fast or slow.
fcrdiv	Fast IRC divider setting (`FCRDIV`).

Return values

kStatus_MCG_Source-Used	Because the internal reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs.
kStatus_Success	MCGIRCLK configuration finished successfully.

4.5.24 status_t CLOCK_SetExternalRefClkConfig (mcg_oscsel_t *oscsel*)

Selects the MCG external reference clock source, changes the MCG_C7[OSCSEL], and waits for the clock source to be stable. Because the external reference clock should not be changed in FEE/FBE/BLP-E/PBE/PEE modes, do not call this function in these modes.

Parameters

oscsel	MCG external reference clock source, MCG_C7[OSCSEL].

Return values

kStatus_MCG_SourceUsed	Because the external reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs.
kStatus_Success	External reference clock set successfully.

4.5.25 static void CLOCK_SetFllExtRefDiv (uint8_t *frdiv*) `[inline]`, `[static]`

Sets the FLL external reference clock divider value, the register MCG_C1[FRDIV].

Parameters

frdiv	The FLL external reference clock divider value, MCG_C1[FRDIV].

4.5.26 void CLOCK_EnablePll0 (mcg_pll_config_t const * *config*)

This function sets us the PLL0 in FLL mode and reconfigures the PLL0. Ensure that the PLL reference clock is enabled before calling this function and that the PLL0 is not used as a clock source. The function CLOCK_CalcPllDiv gets the correct PLL divider values.

Parameters

config	Pointer to the configuration structure.

4.5.27 static void CLOCK_DisablePll0 (void) `[inline]`, `[static]`

This function disables the PLL0 in FLL mode. It should be used together with the CLOCK_EnablePll0.

4.5.28 void CLOCK_SetOsc0MonitorMode (mcg_monitor_mode_t *mode*)

This function sets the OSC0 clock monitor mode. See mcg_monitor_mode_t for details.

Parameters

mode	Monitor mode to set.

4.5.29 void CLOCK_SetRtcOscMonitorMode (mcg_monitor_mode_t *mode*)

This function sets the RTC OSC clock monitor mode. See mcg_monitor_mode_t for details.

Parameters

mode	Monitor mode to set.

4.5.30 void CLOCK_SetPll0MonitorMode (mcg_monitor_mode_t *mode*)

This function sets the PLL0 clock monitor mode. See mcg_monitor_mode_t for details.

Parameters

mode	Monitor mode to set.

4.5.31 uint32_t CLOCK_GetStatusFlags (void)

This function gets the MCG clock status flags. All status flags are returned as a logical OR of the enumeration refer to _mcg_status_flags_t. To check a specific flag, compare the return value with the flag.

Example:

```
* To check the clock lost lock status of OSC0 and PLL0.
* uint32_t mcgFlags;
*
* mcgFlags = CLOCK_GetStatusFlags();
*
* if (mcgFlags & kMCG_Osc0LostFlag)
* {
*     OSC0 clock lock lost. Do something.
* }
* if (mcgFlags & kMCG_Pll0LostFlag)
* {
*     PLL0 clock lock lost. Do something.
* }
*
```

Returns

Logical OR value of the enumeration _mcg_status_flags_t.

4.5.32 void CLOCK_ClearStatusFlags (uint32_t *mask*)

This function clears the MCG clock lock lost status. The parameter is a logical OR value of the flags to clear. See the enumeration _mcg_status_flags_t.

Example:

```
* To clear the clock lost lock status flags of OSC0 and PLL0.
*
* CLOCK_ClearStatusFlags(kMCG_Osc0LostFlag | kMCG_Pll0LostFlag);
*
```

Parameters

mask	The status flags to clear. This is a logical OR of members of the enumeration _mcg_-status_flags_t.

4.5.33 static void OSC_SetExtRefClkConfig (OSC_Type ∗ *base*, oscer_config_t const ∗ *config*) [inline], [static]

This function configures the OSC external reference clock (OSCERCLK). This is an example to enable the OSCERCLK in normal and stop modes and also set the output divider to 1:

```
oscer_config_t config =
{
    .enableMode = kOSC_ErClkEnable |
      kOSC_ErClkEnableInStop,
    .erclkDiv   = 1U,
};

OSC_SetExtRefClkConfig(OSC, &config);
```

Parameters

base	OSC peripheral address.
config	Pointer to the configuration structure.

4.5.34 static void OSC_SetCapLoad (OSC_Type ∗ *base*, uint8_t *capLoad*) [inline], [static]

This function sets the specified capacitors configuration for the oscillator. This should be done in the early system level initialization function call based on the system configuration.

Parameters

base	OSC peripheral address.
capLoad	OR'ed value for the capacitor load option, see _osc_cap_load.

Example:

```
To enable only 2 pF and 8 pF capacitor load, please use like this.
OSC_SetCapLoad(OSC, kOSC_Cap2P | kOSC_Cap8P);
```

4.5.35 void CLOCK_InitOsc0 (osc_config_t const * *config*)

This function initializes the OSC0 according to the board configuration.

Parameters

config	Pointer to the OSC0 configuration structure.

4.5.36 void CLOCK_DeinitOsc0 (void)

This function deinitializes the OSC0.

4.5.37 static void CLOCK_SetXtal0Freq (uint32_t *freq*) [inline], [static]

Parameters

freq	The XTAL0/EXTAL0 input clock frequency in Hz.

4.5.38 static void CLOCK_SetXtal32Freq (uint32_t *freq*) [inline], [static]

Parameters

freq	The XTAL32/EXTAL32/RTC_CLKIN input clock frequency in Hz.

4.5.39 void CLOCK_SetSlowIrcFreq (uint32_t *freq*)

Parameters

freq	The Slow IRC frequency input clock frequency in Hz.

4.5.40 void CLOCK_SetFastIrcFreq (uint32_t *freq*)

Parameters

freq	The Fast IRC frequency input clock frequency in Hz.

4.5.41 status_t CLOCK_TrimInternalRefClk (uint32_t *extFreq*, uint32_t *desireFreq*, uint32_t * *actualFreq*, mcg_atm_select_t *atms*)

This function trims the internal reference clock by using the external clock. If successful, it returns the kStatus_Success and the frequency after trimming is received in the parameter `actualFreq`. If an error occurs, the error code is returned.

Parameters

extFreq	External clock frequency, which should be a bus clock.
desireFreq	Frequency to trim to.
actualFreq	Actual frequency after trimming.
atms	Trim fast or slow internal reference clock.

Return values

kStatus_Success	ATM success.
kStatus_MCG_AtmBus-ClockInvalid	The bus clock is not in allowed range for the ATM.
kStatus_MCG_Atm-DesiredFreqInvalid	MCGIRCLK could not be trimmed to the desired frequency.
kStatus_MCG_AtmIrc-Used	Could not trim because MCGIRCLK is used as a bus clock source.

kStatus_MCG_AtmHardwareFail	Hardware fails while trimming.

4.5.42 mcg_mode_t CLOCK_GetMode (void)

This function checks the MCG registers and determines the current MCG mode.

Returns

Current MCG mode or error code; See mcg_mode_t.

4.5.43 status_t CLOCK_SetFeiMode (mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(∗)(void) *fllStableDelay*)

This function sets the MCG to FEI mode. If setting to FEI mode fails from the current mode, this function returns an error.

Parameters

dmx32	DMX32 in FEI mode.
drs	The DCO range selection.
fllStableDelay	Delay function to ensure that the FLL is stable. Passing NULL does not cause a delay.

Return values

kStatus_MCG_ModeUnreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

Note

If `dmx32` is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to a frequency above 32768 Hz.

4.5.44 status_t CLOCK_SetFeeMode (uint8_t *frdiv*, mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(∗)(void) *fllStableDelay*)

This function sets the MCG to FEE mode. If setting to FEE mode fails from the current mode, this function returns an error.

Parameters

frdiv	FLL reference clock divider setting, FRDIV.
dmx32	DMX32 in FEE mode.
drs	The DCO range selection.
fllStableDelay	Delay function to make sure FLL is stable. Passing NULL does not cause a delay.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.45 status_t CLOCK_SetFbiMode (mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(∗)(void) *fllStableDelay*)

This function sets the MCG to FBI mode. If setting to FBI mode fails from the current mode, this function returns an error.

Parameters

dmx32	DMX32 in FBI mode.
drs	The DCO range selection.
fllStableDelay	Delay function to make sure FLL is stable. If the FLL is not used in FBI mode, this parameter can be NULL. Passing NULL does not cause a delay.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

Note

If `dmx32` is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

4.5.46 status_t CLOCK_SetFbeMode (uint8_t *frdiv*, mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(*)(void) *fllStableDelay*)

This function sets the MCG to FBE mode. If setting to FBE mode fails from the current mode, this function returns an error.

Parameters

frdiv	FLL reference clock divider setting, FRDIV.
dmx32	DMX32 in FBE mode.
drs	The DCO range selection.
fllStableDelay	Delay function to make sure FLL is stable. If the FLL is not used in FBE mode, this parameter can be NULL. Passing NULL does not cause a delay.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.47 status_t CLOCK_SetBlpiMode (void)

This function sets the MCG to BLPI mode. If setting to BLPI mode fails from the current mode, this function returns an error.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.48 status_t CLOCK_SetBlpeMode (void)

This function sets the MCG to BLPE mode. If setting to BLPE mode fails from the current mode, this function returns an error.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.

kStatus_Success	Switched to the target mode successfully.

4.5.49 status_t CLOCK_SetPbeMode (mcg_pll_clk_select_t *pllcs*, mcg_pll_config_t const * *config*)

This function sets the MCG to PBE mode. If setting to PBE mode fails from the current mode, this function returns an error.

Parameters

pllcs	The PLL selection, PLLCS.
config	Pointer to the PLL configuration.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

Note

1. The parameter `pllcs` selects the PLL. For platforms with only one PLL, the parameter pllcs is kept for interface compatibility.
2. The parameter `config` is the PLL configuration structure. On some platforms, it is possible to choose the external PLL directly, which renders the configuration structure not necessary. In this case, pass in NULL. For example: CLOCK_SetPbeMode(kMCG_OscselOsc, kMCG_Pll-ClkSelExtPll, NULL);

4.5.50 status_t CLOCK_SetPeeMode (void)

This function sets the MCG to PEE mode.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.

kStatus_Success	Switched to the target mode successfully.

Note

This function only changes the CLKS to use the PLL/FLL output. If the PRDIV/VDIV are different than in the PBE mode, set them up in PBE mode and wait. When the clock is stable, switch to PEE mode.

4.5.51 status_t CLOCK_SetPbiMode (void)

This function sets the MCG to PBI mode.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.52 status_t CLOCK_SetPeiMode (void)

This function sets the MCG to PEI mode.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.53 status_t CLOCK_ExternalModeToFbeModeQuick (void)

This function switches the MCG from external modes (PEE/PBE/BLPE/FEE) to the FBE mode quickly. The external clock is used as the system clock source and PLL is disabled. However, the FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEE mode to FEI mode:

```
* CLOCK_ExternalModeToFbeModeQuick();
* CLOCK_SetFeiMode(...);
*
```

Return values

kStatus_Success	Switched successfully.
kStatus_MCG_Mode-Invalid	If the current mode is not an external mode, do not call this function.

4.5.54 status_t CLOCK_InternalModeToFbiModeQuick (void)

This function switches the MCG from internal modes (PEI/PBI/BLPI/FEI) to the FBI mode quickly. The MCGIRCLK is used as the system clock source and PLL is disabled. However, FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEI mode to FEE mode:

```
* CLOCK_InternalModeToFbiModeQuick();
* CLOCK_SetFeeMode(...);
*
```

Return values

kStatus_Success	Switched successfully.
kStatus_MCG_Mode-Invalid	If the current mode is not an internal mode, do not call this function.

4.5.55 status_t CLOCK_BootToFeiMode (mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(∗)(void) *fllStableDelay*)

This function sets the MCG to FEI mode from the reset mode. It can also be used to set up MCG during system boot up.

Parameters

dmx32	DMX32 in FEI mode.
drs	The DCO range selection.
fllStableDelay	Delay function to ensure that the FLL is stable.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

Note

If `dmx32` is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

4.5.56 status_t CLOCK_BootToFeeMode (mcg_oscsel_t *oscsel,* uint8_t *frdiv,* mcg_dmx32_t *dmx32,* mcg_drs_t *drs,* void(∗)(void) *fllStableDelay*)

This function sets MCG to FEE mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

oscsel	OSC clock select, OSCSEL.
frdiv	FLL reference clock divider setting, FRDIV.
dmx32	DMX32 in FEE mode.
drs	The DCO range selection.
fllStableDelay	Delay function to ensure that the FLL is stable.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.57 status_t CLOCK_BootToBlpiMode (uint8_t *fcrdiv,* mcg_irc_mode_t *ircs,* uint8_t *ircEnableMode*)

This function sets the MCG to BLPI mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

fcrdiv	Fast IRC divider, FCRDIV.
ircs	The internal reference clock to select, IRCS.
ircEnableMode	The MCGIRCLK enable mode, OR'ed value of the enumeration _mcg_irclk_enable_mode.

Return values

kStatus_MCG_SourceUsed	Could not change MCGIRCLK setting.
kStatus_Success	Switched to the target mode successfully.

4.5.58 status_t CLOCK_BootToBlpeMode (mcg_oscsel_t *oscsel*)

This function sets the MCG to BLPE mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

oscsel	OSC clock select, MCG_C7[OSCSEL].

Return values

kStatus_MCG_ModeUnreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.59 status_t CLOCK_BootToPeeMode (mcg_oscsel_t *oscsel*, mcg_pll_clk_select_t *pllcs*, mcg_pll_config_t const * *config*)

This function sets the MCG to PEE mode from reset mode. It can also be used to set up the MCG during system boot up.

Parameters

oscsel	OSC clock select, MCG_C7[OSCSEL].
pllcs	The PLL selection, PLLCS.
config	Pointer to the PLL configuration.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.60 status_t CLOCK_BootToPeiMode (void)

This function sets the MCG to PEI mode from the reset mode. It can be used to set up the MCG during system boot up.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

4.5.61 status_t CLOCK_SetMcgConfig (mcg_config_t const * *config*)

This function sets MCG to a target mode defined by the configuration structure. If switching to the target mode fails, this function chooses the correct path.

Parameters

config	Pointer to the target MCG mode configuration structure.

Returns

Return kStatus_Success if switched successfully; Otherwise, it returns an error code _mcg_status.

Note

If the external clock is used in the target mode, ensure that it is enabled. For example, if the OSC0 is used, set up OSC0 correctly before calling this function.

4.6 Variable Documentation

4.6.1 volatile uint32_t g_xtal0Freq

The XTAL0/EXTAL0 (OSC0) 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:

```
* Set up the OSC0
* CLOCK_InitOsc0(...);
* Set the XTAL0 value to the clock driver.
* CLOCK_SetXtal0Freq(80000000);
*
```

This is important for the multicore platforms where only one core needs to set up the OSC0 using the CLOCK_InitOsc0. All other cores need to call the CLOCK_SetXtal0Freq to get a valid clock frequency.

4.6.2 volatile uint32_t g_xtal32Freq

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

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

4.7 Multipurpose Clock Generator (MCG)

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

4.7.1 Function description

MCG driver provides these functions:

- Functions to get the MCG clock frequency.
- Functions to configure the MCG clock, such as PLLCLK and MCGIRCLK.
- Functions for the MCG clock lock lost monitor.
- Functions for the OSC configuration.
- Functions for the MCG auto-trim machine.
- Functions for the MCG mode.

4.7.1.1 MCG frequency functions

MCG module provides clocks, such as MCGOUTCLK, MCGIRCLK, MCGFFCLK, MCGFLLCLK, and MCGPLLCLK. The MCG driver provides functions to get the frequency of these clocks, such as CLOCK_GetOutClkFreq(), CLOCK_GetInternalRefClkFreq(), CLOCK_GetFixedFreqClkFreq(), CLOCK_GetFllFreq(), CLOCK_GetPll0Freq(), CLOCK_GetPll1Freq(), and CLOCK_GetExtPllFreq(). These functions get the clock frequency based on the current MCG registers.

4.7.1.2 MCG clock configuration

The MCG driver provides functions to configure the internal reference clock (MCGIRCLK), the external reference clock, and MCGPLLCLK.

The function CLOCK_SetInternalRefClkConfig() configures the MCGIRCLK, including the source and the driver. Do not change MCGIRCLK when the MCG mode is BLPI/FBI/PBI because the MCGIRCLK is used as a system clock in these modes and changing settings makes the system clock unstable.

The function CLOCK_SetExternalRefClkConfig() configures the external reference clock source (MCG_C7[OSCSEL]). Do not call this function when the MCG mode is BLPE/FBE/PBE/FEE/PEE because the external reference clock is used as a clock source in these modes. Changing the external reference clock source requires at least a 50 microseconds wait. The function CLOCK_SetExternalRefClkConfig() implements a for loop delay internally. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 50 micro seconds delay. However, when the system clock is slow, the delay time may significantly increase. This for loop count can be optimized for better performance for specific cases.

The MCGPLLCLK is disabled in FBE/FEE/FBI/FEI modes by default. Applications can enable the MCGPLLCLK in these modes using the functions CLOCK_EnablePll0() and CLOCK_EnablePll1(). To enable the MCGPLLCLK, the PLL reference clock divider(PRDIV) and the PLL VCO divider(VDIV) must be set to a proper value. The function CLOCK_CalcPllDiv() helps to get the PRDIV/VDIV.

4.7.1.3 MCG clock lock monitor functions

The MCG module monitors the OSC and the PLL clock lock status. The MCG driver provides the functions to set the clock monitor mode, check the clock lost status, and clear the clock lost status.

4.7.1.4 OSC configuration

The MCG is needed together with the OSC module to enable the OSC clock. The function CLOCK_InitOsc0() CLOCK_InitOsc1 uses the MCG and OSC to initialize the OSC. The OSC should be configured based on the board design.

4.7.1.5 MCG auto-trim machine

The MCG provides an auto-trim machine to trim the MCG internal reference clock based on the external reference clock (BUS clock). During clock trimming, the MCG must not work in FEI/FBI/BLPI/PBI/PEI modes. The function CLOCK_TrimInternalRefClk() is used for the auto clock trimming.

4.7.1.6 MCG mode functions

The function CLOCK_GetMcgMode returns the current MCG mode. The MCG can only switch between the neighbouring modes. If the target mode is not current mode's neighbouring mode, the application must choose the proper switch path. For example, to switch to PEE mode from FEI mode, use FEI -> FBE -> PBE -> PEE.

For the MCG modes, the MCG driver provides three kinds of functions:

The first type of functions involve functions CLOCK_SetXxxMode, such as CLOCK_SetFeiMode(). These functions only set the MCG mode from neighbouring modes. If switching to the target mode directly from current mode is not possible, the functions return an error.

The second type of functions are the functions CLOCK_BootToXxxMode, such as CLOCK_BootToFeiMode(). These functions set the MCG to specific modes from reset mode. Because the source mode and target mode are specific, these functions choose the best switch path. The functions are also useful to set up the system clock during boot up.

The third type of functions is the CLOCK_SetMcgConfig(). This function chooses the right path to switch to the target mode. It is easy to use, but introduces a large code size.

Whenever the FLL settings change, there should be a 1 millisecond delay to ensure that the FLL is stable. The function CLOCK_SetMcgConfig() implements a for loop delay internally to ensure that the FLL is stable. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 1 millisecond delay. However, when the system clock is slow, the delay time may increase significantly. The for loop count can be optimized for better performance according to a specific use case.

4.7.2 Typical use case

The function CLOCK_SetMcgConfig is used to switch between any modes. However, this heavy-light function introduces a large code size. This section shows how to use the mode function to implement a quick and light-weight switch between typical specific modes. Note that the step to enable the external clock is not included in the following steps. Enable the corresponding clock before using it as a clock source.

4.7.2.1 Switch between BLPI and FEI

Use case	Steps	Functions
BLPI -> FEI	BLPI -> FBI	CLOCK_InternalModeToFbiModeQuick(...)
	FBI -> FEI	CLOCK_SetFeiMode(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
FEI -> BLPI	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
	FEI -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	FBI -> BLPI	CLOCK_SetLowPowerEnable(true)

4.7.2.2 Switch between BLPI and FEE

Use case	Steps	Functions
BLPI -> FEE	BLPI -> FBI	CLOCK_InternalModeToFbiModeQuick(...)
	Change external clock source if need	CLOCK_SetExternalRefClkConfig(...)
	FBI -> FEE	CLOCK_SetFeeMode(...)
FEE -> BLPI	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
	FEE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	FBI -> BLPI	CLOCK_SetLowPowerEnable(true)

4.7.2.3 Switch between BLPI and PEE

Use case	Steps	Functions
BLPI -> PEE	BLPI -> FBI	CLOCK_InternalModeToFbiModeQuick(...)
	Change external clock source if need	CLOCK_SetExternalRefClkConfig(...)
	FBI -> FBE	CLOCK_SetFbeMode(...) // fllStableDelay=NULL
	FBE -> PBE	CLOCK_SetPbeMode(...)
	PBE -> PEE	CLOCK_SetPeeMode(...)
PEE -> BLPI	PEE -> FBE	CLOCK_ExternalModeToFbeModeQuick(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	FBI -> BLPI	CLOCK_SetLowPowerEnable(true)

4.7.2.4 Switch between BLPE and PEE

This table applies when using the same external clock source (MCG_C7[OSCSEL]) in BLPE mode and PEE mode.

Use case	Steps	Functions
BLPE -> PEE	BLPE -> PBE	CLOCK_SetPbeMode(...)
	PBE -> PEE	CLOCK_SetPeeMode(...)
PEE -> BLPE	PEE -> FBE	CLOCK_ExternalModeToFbeModeQuick(...)
	FBE -> BLPE	CLOCK_SetLowPowerEnable(true)

If using different external clock sources (MCG_C7[OSCSEL]) in BLPE mode and PEE mode, call the CLOCK_SetExternalRefClkConfig() in FBI or FEI mode to change the external reference clock.

Use case	Steps	Functions
BLPE -> PEE	BLPE -> FBE	CLOCK_ExternalModeToFbeModeQuick(...)

	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	FBI -> FBE	CLOCK_SetFbeMode(...) with fllStableDelay=NULL
	FBE -> PBE	CLOCK_SetPbeMode(...)
	PBE -> PEE	CLOCK_SetPeeMode(...)
PEE -> BLPE	PEE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)
	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	PBI -> FBE	CLOCK_SetFbeMode(...) with fllStableDelay=NULL
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

4.7.2.5 Switch between BLPE and FEE

This table applies when using the same external clock source (MCG_C7[OSCSEL]) in BLPE mode and FEE mode.

Use case	Steps	Functions
BLPE -> FEE	BLPE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)
	FBE -> FEE	CLOCK_SetFeeMode(...)
FEE -> BLPE	PEE -> FBE	CLOCK_SetPbeMode(...)
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

If using different external clock sources (MCG_C7[OSCSEL]) in BLPE mode and FEE mode, call the CLOCK_SetExternalRefClkConfig() in FBI or FEI mode to change the external reference clock.

Use case	Steps	Functions
BLPE -> FEE	BLPE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)

	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	FBI -> FEE	CLOCK_SetFeeMode(...)
FEE -> BLPE	FEE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	PBI -> FBE	CLOCK_SetFbeMode(...) with fllStableDelay=NULL
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

4.7.2.6 Switch between BLPI and PEI

Use case	Steps	Functions
BLPI -> PEI	BLPI -> PBI	CLOCK_SetPbiMode(...)
	PBI -> PEI	CLOCK_SetPeiMode(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClk-Config(...)
PEI -> BLPI	Configure MCGIRCLK if need	CLOCK_SetInternalRefClk-Config
	PEI -> FBI	CLOCK_InternalModeToFbi-ModeQuick(...)
	FBI -> BLPI	CLOCK_SetLowPower-Enable(true)

4.7.3 Code Configuration Option

4.7.3.1 MCG_USER_CONFIG_FLL_STABLE_DELAY_EN

When switching to use FLL with function CLOCK_SetFeiMode() and CLOCK_SetFeeMode(), there is an internal function CLOCK_FllStableDelay(). It is used to delay a few ms so that to wait the FLL to be stable enough. By default, it is implemented in driver code like the following:

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/mcg Once user is willing to create their own delay funcion, just assert the macro MCG_USER_CONFIG_FLL_STABLE_DELAY_EN, and then define function CLOCK_FllStableDelay in the application code.

Chapter 5
ADC16: 16-bit SAR Analog-to-Digital Converter Driver

5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 16-bit SAR Analog-to-Digital Converter (A-DC16) module of MCUXpresso SDK devices.

5.2 Typical use case

5.2.1 Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/adc16

5.2.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/adc16

Data Structures

- struct adc16_config_t
  ADC16 converter configuration. More...
- struct adc16_hardware_compare_config_t
  ADC16 Hardware comparison configuration. More...
- struct adc16_channel_config_t
  ADC16 channel conversion configuration. More...

Enumerations

- enum _adc16_channel_status_flags { kADC16_ChannelConversionDoneFlag = ADC_SC1_COCO_MASK }
  Channel status flags.
- enum _adc16_status_flags {
  kADC16_ActiveFlag = ADC_SC2_ADACT_MASK,
  kADC16_CalibrationFailedFlag = ADC_SC3_CALF_MASK }
  Converter status flags.
- enum adc16_clock_divider_t {
  kADC16_ClockDivider1 = 0U,
  kADC16_ClockDivider2 = 1U,
  kADC16_ClockDivider4 = 2U,
  kADC16_ClockDivider8 = 3U }
  Clock divider for the converter.

- enum adc16_resolution_t {
  kADC16_Resolution8or9Bit = 0U,
  kADC16_Resolution12or13Bit = 1U,
  kADC16_Resolution10or11Bit = 2U,
  kADC16_ResolutionSE8Bit = kADC16_Resolution8or9Bit,
  kADC16_ResolutionSE12Bit = kADC16_Resolution12or13Bit,
  kADC16_ResolutionSE10Bit = kADC16_Resolution10or11Bit,
  kADC16_Resolution16Bit = 3U,
  kADC16_ResolutionSE16Bit = kADC16_Resolution16Bit }
  Converter's resolution.
- enum adc16_clock_source_t {
  kADC16_ClockSourceAlt0 = 0U,
  kADC16_ClockSourceAlt1 = 1U,
  kADC16_ClockSourceAlt2 = 2U,
  kADC16_ClockSourceAlt3 = 3U,
  kADC16_ClockSourceAsynchronousClock = kADC16_ClockSourceAlt3 }
  Clock source.
- enum adc16_long_sample_mode_t {
  kADC16_LongSampleCycle24 = 0U,
  kADC16_LongSampleCycle16 = 1U,
  kADC16_LongSampleCycle10 = 2U,
  kADC16_LongSampleCycle6 = 3U,
  kADC16_LongSampleDisabled = 4U }
  Long sample mode.
- enum adc16_reference_voltage_source_t {
  kADC16_ReferenceVoltageSourceVref = 0U,
  kADC16_ReferenceVoltageSourceValt = 1U }
  Reference voltage source.
- enum adc16_hardware_average_mode_t {
  kADC16_HardwareAverageCount4 = 0U,
  kADC16_HardwareAverageCount8 = 1U,
  kADC16_HardwareAverageCount16 = 2U,
  kADC16_HardwareAverageCount32 = 3U,
  kADC16_HardwareAverageDisabled = 4U }
  Hardware average mode.
- enum adc16_hardware_compare_mode_t {
  kADC16_HardwareCompareMode0 = 0U,
  kADC16_HardwareCompareMode1 = 1U,
  kADC16_HardwareCompareMode2 = 2U,
  kADC16_HardwareCompareMode3 = 3U }
  Hardware compare mode.

Driver version

- #define FSL_ADC16_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  ADC16 driver version 2.3.0.

Initialization

- void ADC16_Init (ADC_Type *base, const adc16_config_t *config)
  Initializes the ADC16 module.
- void ADC16_Deinit (ADC_Type *base)
  De-initializes the ADC16 module.
- void ADC16_GetDefaultConfig (adc16_config_t *config)
  Gets an available pre-defined settings for the converter's configuration.
- status_t ADC16_DoAutoCalibration (ADC_Type *base)
  Automates the hardware calibration.
- static void ADC16_SetOffsetValue (ADC_Type *base, int16_t value)
  Sets the offset value for the conversion result.

Advanced Features

- static void ADC16_EnableDMA (ADC_Type *base, bool enable)
  Enables generating the DMA trigger when the conversion is complete.
- static void ADC16_EnableHardwareTrigger (ADC_Type *base, bool enable)
  Enables the hardware trigger mode.
- void ADC16_SetHardwareCompareConfig (ADC_Type *base, const adc16_hardware_compare_-config_t *config)
  Configures the hardware compare mode.
- void ADC16_SetHardwareAverage (ADC_Type *base, adc16_hardware_average_mode_t mode)
  Sets the hardware average mode.
- uint32_t ADC16_GetStatusFlags (ADC_Type *base)
  Gets the status flags of the converter.
- void ADC16_ClearStatusFlags (ADC_Type *base, uint32_t mask)
  Clears the status flags of the converter.
- static void ADC16_EnableAsynchronousClockOutput (ADC_Type *base, bool enable)
  Enable/disable ADC Asynchronous clock output to other modules.

Conversion Channel

- void ADC16_SetChannelConfig (ADC_Type *base, uint32_t channelGroup, const adc16_channel-_config_t *config)
  Configures the conversion channel.
- static uint32_t ADC16_GetChannelConversionValue (ADC_Type *base, uint32_t channelGroup)
  Gets the conversion value.
- uint32_t ADC16_GetChannelStatusFlags (ADC_Type *base, uint32_t channelGroup)
  Gets the status flags of channel.

5.3 Data Structure Documentation

5.3.1 struct adc16_config_t

Data Fields

- adc16_reference_voltage_source_t referenceVoltageSource
  Select the reference voltage source.
- adc16_clock_source_t clockSource

Select the input clock source to converter.
- bool enableAsynchronousClock
  Enable the asynchronous clock output.
- adc16_clock_divider_t clockDivider
  Select the divider of input clock source.
- adc16_resolution_t resolution
  Select the sample resolution mode.
- adc16_long_sample_mode_t longSampleMode
  Select the long sample mode.
- bool enableHighSpeed
  Enable the high-speed mode.
- bool enableLowPower
  Enable low power.
- bool enableContinuousConversion
  Enable continuous conversion mode.
- adc16_hardware_average_mode_t hardwareAverageMode
  Set hardware average mode.

Field Documentation

(1) adc16_reference_voltage_source_t adc16_config_t::referenceVoltageSource

(2) adc16_clock_source_t adc16_config_t::clockSource

(3) bool adc16_config_t::enableAsynchronousClock

(4) adc16_clock_divider_t adc16_config_t::clockDivider

(5) adc16_resolution_t adc16_config_t::resolution

(6) adc16_long_sample_mode_t adc16_config_t::longSampleMode

(7) bool adc16_config_t::enableHighSpeed

(8) bool adc16_config_t::enableLowPower

(9) bool adc16_config_t::enableContinuousConversion

(10) adc16_hardware_average_mode_t adc16_config_t::hardwareAverageMode

5.3.2 struct adc16_hardware_compare_config_t

Data Fields

- adc16_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) adc16_hardware_compare_mode_t adc16_hardware_compare_config_t::hardwareCompareMode

See "adc16_hardware_compare_mode_t".

(2) int16_t adc16_hardware_compare_config_t::value1

(3) int16_t adc16_hardware_compare_config_t::value2

5.3.3 struct adc16_channel_config_t

Data Fields

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

Field Documentation

(1) uint32_t adc16_channel_config_t::channelNumber

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

(2) bool adc16_channel_config_t::enableInterruptOnConversionCompleted

5.4 Macro Definition Documentation

5.4.1 #define FSL_ADC16_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

5.5 Enumeration Type Documentation

5.5.1 enum _adc16_channel_status_flags

Enumerator

kADC16_ChannelConversionDoneFlag Conversion done.

5.5.2 enum _adc16_status_flags

Enumerator

kADC16_ActiveFlag Converter is active.
kADC16_CalibrationFailedFlag Calibration is failed.

5.5.3 enum adc16_clock_divider_t

Enumerator

kADC16_ClockDivider1 For divider 1 from the input clock to the module.
kADC16_ClockDivider2 For divider 2 from the input clock to the module.
kADC16_ClockDivider4 For divider 4 from the input clock to the module.
kADC16_ClockDivider8 For divider 8 from the input clock to the module.

5.5.4 enum adc16_resolution_t

Enumerator

kADC16_Resolution8or9Bit Single End 8-bit or Differential Sample 9-bit.
kADC16_Resolution12or13Bit Single End 12-bit or Differential Sample 13-bit.
kADC16_Resolution10or11Bit Single End 10-bit or Differential Sample 11-bit.
kADC16_ResolutionSE8Bit Single End 8-bit.
kADC16_ResolutionSE12Bit Single End 12-bit.
kADC16_ResolutionSE10Bit Single End 10-bit.
kADC16_Resolution16Bit Single End 16-bit or Differential Sample 16-bit.
kADC16_ResolutionSE16Bit Single End 16-bit.

5.5.5 enum adc16_clock_source_t

Enumerator

kADC16_ClockSourceAlt0 Selection 0 of the clock source.
kADC16_ClockSourceAlt1 Selection 1 of the clock source.
kADC16_ClockSourceAlt2 Selection 2 of the clock source.
kADC16_ClockSourceAlt3 Selection 3 of the clock source.
kADC16_ClockSourceAsynchronousClock Using internal asynchronous clock.

5.5.6 enum adc16_long_sample_mode_t

Enumerator

kADC16_LongSampleCycle24 20 extra ADCK cycles, 24 ADCK cycles total.
kADC16_LongSampleCycle16 12 extra ADCK cycles, 16 ADCK cycles total.
kADC16_LongSampleCycle10 6 extra ADCK cycles, 10 ADCK cycles total.
kADC16_LongSampleCycle6 2 extra ADCK cycles, 6 ADCK cycles total.
kADC16_LongSampleDisabled Disable the long sample feature.

5.5.7 enum adc16_reference_voltage_source_t

Enumerator

kADC16_ReferenceVoltageSourceVref For external pins pair of VrefH and VrefL.
kADC16_ReferenceVoltageSourceValt For alternate reference pair of ValtH and ValtL.

5.5.8 enum adc16_hardware_average_mode_t

Enumerator

kADC16_HardwareAverageCount4 For hardware average with 4 samples.
kADC16_HardwareAverageCount8 For hardware average with 8 samples.
kADC16_HardwareAverageCount16 For hardware average with 16 samples.
kADC16_HardwareAverageCount32 For hardware average with 32 samples.
kADC16_HardwareAverageDisabled Disable the hardware average feature.

5.5.9 enum adc16_hardware_compare_mode_t

Enumerator

kADC16_HardwareCompareMode0 x < value1.
kADC16_HardwareCompareMode1 x > value1.
kADC16_HardwareCompareMode2 if value1 <= value2, then x < value1 || x > value2; else, value1 > x > value2.
kADC16_HardwareCompareMode3 if value1 <= value2, then value1 <= x <= value2; else x >= value1 || x <= value2.

5.6 Function Documentation

5.6.1 void ADC16_Init (ADC_Type * *base*, const adc16_config_t * *config*)

Parameters

base	ADC16 peripheral base address.
config	Pointer to configuration structure. See "adc16_config_t".

5.6.2 void ADC16_Deinit (ADC_Type * *base*)

Parameters

base	ADC16 peripheral base address.

5.6.3 void ADC16_GetDefaultConfig (adc16_config_t * *config*)

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

```
*   config->referenceVoltageSource     = kADC16_ReferenceVoltageSourceVref
    ;
*   config->clockSource                = kADC16_ClockSourceAsynchronousClock
    ;
*   config->enableAsynchronousClock    = false;
*   config->clockDivider               = kADC16_ClockDivider8;
*   config->resolution                 = kADC16_ResolutionSE12Bit;
*   config->longSampleMode             = kADC16_LongSampleDisabled;
*   config->enableHighSpeed            = false;
*   config->enableLowPower             = false;
*   config->enableContinuousConversion = false;
*
```

Parameters

config	Pointer to the configuration structure.

5.6.4 status_t ADC16_DoAutoCalibration (ADC_Type * *base*)

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

Parameters

base	ADC16 peripheral base address.

Returns

Execution status.

Return values

kStatus_Success	Calibration is done successfully.
kStatus_Fail	Calibration has failed.

5.6.5 static void ADC16_SetOffsetValue (ADC_Type ∗ *base*, int16_t *value*) [inline], [static]

This offset value takes effect on the conversion result. If the offset value is not zero, the reading result is subtracted by it. Note, the hardware calibration fills the offset value automatically.

Parameters

base	ADC16 peripheral base address.
value	Setting offset value.

5.6.6 static void ADC16_EnableDMA (ADC_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	ADC16 peripheral base address.
enable	Switcher of the DMA feature. "true" means enabled, "false" means not enabled.

5.6.7 static void ADC16_EnableHardwareTrigger (ADC_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	ADC16 peripheral base address.
enable	Switcher of the hardware trigger feature. "true" means enabled, "false" means not enabled.

5.6.8 void ADC16_SetHardwareCompareConfig (ADC_Type ∗ *base*, const adc16_hardware_compare_config_t ∗ *config*)

The hardware compare mode provides a way to process the conversion result automatically by using hardware. Only the result in the compare range is available. To compare the range, see "adc16_hardware-

_compare_mode_t" or the appopriate reference manual for more information.

Parameters

base	ADC16 peripheral base address.
config	Pointer to the "adc16_hardware_compare_config_t" structure. Passing "NULL" disables the feature.

5.6.9 void ADC16_SetHardwareAverage (ADC_Type ∗ *base*, adc16_hardware_average_mode_t *mode*)

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

Parameters

base	ADC16 peripheral base address.
mode	Setting the hardware average mode. See "adc16_hardware_average_mode_t".

5.6.10 uint32_t ADC16_GetStatusFlags (ADC_Type ∗ *base*)

Parameters

base	ADC16 peripheral base address.

Returns

Flags' mask if indicated flags are asserted. See "_adc16_status_flags".

5.6.11 void ADC16_ClearStatusFlags (ADC_Type ∗ *base*, uint32_t *mask*)

Parameters

base	ADC16 peripheral base address.
mask	Mask value for the cleared flags. See "_adc16_status_flags".

5.6.12 static void ADC16_EnableAsynchronousClockOutput (ADC_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	ADC16 peripheral base address.
enable	Used to enable/disable ADC ADACK output. • **true** Asynchronous clock and clock output is enabled regardless of the state of the ADC. • **false** Asynchronous clock output disabled, asynchronous clock is enabled only if it is selected as input clock and a conversion is active.

5.6.13 void ADC16_SetChannelConfig (ADC_Type ∗ *base,* uint32_t *channelGroup,* const adc16_channel_config_t ∗ *config*)

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

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC has more than one group of status and control registers, one for each conversion. The channel group parameter indicates which group of registers are used, for example, channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any point, only one of the channel groups is actively controlling ADC conversions. The channel group 0 is used for both software and hardware trigger modes. Channel group 1 and greater indicates multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the appropriate MCU reference manual for the number of SC1n registers (channel groups) specific to this device. Channel group 1 or greater are not used for software trigger operation. Therefore, writing to these channel groups does not initiate a new conversion. Updating the channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

Parameters

base	ADC16 peripheral base address.
channelGroup	Channel group index.
config	Pointer to the "adc16_channel_config_t" structure for the conversion channel.

5.6.14 static uint32_t ADC16_GetChannelConversionValue (ADC_Type ∗ *base,* uint32_t *channelGroup*) [inline], [static]

Parameters

base	ADC16 peripheral base address.
channelGroup	Channel group index.

Returns

Conversion value.

5.6.15 uint32_t ADC16_GetChannelStatusFlags (ADC_Type * *base*, uint32_t *channelGroup*)

Parameters

base	ADC16 peripheral base address.
channelGroup	Channel group index.

Returns

Flags' mask if indicated flags are asserted. See "_adc16_channel_status_flags".

Chapter 6
AFE: Analog Front End Driver

6.1 Overview

The MCUXpresso SDK provides a driver for the Analog Front End (AFE) module of MCUXpresso SDK devices.

The Analog Front End or AFE is an integrated module that is comprised of ADCs, PGA, filtering, and phase compensation blocks. The AFE is responsible for measuring the phase voltage, phase current, and neutral current.

6.2 Function groups

6.2.1 Channel configuration structures

The driver uses instances of the channel configuration structures to configuration and initialization AF-E channel. This structure holds the settings of the AFE measurement channel. The settings include AFE hardware/software triggering, AFE continuous/Single conversion mode, AFE channel mode, AFE channel analog gain, AFE channel oversampling ration. The AFE channel mode selects whether the bypass mode is enabled or disabled and the external clock selection.

6.2.2 User configuration structures

The AFE driver uses instances of the user configuration structure afe_config_t for the AFE driver configuration. This structure holds the configuration which is common for all AFE channels. The settings include AFE low-power mode, AFE result format, AFE clock divider mode, AFE clock source mode, and AFE start up delay of modulators.

6.2.3 AFE Initialization

To initialize the AFE driver for a typical use case, call the AFE_GetDefaultConfig() function which populates the structure. Then, call the AFE_Init() function and pass the base address of the AFE peripheral and a pointer to the user configuration structure.

To configure the AFE channel, for a typical use case call the AFE_GetDefaultChnConfig() function which populates the structure. Then, call the AFE_SetChnConfig() function and pass the base address of the AFE peripheral and a pointer to the channel configuration structure.

6.2.4 AFE Conversion

The driver contains functions for software triggering, a channel delay after trigger setting, a result (raw or converted to right justified), reading, and waiting functions.

If the software triggering is enabled (hwTriggerEnable parameter in afe_chn_config_t is a false value), call the AFE_SoftTriggerConv() function to start conversion.

6.3 Typical use case

6.3.1 AFE Initialization

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

6.3.2 AFE Conversion

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

Data Structures

- struct afe_channel_config_t
  Defines the structure to initialize the AFE channel. More...
- struct afe_config_t
  Defines the structure to initialize the AFE module. More...

Enumerations

- enum _afe_channel_status_flag {
  kAFE_Channel0OverflowFlag = AFE_SR_OVR0_MASK,
  kAFE_Channel1OverflowFlag = AFE_SR_OVR1_MASK,
  kAFE_Channel2OverflowFlag = AFE_SR_OVR2_MASK,
  kAFE_Channel0ReadyFlag = AFE_SR_RDY0_MASK,
  kAFE_Channel1ReadyFlag = AFE_SR_RDY1_MASK,
  kAFE_Channel2ReadyFlag = AFE_SR_RDY2_MASK,
  kAFE_Channel0ConversionCompleteFlag = AFE_SR_COC0_MASK,
  kAFE_Channel1ConversionCompleteFlag = AFE_SR_COC1_MASK,
  kAFE_Channel2ConversionCompleteFlag = AFE_SR_COC2_MASK,
  kAFE_Channel3OverflowFlag = AFE_SR_OVR3_MASK,
  kAFE_Channel3ReadyFlag = AFE_SR_RDY3_MASK,
  kAFE_Channel3ConversionCompleteFlag = AFE_SR_COC3_MASK }
  Defines the type of status flags.
- enum {
  kAFE_Channel0InterruptEnable = AFE_DI_INTEN0_MASK,
  kAFE_Channel1InterruptEnable = AFE_DI_INTEN1_MASK,
  kAFE_Channel2InterruptEnable = AFE_DI_INTEN2_MASK,
  kAFE_Channel3InterruptEnable = AFE_DI_INTEN3_MASK }

Defines AFE interrupt enable.
- enum {
kAFE_Channel0DMAEnable = AFE_DI_DMAEN0_MASK,
kAFE_Channel1DMAEnable = AFE_DI_DMAEN1_MASK,
kAFE_Channel2DMAEnable = AFE_DI_DMAEN2_MASK,
kAFE_Channel3DMAEnable = AFE_DI_DMAEN3_MASK }
Defines AFE DMA enable.
- enum {
kAFE_Channel0Trigger = AFE_CR_SOFT_TRG0_MASK,
kAFE_Channel1Trigger = AFE_CR_SOFT_TRG1_MASK,
kAFE_Channel2Trigger = AFE_CR_SOFT_TRG2_MASK,
kAFE_Channel3Trigger = AFE_CR_SOFT_TRG3_MASK }
Defines AFE channel trigger flag.
- enum afe_decimator_oversample_ratio_t {
kAFE_DecimatorOversampleRatio64 = 0U,
kAFE_DecimatorOversampleRatio128 = 1U,
kAFE_DecimatorOversampleRatio256 = 2U,
kAFE_DecimatorOversampleRatio512 = 3U,
kAFE_DecimatorOversampleRatio1024 = 4U,
kAFE_DecimatorOversampleRatio2048 = 5U }
AFE OSR modes.
- enum afe_result_format_t {
kAFE_ResultFormatLeft = 0U,
kAFE_ResultFormatRight = 1U }
Defines the AFE result format modes.
- enum afe_clock_divider_t {
kAFE_ClockDivider1 = 0U,
kAFE_ClockDivider2 = 1U,
kAFE_ClockDivider4 = 2U,
kAFE_ClockDivider8 = 3U,
kAFE_ClockDivider16 = 4U,
kAFE_ClockDivider32 = 5U,
kAFE_ClockDivider64 = 6U,
kAFE_ClockDivider128 = 7U,
kAFE_ClockDivider256 = 8U }
Defines the AFE clock divider modes.
- enum afe_clock_source_t {
kAFE_ClockSource0 = 0U,
kAFE_ClockSource1 = 1U,
kAFE_ClockSource2 = 2U,
kAFE_ClockSource3 = 3U }
Defines the AFE clock source modes.
- enum afe_pga_gain_t {

kAFE_PgaDisable = 0U,
kAFE_PgaGain1 = 1U,
kAFE_PgaGain2 = 2U,
kAFE_PgaGain4 = 3U,
kAFE_PgaGain8 = 4U,
kAFE_PgaGain16 = 5U,
kAFE_PgaGain32 = 6U }
Defines the PGA's values.
- enum afe_bypass_mode_t {
kAFE_BypassInternalClockPositiveEdge = 0U,
kAFE_BypassExternalClockPositiveEdge = 1U,
kAFE_BypassInternalClockNegativeEdge = 2U,
kAFE_BypassExternalClockNegativeEdge = 3U,
kAFE_BypassDisable = 4U }
Defines the bypass modes.

Driver version

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

AFE Initialization

- void AFE_Init (AFE_Type ∗base, const afe_config_t ∗config)
Initialization for the AFE module.
- void AFE_Deinit (AFE_Type ∗base)
De-Initialization for the AFE module.
- void AFE_GetDefaultConfig (afe_config_t ∗config)
Fills the user configure structure.
- static void AFE_SoftwareReset (AFE_Type ∗base, bool enable)
Software reset the AFE module.
- static void AFE_Enable (AFE_Type ∗base, bool enable)
Enables all configured AFE channels.

AFE Conversion

- void AFE_SetChannelConfig (AFE_Type ∗base, uint32_t channel, const afe_channel_config_t ∗config)
Configure the selected AFE channel.
- void AFE_GetDefaultChannelConfig (afe_channel_config_t ∗config)
Fills the channel configuration structure.
- uint32_t AFE_GetChannelConversionValue (AFE_Type ∗base, uint32_t channel)
Reads the raw conversion value.
- static void AFE_DoSoftwareTriggerChannel (AFE_Type ∗base, uint32_t mask)
Triggers the AFE conversion by software.
- static uint32_t AFE_GetChannelStatusFlags (AFE_Type ∗base)
Gets the AFE status flag state.
- void AFE_SetChannelPhaseDelayValue (AFE_Type ∗base, uint32_t channel, uint32_t value)
Sets phase delays value.

- static void AFE_SetChannelPhasetDelayOk (AFE_Type *base)
  Asserts the phase delay setting.
- static void AFE_EnableChannelInterrupts (AFE_Type *base, uint32_t mask)
  Enables AFE interrupt.
- static void AFE_DisableChannelInterrupts (AFE_Type *base, uint32_t mask)
  Disables AFE interrupt.
- static uint32_t AFE_GetEnabledChannelInterrupts (AFE_Type *base)
  Returns mask of all enabled AFE interrupts.
- void AFE_EnableChannelDMA (AFE_Type *base, uint32_t mask, bool enable)
  Enables/Disables AFE DMA.

6.4 Data Structure Documentation

6.4.1 struct afe_channel_config_t

This structure keeps the configuration for the AFE channel.

Data Fields

- bool enableHardwareTrigger
  Enable triggering by hardware.
- bool enableContinuousConversion
  Enable continuous conversion mode.
- afe_bypass_mode_t channelMode
  Select if channel is in bypassed mode.
- afe_pga_gain_t pgaGainSelect
  Select the analog gain applied to the input signal.
- afe_decimator_oversample_ratio_t decimatorOversampleRatio
  Select the over sampling ration.

Field Documentation

(1) bool afe_channel_config_t::enableHardwareTrigger

(2) bool afe_channel_config_t::enableContinuousConversion

(3) afe_bypass_mode_t afe_channel_config_t::channelMode

(4) afe_pga_gain_t afe_channel_config_t::pgaGainSelect

(5) afe_decimator_oversample_ratio_t afe_channel_config_t::decimatorOversampleRatio

6.4.2 struct afe_config_t

This structure keeps the configuration for the AFE module.

Data Fields

- bool enableLowPower
  Enable low power mode.
- afe_result_format_t resultFormat
  Select the result format.
- afe_clock_divider_t clockDivider
  Select the clock divider ration for the modulator clock.
- afe_clock_source_t clockSource
  Select clock source for modulator clock.
- uint8_t startupCount
  Select the start up delay of modulators.

Field Documentation

(1) bool afe_config_t::enableLowPower

(2) afe_result_format_t afe_config_t::resultFormat

(3) afe_clock_divider_t afe_config_t::clockDivider

(4) afe_clock_source_t afe_config_t::clockSource

(5) uint8_t afe_config_t::startupCount

6.5 Macro Definition Documentation

6.5.1 #define FSL_AFE_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

6.6 Enumeration Type Documentation

6.6.1 enum _afe_channel_status_flag

Enumerator

kAFE_Channel0OverflowFlag Channel 0 previous conversion result has not been read and new data has already arrived.
kAFE_Channel1OverflowFlag Channel 1 previous conversion result has not been read and new data has already arrived.
kAFE_Channel2OverflowFlag Channel 2 previous conversion result has not been read and new data has already arrived.
kAFE_Channel0ReadyFlag Channel 0 is ready to conversion.
kAFE_Channel1ReadyFlag Channel 1 is ready to conversion.
kAFE_Channel2ReadyFlag Channel 2 is ready to conversion.
kAFE_Channel0ConversionCompleteFlag Channel 0 conversion is complete.
kAFE_Channel1ConversionCompleteFlag Channel 1 conversion is complete.
kAFE_Channel2ConversionCompleteFlag Channel 2 conversion is complete.
kAFE_Channel3OverflowFlag Channel 3 previous conversion result has not been read and new data has already arrived.

kAFE_Channel3ReadyFlag Channel 3 is ready to conversion.
kAFE_Channel3ConversionCompleteFlag Channel 3 conversion is complete.

6.6.2 anonymous enum

Enumerator

kAFE_Channel0InterruptEnable Channel 0 Interrupt.
kAFE_Channel1InterruptEnable Channel 1 Interrupt.
kAFE_Channel2InterruptEnable Channel 2 Interrupt.
kAFE_Channel3InterruptEnable Channel 3 Interrupt.

6.6.3 anonymous enum

Enumerator

kAFE_Channel0DMAEnable Channel 0 DMA.
kAFE_Channel1DMAEnable Channel 1 DMA.
kAFE_Channel2DMAEnable Channel 2 DMA.
kAFE_Channel3DMAEnable Channel 3 DMA.

6.6.4 anonymous enum

Enumerator

kAFE_Channel0Trigger Channel 0 software trigger.
kAFE_Channel1Trigger Channel 1 software trigger.
kAFE_Channel2Trigger Channel 2 software trigger.
kAFE_Channel3Trigger Channel 3 software trigger.

6.6.5 enum afe_decimator_oversample_ratio_t

Enumerator

kAFE_DecimatorOversampleRatio64 Decimator over sample ratio is 64.
kAFE_DecimatorOversampleRatio128 Decimator over sample ratio is 128.
kAFE_DecimatorOversampleRatio256 Decimator over sample ratio is 256.
kAFE_DecimatorOversampleRatio512 Decimator over sample ratio is 512.
kAFE_DecimatorOversampleRatio1024 Decimator over sample ratio is 1024.
kAFE_DecimatorOversampleRatio2048 Decimator over sample ratio is 2048.

6.6.6 enum afe_result_format_t

Enumerator

kAFE_ResultFormatLeft Left justified result format.
kAFE_ResultFormatRight Right justified result format.

6.6.7 enum afe_clock_divider_t

Enumerator

kAFE_ClockDivider1 Clock divided by 1.
kAFE_ClockDivider2 Clock divided by 2.
kAFE_ClockDivider4 Clock divided by 4.
kAFE_ClockDivider8 Clock divided by 8.
kAFE_ClockDivider16 Clock divided by 16.
kAFE_ClockDivider32 Clock divided by 32.
kAFE_ClockDivider64 Clock divided by 64.
kAFE_ClockDivider128 Clock divided by 128.
kAFE_ClockDivider256 Clock divided by 256.

6.6.8 enum afe_clock_source_t

Enumerator

kAFE_ClockSource0 Modulator clock source 0.
kAFE_ClockSource1 Modulator clock source 1.
kAFE_ClockSource2 Modulator clock source 2.
kAFE_ClockSource3 Modulator clock source 3.

6.6.9 enum afe_pga_gain_t

Enumerator

kAFE_PgaDisable PGA disabled.
kAFE_PgaGain1 Input gained by 1.
kAFE_PgaGain2 Input gained by 2.
kAFE_PgaGain4 Input gained by 4.
kAFE_PgaGain8 Input gained by 8.
kAFE_PgaGain16 Input gained by 16.
kAFE_PgaGain32 Input gained by 32.

6.6.10 enum afe_bypass_mode_t

Enumerator

kAFE_BypassInternalClockPositiveEdge Bypassed channel mode - internal clock selected, positive edge for registering data by the decimation filter.

kAFE_BypassExternalClockPositiveEdge Bypassed channel mode - external clock selected, positive edge for registering data by the decimation filter.

kAFE_BypassInternalClockNegativeEdge Bypassed channel mode - internal clock selected, negative edge for registering data by the decimation filter.

kAFE_BypassExternalClockNegativeEdge Bypassed channel mode - external clock selected, negative edge for registering data by the decimation filter.

kAFE_BypassDisable Normal channel mode.

6.7 Function Documentation

6.7.1 void AFE_Init (AFE_Type * *base*, const afe_config_t * *config*)

This function configures the AFE module for the configuration which are shared by all channels.

Parameters

base	AFE peripheral base address.
config	Pointer to structure of "afe_config_t".

6.7.2 void AFE_Deinit (AFE_Type * *base*)

This function disables clock.

Parameters

base	AFE peripheral base address.

6.7.3 void AFE_GetDefaultConfig (afe_config_t * *config*)

This function fills the afe_config_t structure with default settings. Defaut value are:

```
* config->enableLowPower    = false;
* config->resultFormat      = kAFE_ResultFormatRight;
* config->clockDivider      = kAFE_ClockDivider2;
* config->clockSource       = kAFE_ClockSource1;
* config->startupCount      = 2U;
*
```

Parameters

config	Pointer to structure of "afe_config_t".

6.7.4 static void AFE_SoftwareReset (AFE_Type * *base*, bool *enable*) [inline], [static]

This function is to reset all the ADCs, PGAs, decimation filters and clock configuration bits. When asserted as "false", all ADCs, PGAs and decimation filters are disabled. Clock Configuration bits are reset. When asserted as "true", all ADCs, PGAs and decimation filters are enabled.

Parameters

base	AFE peripheral base address.
enable	Assert the reset command.

6.7.5 static void AFE_Enable (AFE_Type * *base*, bool *enable*) [inline], [static]

This function enables AFE and filter.

Parameters

base	AFE peripheral base address.
enable	Enable the AFE module or not.

6.7.6 void AFE_SetChannelConfig (AFE_Type * *base*, uint32_t *channel*, const afe_channel_config_t * *config*)

This function configures the selected AFE channel.

Parameters

base	AFE peripheral base address.
channel	AFE channel index.

config	Pointer to structure of "afe_channel_config_t".

6.7.7 void AFE_GetDefaultChannelConfig (afe_channel_config_t * *config*)

This function fills the afe_channel_config_t structure with default settings. Default value are:

```
* config->enableHardwareTrigger      = false;
* config->enableContinuousConversion = false;
* config->channelMode                = kAFE_Normal;
* config->decimatorOversampleRatio   = kAFE_DecimatorOversampleRatio64;
* config->pgaGainSelect              = kAFE_PgaGain1;
*
```

Parameters

config	Pointer to structure of "afe_channel_config_t".

6.7.8 uint32_t AFE_GetChannelConversionValue (AFE_Type * *base,* uint32_t *channel*)

This function returns the raw conversion value of the selected channel.

Parameters

base	AFE peripheral base address.
channel	AFE channel index.

Returns

Conversion value.

Note

The returned value could be left or right adjusted according to the AFE module configuration.

6.7.9 static void AFE_DoSoftwareTriggerChannel (AFE_Type * *base,* uint32_t *mask*) [inline], [static]

This function triggers the AFE conversion by executing a software command. It starts the conversion on selected channels if the software trigger option is selected for the channels.

Parameters

base	AFE peripheral base address.
mask	AFE channel mask software trigger. The parameter can be combination of the following source if defined: • kAFE_Channel0Trigger • kAFE_Channel1Trigger • kAFE_Channel2Trigger • kAFE_Channel3Trigger

6.7.10 static uint32_t AFE_GetChannelStatusFlags (AFE_Type * *base*) [inline], [static]

This function gets all AFE status.

Parameters

base	AFE peripheral base address.

Returns

the mask of these status flag bits.

6.7.11 void AFE_SetChannelPhaseDelayValue (AFE_Type * *base*, uint32_t *channel*, uint32_t *value*)

This function sets the phase delays for channels. This delay is inserted before the trigger response of the decimation filters. The delay is used to provide a phase compensation between AFE channels in step of prescaled modulator clock periods.

Parameters

base	AFE peripheral base address.
channel	AFE channel index.

value	delay time value.

6.7.12 static void AFE_SetChannelPhasetDelayOk (AFE_Type ∗ *base*) [inline], [static]

This function should be called after all desired channel's delay registers are loaded. Values in channel's delay registers are active after calling this function and after the conversation starts.

Parameters

base	AFE peripheral base address.

6.7.13 static void AFE_EnableChannelInterrupts (AFE_Type ∗ *base,* uint32_t *mask*) [inline], [static]

This function enables one channel interrupt.

Parameters

base	AFE peripheral base address.
mask	AFE channel interrupt mask. The parameter can be combination of the following source if defined: • kAFE_Channel0InterruptEnable • kAFE_Channel1InterruptEnable • kAFE_Channel2InterruptEnable • kAFE_Channel3InterruptEnable

6.7.14 static void AFE_DisableChannelInterrupts (AFE_Type ∗ *base,* uint32_t *mask*) [inline], [static]

This function disables one channel interrupt.

Parameters

base	AFE peripheral base address.
mask	AFE channel interrupt mask. The parameter can be combination of the following source if defined: • kAFE_Channel0InterruptEnable • kAFE_Channel1InterruptEnable • kAFE_Channel2InterruptEnable • kAFE_Channel3InterruptEnable

6.7.15 static uint32_t AFE_GetEnabledChannelInterrupts (AFE_Type * *base*) [inline], [static]

Parameters

base	AFE peripheral base address.

Returns

Return the mask of these interrupt enable/disable bits.

6.7.16 void AFE_EnableChannelDMA (AFE_Type * *base*, uint32_t *mask*, bool *enable*)

This function enables/disables one channel DMA request.

Parameters

base	AFE peripheral base address.
mask	AFE channel dma mask.
enable	Pass true to enable interrupt, false to disable. The parameter can be combination of the following source if defined: • kAFE_Channel0DMAEnable • kAFE_Channel1DMAEnable • kAFE_Channel2DMAEnable • kAFE_Channel3DMAEnable

Chapter 7
CMP: Analog Comparator Driver

7.1 Overview

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

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

7.2 Typical use case

7.2.1 Polling Configuration

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

7.2.2 Interrupt Configuration

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

Data Structures

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

Enumerations

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

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

Driver version

- #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  CMP driver version 2.0.2.

Initialization

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

Advanced Features

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

Results

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

7.3 Data Structure Documentation

7.3.1 struct cmp_config_t

Data Fields

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

Field Documentation

(1) bool cmp_config_t::enableCmp

(2) cmp_hysteresis_mode_t cmp_config_t::hysteresisMode

(3) bool cmp_config_t::enableHighSpeed

(4) bool cmp_config_t::enableInvertOutput

(5) bool cmp_config_t::useUnfilteredOutput

(6) bool cmp_config_t::enablePinOut

(7) bool cmp_config_t::enableTriggerMode

7.3.2 struct cmp_filter_config_t

Data Fields

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

Field Documentation

(1) bool cmp_filter_config_t::enableSample

(2) uint8_t cmp_filter_config_t::filterCount

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

(3) uint8_t cmp_filter_config_t::filterPeriod

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

7.3.3 struct cmp_dac_config_t

Data Fields

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

Field Documentation

(1) cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource

(2) uint8_t cmp_dac_config_t::DACValue

Available range is 0-63.

7.4 Macro Definition Documentation

7.4.1 #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

7.5 Enumeration Type Documentation

7.5.1 enum _cmp_interrupt_enable

Enumerator

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

7.5.2 enum _cmp_status_flags

Enumerator

kCMP_OutputRisingEventFlag Rising-edge on the comparison output has occurred.
kCMP_OutputFallingEventFlag Falling-edge on the comparison output has occurred.

kCMP_OutputAssertEventFlag Return the current value of the analog comparator output.

7.5.3 enum cmp_hysteresis_mode_t

Enumerator

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

7.5.4 enum cmp_reference_voltage_source_t

Enumerator

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

7.6 Function Documentation

7.6.1 void CMP_Init (CMP_Type ∗ *base,* const cmp_config_t ∗ *config*)

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

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

Parameters

base	CMP peripheral base address.
config	Pointer to the configuration structure.

7.6.2 void CMP_Deinit (CMP_Type ∗ *base*)

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

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

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

Parameters

base	CMP peripheral base address.

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

Parameters

base	CMP peripheral base address.
enable	Enables or disables the module.

7.6.4 void CMP_GetDefaultConfig (cmp_config_t * *config*)

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

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

Parameters

config	Pointer to the configuration structure.

7.6.5 void CMP_SetInputChannels (CMP_Type * *base*, uint8_t *positiveChannel*, uint8_t *negativeChannel*)

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

Parameters

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

7.6.6 void CMP_EnableDMA (CMP_Type * *base*, bool *enable*)

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

Parameters

base	CMP peripheral base address.
enable	Enables or disables the feature.

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

Parameters

base	CMP peripheral base address.
enable	Enables or disables the feature.

7.6.8 void CMP_SetFilterConfig (CMP_Type * *base*, const cmp_filter_config_t * *config*)

Parameters

base	CMP peripheral base address.
config	Pointer to the configuration structure.

7.6.9 void CMP_SetDACConfig (CMP_Type * *base*, const cmp_dac_config_t * *config*)

Parameters

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

7.6.10 void CMP_EnableInterrupts (CMP_Type * *base*, uint32_t *mask*)

Parameters

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

7.6.11 void CMP_DisableInterrupts (CMP_Type * *base*, uint32_t *mask*)

Parameters

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

7.6.12 uint32_t CMP_GetStatusFlags (CMP_Type * *base*)

Parameters

base	CMP peripheral base address.

Returns

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

7.6.13 void CMP_ClearStatusFlags (CMP_Type * *base*, uint32_t *mask*)

Parameters

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

Chapter 8
Common Driver

8.1 Overview

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

Macros

- #define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1
  Macro to use the default weak IRQ handler in drivers.
- #define MAKE_STATUS(group, code) ((((group)∗100L) + (code)))
  Construct a status code value from a group and code number.
- #define MAKE_VERSION(major, minor, bugfix) (((major) ∗ 65536L) + ((minor) ∗ 256L) + (bugfix))
  Construct the version number for drivers.
- #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U
  No debug console.
- #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U
  Debug console based on UART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U
  Debug console based on LPUART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U
  Debug console based on LPSCI.
- #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U
  Debug console based on USBCDC.
- #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U
  Debug console based on FLEXCOMM.
- #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U
  Debug console based on i.MX UART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U
  Debug console based on LPC_VUSART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U
  Debug console based on LPC_USART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U
  Debug console based on SWO.
- #define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U
  Debug console based on QSCI.
- #define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
  Computes the number of elements in an array.

Typedefs

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

Enumerations

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

kStatusGroup_BMA = 164 }
Status group numbers.
- enum {
kStatus_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
kStatus_Fail = MAKE_STATUS(kStatusGroup_Generic, 1),
kStatus_ReadOnly = MAKE_STATUS(kStatusGroup_Generic, 2),
kStatus_OutOfRange = MAKE_STATUS(kStatusGroup_Generic, 3),
kStatus_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
kStatus_Timeout = MAKE_STATUS(kStatusGroup_Generic, 5),
kStatus_NoTransferInProgress,
kStatus_Busy = MAKE_STATUS(kStatusGroup_Generic, 7),
kStatus_NoData }
Generic status return codes.

Functions

- void * SDK_Malloc (size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.
- void SDK_Free (void *ptr)
Free memory.
- void SDK_DelayAtLeastUs (uint32_t delayTime_us, uint32_t coreClock_Hz)
Delay at least for some time.

Driver version

- #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))
common driver version.

Min/max macros

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

UINT16_MAX/UINT32_MAX value

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

Suppress fallthrough warning macro

- #define **SUPPRESS_FALL_THROUGH_WARNING**()

8.2 Macro Definition Documentation

8.2.1 #define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1

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

8.2.3 #define MAKE_VERSION(*major*, *minor*, *bugfix*) (((major) ∗ 65536L) + ((minor) ∗ 256L) + (bugfix))

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

```
| Unused    || Major Version || Minor Version ||  Bug Fix    |
31        25  24           17  16            9  8            0
```

8.2.4 #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))

8.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

8.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

8.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

8.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

8.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

8.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

8.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

8.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

8.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

8.2.14 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

8.2.15 #define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U

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

8.3 Typedef Documentation

8.3.1 typedef int32_t status_t

8.4 Enumeration Type Documentation

8.4.1 enum _status_groups

Enumerator

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

kStatusGroup_FLEXCAN Group number for FlexCAN status codes.
kStatusGroup_LTC Group number for LTC status codes.
kStatusGroup_FLEXIO_CAMERA Group number for FLEXIO CAMERA status codes.
kStatusGroup_LPC_SPI Group number for LPC_SPI status codes.
kStatusGroup_LPC_USART Group number for LPC_USART status codes.
kStatusGroup_DMIC Group number for DMIC status codes.
kStatusGroup_SDIF Group number for SDIF status codes.
kStatusGroup_SPIFI Group number for SPIFI status codes.
kStatusGroup_OTP Group number for OTP status codes.
kStatusGroup_MCAN Group number for MCAN status codes.
kStatusGroup_CAAM Group number for CAAM status codes.
kStatusGroup_ECSPI Group number for ECSPI status codes.
kStatusGroup_USDHC Group number for USDHC status codes.
kStatusGroup_LPC_I2C Group number for LPC_I2C status codes.
kStatusGroup_DCP Group number for DCP status codes.
kStatusGroup_MSCAN Group number for MSCAN status codes.
kStatusGroup_ESAI Group number for ESAI status codes.
kStatusGroup_FLEXSPI Group number for FLEXSPI status codes.
kStatusGroup_MMDC Group number for MMDC status codes.
kStatusGroup_PDM Group number for MIC status codes.
kStatusGroup_SDMA Group number for SDMA status codes.
kStatusGroup_ICS Group number for ICS status codes.
kStatusGroup_SPDIF Group number for SPDIF status codes.
kStatusGroup_LPC_MINISPI Group number for LPC_MINISPI status codes.
kStatusGroup_HASHCRYPT Group number for Hashcrypt status codes.
kStatusGroup_LPC_SPI_SSP Group number for LPC_SPI_SSP status codes.
kStatusGroup_I3C Group number for I3C status codes.
kStatusGroup_LPC_I2C_1 Group number for LPC_I2C_1 status codes.
kStatusGroup_NOTIFIER Group number for NOTIFIER status codes.
kStatusGroup_DebugConsole Group number for debug console status codes.
kStatusGroup_SEMC Group number for SEMC status codes.
kStatusGroup_ApplicationRangeStart Starting number for application groups.
kStatusGroup_IAP Group number for IAP status codes.
kStatusGroup_SFA Group number for SFA status codes.
kStatusGroup_SPC Group number for SPC status codes.
kStatusGroup_PUF Group number for PUF status codes.
kStatusGroup_TOUCH_PANEL Group number for touch panel status codes.
kStatusGroup_HAL_GPIO Group number for HAL GPIO status codes.
kStatusGroup_HAL_UART Group number for HAL UART status codes.
kStatusGroup_HAL_TIMER Group number for HAL TIMER status codes.
kStatusGroup_HAL_SPI Group number for HAL SPI status codes.
kStatusGroup_HAL_I2C Group number for HAL I2C status codes.
kStatusGroup_HAL_FLASH Group number for HAL FLASH status codes.
kStatusGroup_HAL_PWM Group number for HAL PWM status codes.
kStatusGroup_HAL_RNG Group number for HAL RNG status codes.

kStatusGroup_HAL_I2S Group number for HAL I2S status codes.
kStatusGroup_TIMERMANAGER Group number for TiMER MANAGER status codes.
kStatusGroup_SERIALMANAGER Group number for SERIAL MANAGER status codes.
kStatusGroup_LED Group number for LED status codes.
kStatusGroup_BUTTON Group number for BUTTON status codes.
kStatusGroup_EXTERN_EEPROM Group number for EXTERN EEPROM status codes.
kStatusGroup_SHELL Group number for SHELL status codes.
kStatusGroup_MEM_MANAGER Group number for MEM MANAGER status codes.
kStatusGroup_LIST Group number for List status codes.
kStatusGroup_OSA Group number for OSA status codes.
kStatusGroup_COMMON_TASK Group number for Common task status codes.
kStatusGroup_MSG Group number for messaging status codes.
kStatusGroup_SDK_OCOTP Group number for OCOTP status codes.
kStatusGroup_SDK_FLEXSPINOR Group number for FLEXSPINOR status codes.
kStatusGroup_CODEC Group number for codec status codes.
kStatusGroup_ASRC Group number for codec status ASRC.
kStatusGroup_OTFAD Group number for codec status codes.
kStatusGroup_SDIOSLV Group number for SDIOSLV status codes.
kStatusGroup_MECC Group number for MECC status codes.
kStatusGroup_ENET_QOS Group number for ENET_QOS status codes.
kStatusGroup_LOG Group number for LOG status codes.
kStatusGroup_I3CBUS Group number for I3CBUS status codes.
kStatusGroup_QSCI Group number for QSCI status codes.
kStatusGroup_SNT Group number for SNT status codes.
kStatusGroup_QUEUEDSPI Group number for QSPI status codes.
kStatusGroup_POWER_MANAGER Group number for POWER_MANAGER status codes.
kStatusGroup_IPED Group number for IPED status codes.
kStatusGroup_CSS_PKC Group number for CSS PKC status codes.
kStatusGroup_HOSTIF Group number for HOSTIF status codes.
kStatusGroup_CLIF Group number for CLIF status codes.
kStatusGroup_BMA Group number for BMA status codes.

8.4.2 anonymous enum

Enumerator

kStatus_Success Generic status for Success.
kStatus_Fail Generic status for Fail.
kStatus_ReadOnly Generic status for read only failure.
kStatus_OutOfRange Generic status for out of range access.
kStatus_InvalidArgument Generic status for invalid argument check.
kStatus_Timeout Generic status for timeout.
kStatus_NoTransferInProgress Generic status for no transfer in progress.
kStatus_Busy Generic status for module is busy.

kStatus_NoData Generic status for no data is found for the operation.

8.5 Function Documentation

8.5.1 void* SDK_Malloc (size_t *size,* size_t *alignbytes*)

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

Parameters

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

Return values

The	allocated memory.

8.5.2 void SDK_Free (void * *ptr*)

Parameters

ptr	The memory to be release.

8.5.3 void SDK_DelayAtLeastUs (uint32_t *delayTime_us,* uint32_t *coreClock_Hz*)

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

Parameters

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

Chapter 9
CRC: Cyclic Redundancy Check Driver

9.1 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.

9.2 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.

9.3 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.

9.4 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.

9.5 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.

9.6 Data Structure Documentation

9.6.1 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

9.7 Macro Definition Documentation

9.7.1 #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

9.7.2 #define CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT 1

Use CRC16-CCIT-FALSE as defeault.

9.8 Enumeration Type Documentation

9.8.1 enum crc_bits_t

Enumerator

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

9.8.2 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.

9.9 Function Documentation

9.9.1 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.

9.9.2 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.

9.9.3 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.

9.9.4 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.

9.9.5 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.

9.9.6 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 10
DMA: Direct Memory Access Controller Driver

10.1 Overview

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

10.2 Typical use case

10.2.1 DMA Operation

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

Data Structures

- struct dma_transfer_config_t
  DMA transfer configuration structure. More...
- struct dma_channel_link_config_t
  DMA transfer configuration structure. More...
- struct dma_handle_t
  DMA DMA handle structure. More...

Typedefs

- typedef void(∗ dma_callback)(struct _dma_handle ∗handle, void ∗userData)
  Callback function prototype for the DMA driver.

Enumerations

- enum {
  kDMA_TransactionsBCRFlag = DMA_DSR_BCR_BCR_MASK,
  kDMA_TransactionsDoneFlag = DMA_DSR_BCR_DONE_MASK,
  kDMA_TransactionsBusyFlag = DMA_DSR_BCR_BSY_MASK,
  kDMA_TransactionsRequestFlag = DMA_DSR_BCR_REQ_MASK,
  kDMA_BusErrorOnDestinationFlag = DMA_DSR_BCR_BED_MASK,
  kDMA_BusErrorOnSourceFlag = DMA_DSR_BCR_BES_MASK,
  kDMA_ConfigurationErrorFlag = DMA_DSR_BCR_CE_MASK }
  _dma_channel_status_flags status flag for the DMA driver.
- enum dma_transfer_size_t {
  kDMA_Transfersize32bits = 0x0U,
  kDMA_Transfersize8bits,
  kDMA_Transfersize16bits }
  DMA transfer size type.

- enum dma_modulo_t {
kDMA_ModuloDisable = 0x0U,
kDMA_Modulo16Bytes,
kDMA_Modulo32Bytes,
kDMA_Modulo64Bytes,
kDMA_Modulo128Bytes,
kDMA_Modulo256Bytes,
kDMA_Modulo512Bytes,
kDMA_Modulo1KBytes,
kDMA_Modulo2KBytes,
kDMA_Modulo4KBytes,
kDMA_Modulo8KBytes,
kDMA_Modulo16KBytes,
kDMA_Modulo32KBytes,
kDMA_Modulo64KBytes,
kDMA_Modulo128KBytes,
kDMA_Modulo256KBytes }
Configuration type for the DMA modulo.
- enum dma_channel_link_type_t {
kDMA_ChannelLinkDisable = 0x0U,
kDMA_ChannelLinkChannel1AndChannel2,
kDMA_ChannelLinkChannel1,
kDMA_ChannelLinkChannel1AfterBCR0 }
DMA channel link type.
- enum dma_transfer_type_t {
kDMA_MemoryToMemory = 0x0U,
kDMA_PeripheralToMemory,
kDMA_MemoryToPeripheral }
DMA transfer type.
- enum dma_transfer_options_t {
kDMA_NoOptions = 0x0U,
kDMA_EnableInterrupt }
DMA transfer options.
- enum dma_addr_increment_t {
kDMA_AddrNoIncrement = 0x0U,
kDMA_AddrIncrementPerTransferWidth = 0x1U }
dma addre increment type
- enum { kStatus_DMA_Busy = MAKE_STATUS(kStatusGroup_DMA, 0) }
_dma_transfer_status DMA transfer status

Driver version

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

DMA Initialization and De-initialization

- void DMA_Init (DMA_Type ∗base)
  Initializes the DMA peripheral.
- void DMA_Deinit (DMA_Type ∗base)
  Deinitializes the DMA peripheral.

DMA Channel Operation

- void DMA_ResetChannel (DMA_Type ∗base, uint32_t channel)
  Resets the DMA channel.
- void DMA_SetTransferConfig (DMA_Type ∗base, uint32_t channel, const dma_transfer_config_t ∗config)
  Configures the DMA transfer attribute.
- void DMA_SetChannelLinkConfig (DMA_Type ∗base, uint32_t channel, const dma_channel_link-_config_t ∗config)
  Configures the DMA channel link feature.
- static void DMA_SetSourceAddress (DMA_Type ∗base, uint32_t channel, uint32_t srcAddr)
  Sets the DMA source address for the DMA transfer.
- static void DMA_SetDestinationAddress (DMA_Type ∗base, uint32_t channel, uint32_t destAddr)
  Sets the DMA destination address for the DMA transfer.
- static void DMA_SetTransferSize (DMA_Type ∗base, uint32_t channel, uint32_t size)
  Sets the DMA transfer size for the DMA transfer.
- void DMA_SetModulo (DMA_Type ∗base, uint32_t channel, dma_modulo_t srcModulo, dma_-modulo_t destModulo)
  Sets the DMA modulo for the DMA transfer.
- static void DMA_EnableCycleSteal (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables the DMA cycle steal for the DMA transfer.
- static void DMA_EnableAutoAlign (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables the DMA auto align for the DMA transfer.
- static void DMA_EnableAsyncRequest (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables the DMA async request for the DMA transfer.
- static void DMA_EnableInterrupts (DMA_Type ∗base, uint32_t channel)
  Enables an interrupt for the DMA transfer.
- static void DMA_DisableInterrupts (DMA_Type ∗base, uint32_t channel)
  Disables an interrupt for the DMA transfer.

DMA Channel Transfer Operation

- static void DMA_EnableChannelRequest (DMA_Type ∗base, uint32_t channel)
  Enables the DMA hardware channel request.
- static void DMA_DisableChannelRequest (DMA_Type ∗base, uint32_t channel)
  Disables the DMA hardware channel request.
- static void DMA_TriggerChannelStart (DMA_Type ∗base, uint32_t channel)
  Starts the DMA transfer with a software trigger.
- static void DMA_EnableAutoStopRequest (DMA_Type ∗base, uint32_t channel, bool enable)
  Starts the DMA enable/disable auto disable request.

DMA Channel Status Operation

- static uint32_t DMA_GetRemainingBytes (DMA_Type ∗base, uint32_t channel)

Gets the remaining bytes of the current DMA transfer.
- static uint32_t DMA_GetChannelStatusFlags (DMA_Type ∗base, uint32_t channel)
  Gets the DMA channel status flags.
- static void DMA_ClearChannelStatusFlags (DMA_Type ∗base, uint32_t channel, uint32_t mask)
  Clears the DMA channel status flags.

DMA Channel Transactional Operation

- void DMA_CreateHandle (dma_handle_t ∗handle, DMA_Type ∗base, uint32_t channel)
  Creates the DMA handle.
- void DMA_SetCallback (dma_handle_t ∗handle, dma_callback callback, void ∗userData)
  Sets the DMA callback function.
- void DMA_PrepareTransferConfig (dma_transfer_config_t ∗config, void ∗srcAddr, uint32_t srcWidth, void ∗destAddr, uint32_t destWidth, uint32_t transferBytes, dma_addr_increment_t srcIncrement, dma_addr_increment_t destIncrement)
  Prepares the DMA transfer configuration structure.
- void DMA_PrepareTransfer (dma_transfer_config_t ∗config, void ∗srcAddr, uint32_t srcWidth, void ∗destAddr, uint32_t destWidth, uint32_t transferBytes, dma_transfer_type_t type)
  Prepares the DMA transfer configuration structure.
- status_t DMA_SubmitTransfer (dma_handle_t ∗handle, const dma_transfer_config_t ∗config, uint32_t options)
  Submits the DMA transfer request.
- static void DMA_StartTransfer (dma_handle_t ∗handle)
  DMA starts a transfer.
- static void DMA_StopTransfer (dma_handle_t ∗handle)
  DMA stops a transfer.
- void DMA_AbortTransfer (dma_handle_t ∗handle)
  DMA aborts a transfer.
- void DMA_HandleIRQ (dma_handle_t ∗handle)
  DMA IRQ handler for current transfer complete.

10.3 Data Structure Documentation

10.3.1 struct dma_transfer_config_t

Data Fields

- uint32_t srcAddr
  DMA transfer source address.
- uint32_t destAddr
  DMA destination address.
- bool enableSrcIncrement
  Source address increase after each transfer.
- dma_transfer_size_t srcSize
  Source transfer size unit.
- bool enableDestIncrement
  Destination address increase after each transfer.
- dma_transfer_size_t destSize
  Destination transfer unit.
- uint32_t transferSize

The number of bytes to be transferred.

Field Documentation

(1) uint32_t dma_transfer_config_t::srcAddr

(2) uint32_t dma_transfer_config_t::destAddr

(3) bool dma_transfer_config_t::enableSrcIncrement

(4) dma_transfer_size_t dma_transfer_config_t::srcSize

(5) bool dma_transfer_config_t::enableDestIncrement

(6) dma_transfer_size_t dma_transfer_config_t::destSize

(7) uint32_t dma_transfer_config_t::transferSize

10.3.2 struct dma_channel_link_config_t

Data Fields

- dma_channel_link_type_t linkType
  Channel link type.
- uint32_t channel1
  The index of channel 1.
- uint32_t channel2
  The index of channel 2.

Field Documentation

(1) dma_channel_link_type_t dma_channel_link_config_t::linkType

(2) uint32_t dma_channel_link_config_t::channel1

(3) uint32_t dma_channel_link_config_t::channel2

10.3.3 struct dma_handle_t

Data Fields

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

Field Documentation

(1) DMA_Type∗ dma_handle_t::base

(2) uint8_t dma_handle_t::channel

(3) dma_callback dma_handle_t::callback

(4) void∗ dma_handle_t::userData

10.4 Macro Definition Documentation

10.4.1 #define FSL_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

10.5 Typedef Documentation

10.5.1 typedef void(∗ dma_callback)(struct _dma_handle ∗handle, void ∗userData)

10.6 Enumeration Type Documentation

10.6.1 anonymous enum

Enumerator

kDMA_TransactionsBCRFlag Contains the number of bytes yet to be transferred for a given block.

kDMA_TransactionsDoneFlag Transactions Done.
kDMA_TransactionsBusyFlag Transactions Busy.
kDMA_TransactionsRequestFlag Transactions Request.
kDMA_BusErrorOnDestinationFlag Bus Error on Destination.
kDMA_BusErrorOnSourceFlag Bus Error on Source.
kDMA_ConfigurationErrorFlag Configuration Error.

10.6.2 enum dma_transfer_size_t

Enumerator

kDMA_Transfersize32bits 32 bits are transferred for every read/write
kDMA_Transfersize8bits 8 bits are transferred for every read/write
kDMA_Transfersize16bits 16b its are transferred for every read/write

10.6.3 enum dma_modulo_t

Enumerator

kDMA_ModuloDisable Buffer disabled.

kDMA_Modulo16Bytes Circular buffer size is 16 bytes.
kDMA_Modulo32Bytes Circular buffer size is 32 bytes.
kDMA_Modulo64Bytes Circular buffer size is 64 bytes.
kDMA_Modulo128Bytes Circular buffer size is 128 bytes.
kDMA_Modulo256Bytes Circular buffer size is 256 bytes.
kDMA_Modulo512Bytes Circular buffer size is 512 bytes.
kDMA_Modulo1KBytes Circular buffer size is 1 KB.
kDMA_Modulo2KBytes Circular buffer size is 2 KB.
kDMA_Modulo4KBytes Circular buffer size is 4 KB.
kDMA_Modulo8KBytes Circular buffer size is 8 KB.
kDMA_Modulo16KBytes Circular buffer size is 16 KB.
kDMA_Modulo32KBytes Circular buffer size is 32 KB.
kDMA_Modulo64KBytes Circular buffer size is 64 KB.
kDMA_Modulo128KBytes Circular buffer size is 128 KB.
kDMA_Modulo256KBytes Circular buffer size is 256 KB.

10.6.4 enum dma_channel_link_type_t

Enumerator

kDMA_ChannelLinkDisable No channel link.
kDMA_ChannelLinkChannel1AndChannel2 Perform a link to channel LCH1 after each cycle-steal transfer. followed by a link to LCH2 after the BCR decrements to 0.
kDMA_ChannelLinkChannel1 Perform a link to LCH1 after each cycle-steal transfer.
kDMA_ChannelLinkChannel1AfterBCR0 Perform a link to LCH1 after the BCR decrements.

10.6.5 enum dma_transfer_type_t

Enumerator

kDMA_MemoryToMemory Memory to Memory transfer.
kDMA_PeripheralToMemory Peripheral to Memory transfer.
kDMA_MemoryToPeripheral Memory to Peripheral transfer.

10.6.6 enum dma_transfer_options_t

Enumerator

kDMA_NoOptions Transfer without options.
kDMA_EnableInterrupt Enable interrupt while transfer complete.

10.6.7 enum dma_addr_increment_t

Enumerator

kDMA_AddrNoIncrement Transfer address not increment.
kDMA_AddrIncrementPerTransferWidth Transfer address increment per transfer width.

10.6.8 anonymous enum

Enumerator

kStatus_DMA_Busy DMA is busy.

10.7 Function Documentation

10.7.1 void DMA_Init (DMA_Type * *base*)

This function ungates the DMA clock.

Parameters

base	DMA peripheral base address.

10.7.2 void DMA_Deinit (DMA_Type * *base*)

This function gates the DMA clock.

Parameters

base	DMA peripheral base address.

10.7.3 void DMA_ResetChannel (DMA_Type * *base,* uint32_t *channel*)

Sets all register values to reset values and enables the cycle steal and auto stop channel request features.

Parameters

base	DMA peripheral base address.

channel	DMA channel number.

10.7.4 void DMA_SetTransferConfig (DMA_Type * *base*, uint32_t *channel*, const dma_transfer_config_t * *config*)

This function configures the transfer attribute including the source address, destination address, transfer size, and so on. This example shows how to set up the dma_transfer_config_t parameters and how to call the DMA_ConfigBasicTransfer function.

```
*   dma_transfer_config_t transferConfig;
*   memset(&transferConfig, 0, sizeof(transferConfig));
*   transferConfig.srcAddr = (uint32_t)srcAddr;
*   transferConfig.destAddr = (uint32_t)destAddr;
*   transferConfig.enbaleSrcIncrement = true;
*   transferConfig.enableDestIncrement = true;
*   transferConfig.srcSize = kDMA_Transfersize32bits;
*   transferConfig.destSize = kDMA_Transfersize32bits;
*   transferConfig.transferSize = sizeof(uint32_t) * BUFF_LENGTH;
*   DMA_SetTransferConfig(DMA0, 0, &transferConfig);
*
```

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
config	Pointer to the DMA transfer configuration structure.

10.7.5 void DMA_SetChannelLinkConfig (DMA_Type * *base*, uint32_t *channel*, const dma_channel_link_config_t * *config*)

This function allows DMA channels to have their transfers linked. The current DMA channel triggers a DMA request to the linked channels (LCH1 or LCH2) depending on the channel link type. Perform a link to channel LCH1 after each cycle-steal transfer followed by a link to LCH2 after the BCR decrements to 0 if the type is kDMA_ChannelLinkChannel1AndChannel2. Perform a link to LCH1 after each cycle-steal transfer if the type is kDMA_ChannelLinkChannel1. Perform a link to LCH1 after the BCR decrements to 0 if the type is kDMA_ChannelLinkChannel1AfterBCR0.

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
config	Pointer to the channel link configuration structure.

10.7.6 static void DMA_SetSourceAddress (DMA_Type * *base*, uint32_t *channel*, uint32_t *srcAddr*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
srcAddr	DMA source address.

10.7.7 static void DMA_SetDestinationAddress (DMA_Type * *base*, uint32_t *channel*, uint32_t *destAddr*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
destAddr	DMA destination address.

10.7.8 static void DMA_SetTransferSize (DMA_Type * *base*, uint32_t *channel*, uint32_t *size*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
size	The number of bytes to be transferred.

10.7.9 void DMA_SetModulo (DMA_Type * *base,* uint32_t *channel,* dma_modulo_t *srcModulo,* dma_modulo_t *destModulo*)

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

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
srcModulo	source address modulo.
destModulo	destination address modulo.

10.7.10 static void DMA_EnableCycleSteal (DMA_Type * *base,* uint32_t *channel,* bool *enable*) [inline], [static]

If the cycle steal feature is enabled (true), the DMA controller forces a single read/write transfer per request, or it continuously makes read/write transfers until the BCR decrements to 0.

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
enable	The command for enable (true) or disable (false).

10.7.11 static void DMA_EnableAutoAlign (DMA_Type * *base,* uint32_t *channel,* bool *enable*) [inline], [static]

If the auto align feature is enabled (true), the appropriate address register increments regardless of DINC or SINC.

Parameters

base	DMA peripheral base address.

channel	DMA channel number.
enable	The command for enable (true) or disable (false).

10.7.12 static void DMA_EnableAsyncRequest (DMA_Type * *base*, uint32_t *channel*, bool *enable*) [inline], [static]

If the async request feature is enabled (true), the DMA supports asynchronous DREQs while the MCU is in stop mode.

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
enable	The command for enable (true) or disable (false).

10.7.13 static void DMA_EnableInterrupts (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

10.7.14 static void DMA_DisableInterrupts (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

10.7.15 static void DMA_EnableChannelRequest (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	The DMA channel number.

10.7.16 static void DMA_DisableChannelRequest (DMA_Type ∗ *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

10.7.17 static void DMA_TriggerChannelStart (DMA_Type ∗ *base,* uint32_t *channel*) [inline], [static]

This function starts only one read/write iteration.

Parameters

base	DMA peripheral base address.
channel	The DMA channel number.

10.7.18 static void DMA_EnableAutoStopRequest (DMA_Type ∗ *base,* uint32_t *channel,* bool *enable*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	The DMA channel number.
enable	true is enable, false is disable.

10.7.19 static uint32_t DMA_GetRemainingBytes (DMA_Type ∗ *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

The number of bytes which have not been transferred yet.

10.7.20 static uint32_t DMA_GetChannelStatusFlags (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

The mask of the channel status. Use the _dma_channel_status_flags type to decode the return 32 bit variables.

10.7.21 static void DMA_ClearChannelStatusFlags (DMA_Type * *base,* uint32_t *channel,* uint32_t *mask*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
mask	The mask of the channel status to be cleared. Use the defined _dma_channel_status-_flags type.

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

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

Parameters

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

10.7.23 void DMA_SetCallback (dma_handle_t * *handle*, dma_callback *callback*, void * *userData*)

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

Parameters

handle	DMA handle pointer.
callback	DMA callback function pointer.
userData	Parameter for callback function. If it is not needed, just set to NULL.

10.7.24 void DMA_PrepareTransferConfig (dma_transfer_config_t * *config*, void * *srcAddr*, uint32_t *srcWidth*, void * *destAddr*, uint32_t *destWidth*, uint32_t *transferBytes*, dma_addr_increment_t *srcIncrement*, dma_addr_increment_t *destIncrement*)

This function prepares the transfer configuration structure according to the user input. The difference between this function and DMA_PrepareTransfer is that this function expose the address increment parameter to application, but in DMA_PrepareTransfer, only parts of the address increment option can be selected by dma_transfer_type_t.

Parameters

config	Pointer to the user configuration structure of type dma_transfer_config_t.
srcAddr	DMA transfer source address.
srcWidth	DMA transfer source address width (byte).
destAddr	DMA transfer destination address.
destWidth	DMA transfer destination address width (byte).
transferBytes	DMA transfer bytes to be transferred.
srcIncrement	source address increment type.
destIncrement	dest address increment type.

10.7.25 void DMA_PrepareTransfer (dma_transfer_config_t * *config*, void * *srcAddr*, uint32_t *srcWidth*, void * *destAddr*, uint32_t *destWidth*, uint32_t *transferBytes*, dma_transfer_type_t *type*)

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

Parameters

config	Pointer to the user configuration structure of type dma_transfer_config_t.
srcAddr	DMA transfer source address.
srcWidth	DMA transfer source address width (byte).
destAddr	DMA transfer destination address.
destWidth	DMA transfer destination address width (byte).
transferBytes	DMA transfer bytes to be transferred.
type	DMA transfer type.

10.7.26 status_t DMA_SubmitTransfer (dma_handle_t * *handle*, const dma_transfer_config_t * *config*, uint32_t *options*)

This function submits the DMA transfer request according to the transfer configuration structure.

Parameters

handle	DMA handle pointer.
config	Pointer to DMA transfer configuration structure.
options	Additional configurations for transfer. Use the defined dma_transfer_options_t type.

Return values

kStatus_DMA_Success	It indicates that the DMA submit transfer request succeeded.
kStatus_DMA_Busy	It indicates that the DMA is busy. Submit transfer request is not allowed.

Note

This function can't process multi transfer request.

10.7.27 static void DMA_StartTransfer (dma_handle_t * *handle*) [inline], [static]

This function enables the channel request. Call this function after submitting a transfer request.

Parameters

handle	DMA handle pointer.

Return values

kStatus_DMA_Success	It indicates that the DMA start transfer succeed.
kStatus_DMA_Busy	It indicates that the DMA has started a transfer.

10.7.28 static void DMA_StopTransfer (dma_handle_t * *handle*) [inline], [static]

This function disables the channel request to stop a DMA transfer. The transfer can be resumed by calling the DMA_StartTransfer.

Parameters

handle	DMA handle pointer.

10.7.29 void DMA_AbortTransfer (dma_handle_t * *handle*)

This function disables the channel request and clears all status bits. Submit another transfer after calling this API.

Parameters

handle	DMA handle pointer.

10.7.30 void DMA_HandleIRQ (dma_handle_t * *handle*)

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

Parameters

handle	DMA handle pointer.

Chapter 11
DMAMUX: Direct Memory Access Multiplexer Driver

11.1 Overview

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

11.2 Typical use case

11.2.1 DMAMUX Operation

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

Driver version

- #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  DMAMUX driver version 2.0.5.

DMAMUX Initialization and de-initialization

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

DMAMUX Channel Operation

- static void DMAMUX_EnableChannel (DMAMUX_Type ∗base, uint32_t channel)
  Enables the DMAMUX channel.
- static void DMAMUX_DisableChannel (DMAMUX_Type ∗base, uint32_t channel)
  Disables the DMAMUX channel.
- static void DMAMUX_SetSource (DMAMUX_Type ∗base, uint32_t channel, uint32_t source)
  Configures the DMAMUX channel source.
- static void DMAMUX_EnablePeriodTrigger (DMAMUX_Type ∗base, uint32_t channel)
  Enables the DMAMUX period trigger.
- static void DMAMUX_DisablePeriodTrigger (DMAMUX_Type ∗base, uint32_t channel)
  Disables the DMAMUX period trigger.

11.3 Macro Definition Documentation

11.3.1 #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

11.4 Function Documentation

11.4.1 void DMAMUX_Init (DMAMUX_Type * *base*)

This function ungates the DMAMUX clock.

Parameters

base	DMAMUX peripheral base address.

11.4.2 void DMAMUX_Deinit (DMAMUX_Type * *base*)

This function gates the DMAMUX clock.

Parameters

base	DMAMUX peripheral base address.

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

This function enables the DMAMUX channel.

Parameters

base	DMAMUX peripheral base address.
channel	DMAMUX channel number.

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

This function disables the DMAMUX channel.

Note

The user must disable the DMAMUX channel before configuring it.

Parameters

base	DMAMUX peripheral base address.

channel	DMAMUX channel number.

11.4.5 static void DMAMUX_SetSource (DMAMUX_Type ∗ *base*, uint32_t *channel*, uint32_t *source*) [inline], [static]

Parameters

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

11.4.6 static void DMAMUX_EnablePeriodTrigger (DMAMUX_Type ∗ *base*, uint32_t *channel*) [inline], [static]

This function enables the DMAMUX period trigger feature.

Parameters

base	DMAMUX peripheral base address.
channel	DMAMUX channel number.

11.4.7 static void DMAMUX_DisablePeriodTrigger (DMAMUX_Type ∗ *base*, uint32_t *channel*) [inline], [static]

This function disables the DMAMUX period trigger.

Parameters

base	DMAMUX peripheral base address.
channel	DMAMUX channel number.

Chapter 12
EWM: External Watchdog Monitor Driver

12.1 Overview

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

12.2 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.

12.3 Data Structure Documentation

12.3.1 struct ewm_config_t

Data structure for EWM configuration.

This structure is used to configure the EWM.

Data Fields

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

12.4 Macro Definition Documentation

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

12.5 Enumeration Type Documentation

12.5.1 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.

12.5.2 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.

12.6 Function Documentation

12.6.1 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

12.6.2 void EWM_Deinit (EWM_Type * *base*)

This function is used to shut down the EWM.

Parameters

base	EWM peripheral base address

12.6.3 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

12.6.4 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

12.6.5 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

12.6.6 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.

12.6.7 void EWM_Refresh (EWM_Type * *base*)

This function resets the EWM counter to zero.

Parameters

base	EWM peripheral base address

Chapter 13
C90TFS Flash Driver

13.1 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

13.2 Ftftx FLASH Driver

13.2.1 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.

13.2.2 Data Structure Documentation

13.2.2.1 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

13.2.2.2 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.

13.2.3 Macro Definition Documentation

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

Version 3.1.2.

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

Version 3.0.0.

13.2.4 Enumeration Type Documentation

13.2.4.1 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.

13.2.4.2 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.

13.2.5 Function Documentation

13.2.5.1 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_PartitionStatusUpdateFailure	Failed to update the partition status.

13.2.5.2 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.

13.2.5.3 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.

13.2.5.4 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.

13.2.5.5 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.

13.2.5.6 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.

13.2.5.7 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.

13.2.5.8 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.

13.2.5.9 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.

13.2.5.10 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_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.

13.2.5.11 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.

13.2.5.12 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_InvalidArgument	An invalid argument is provided.

13.2.5.13 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_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.

13.2.5.14 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.

13.2.5.15 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.

13.2.5.16 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 protect-Status;
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.

13.2.5.17 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_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_Unknown-Property	An unknown property tag.

13.2.5.18 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.

13.3 Ftftx CACHE Driver

13.3.1 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.

13.3.2 Data Structure Documentation

13.3.2.1 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

13.3.2.2 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

13.3.3 Enumeration Type Documentation

13.3.3.1 enum _ftfx_cache_ram_func_constants

Enumerator

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

13.3.4 Function Documentation

13.3.4.1 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_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.

13.3.4.2 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_InvalidArgument	Invalid argument is provided.
kStatus_FTFx_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.

13.3.4.3 status_t FTFx_CACHE_PflashSetPrefetchSpeculation (ftfx_prefetch_speculation_status_t * *speculationStatus*)

Parameters

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

Return values

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

13.3.4.4 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.

13.4 Ftftx FLEXNVM Driver

13.4.1 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 lengthInBytes, 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 whichProperty, uint32_t *value)
  Returns the desired flexnvm property.

13.4.2 Data Structure Documentation

13.4.2.1 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.

13.4.3 Enumeration Type Documentation

13.4.3.1 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.

13.4.4 Function Documentation

13.4.4.1 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_PartitionStatusUpdateFailure	Failed to update the partition status.

13.4.4.2 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_InvalidArgument	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.

13.4.4.3 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.

13.4.4.4 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_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.

13.4.4.5 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.
eepromDataSizeCode	Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
flexnvmPartitionCode	Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

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

kStatus_FTFx_-CommandFailure	Run-time error during command execution.

13.4.4.6 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.

13.4.4.7 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.

13.4.4.8 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.

13.4.4.9 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_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.

13.4.4.10 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_InvalidArgument	An invalid argument is provided.

13.4.4.11 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_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.

13.4.4.12 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.

13.4.4.13 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.

13.4.4.14 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.

13.4.4.15 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.

13.4.4.16 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_CommandNotSupported	Flash API is not supported.

13.4.4.17 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_UnknownProperty	An unknown property tag.

13.5 ftfx feature

13.5.1 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.

13.5.2 Macro Definition Documentation

13.5.2.1 #define FTFx_DRIVER_IS_FLASH_RESIDENT 1U

Used for the flash resident application.

13.5.2.2 #define FTFx_DRIVER_IS_EXPORTED 0U

Used for the MCUXpresso SDK application.

13.5.2.3 #define FTFx_FLASH1_HAS_PROT_CONTROL (0U)

13.5.2.4 #define FTFx_FLASH1_HAS_XACC_CONTROL (0U)

13.5.3 ftfx adapter

13.6 ftfx controller

13.6.1 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.

13.6.2 Data Structure Documentation

13.6.2.1 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

13.6.2.2 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

13.6.2.3 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

13.6.2.4 struct ftfx_ifr_desc_t

13.6.2.5 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

13.6.3 Macro Definition Documentation

13.6.3.1 #define kStatusGroupGeneric 0

13.6.4 Enumeration Type Documentation

13.6.4.1 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.

13.6.4.2 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.

13.6.4.3 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.

13.6.4.4 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.

13.6.4.5 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.

13.6.4.6 enum ftfx_security_state_t

Enumerator

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

13.6.4.7 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 EEPROM.

13.6.4.8 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.

13.6.5 Function Documentation

13.6.5.1 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.

13.6.5.2 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_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.

13.6.5.3 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_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.

13.6.5.4 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_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.

13.6.5.5 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_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 the command execution.

13.6.5.6 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.

13.6.5.7 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.

13.6.5.8 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.

13.6.5.9 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.

13.6.5.10 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_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.

13.6.5.11 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_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.

13.6.5.12 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.

13.6.5.13 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.

13.6.5.14 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_Invalid-Argument	An invalid argument is provided.

13.6.5.15 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_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.

13.6.6 ftfx utilities

13.6.6.1 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.

13.6.6.2 Macro Definition Documentation

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

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

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

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

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

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

Chapter 14
GPIO: General-Purpose Input/Output Driver

14.1 Overview

Modules

- FGPIO Driver
- GPIO Driver

Data Structures

- struct gpio_pin_config_t
  The GPIO pin configuration structure. More...

Macros

- #define GPIO_FIT_REG(value) ((uint8_t)(value))
  For some platforms with 8-bit register width, cast the type to uint8_t.

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  GPIO direction definition.
- enum gpio_checker_attribute_t {
  kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW,
  kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW,
  kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW,
  kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW,
  kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW,
  kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW,
  kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR,
  kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN,
  kGPIO_IgnoreAttributeCheck = 0x80U }
  GPIO checker attribute.

Driver version

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

14.2 Data Structure Documentation

14.2.1 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.

14.3 Macro Definition Documentation

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

14.4 Enumeration Type Documentation

14.4.1 enum gpio_pin_direction_t

Enumerator

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

14.4.2 enum gpio_checker_attribute_t

Enumerator

kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write.
kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW User nonsecure:Read; User Secure-:Read+Write; Privileged Secure:Read+Write.
kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW User nonsecure:None; User Secure-:Read+Write; Privileged Secure:Read+Write.
kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW User nonsecure:Read; User Secure-:Read; Privileged Secure:Read+Write.
kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW User nonsecure:None; User Secure-:Read; Privileged Secure:Read+Write.
kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW User nonsecure:None; User Secure-:None; Privileged Secure:Read+Write.
kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR User nonsecure:None; User Secure-:None; Privileged Secure:Read.

kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN User nonsecure:None; User Secure-:None; Privileged Secure:None.

kGPIO_IgnoreAttributeCheck Ignores the attribute check.

14.5 GPIO Driver

14.5.1 Overview

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

14.5.2 Typical use case

14.5.2.1 Output Operation

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

14.5.2.2 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.
- void GPIO_CheckAttributeBytes (GPIO_Type *base, gpio_checker_attribute_t attribute)
  brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes.

14.5.3 Function Documentation

14.5.3.1 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

14.5.3.2 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.

14.5.3.3 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

14.5.3.4 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

14.5.3.5 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

14.5.3.6 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.

14.5.3.7 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.

14.5.3.8 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

14.5.3.9 void GPIO_CheckAttributeBytes (GPIO_Type ∗ *base,* gpio_checker_attribute_t *attribute*)

Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

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

14.6 FGPIO Driver

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.

14.6.1 Typical use case

14.6.1.1 Output Operation

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

14.6.1.2 Input Operation

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

Chapter 15
I2C: Inter-Integrated Circuit Driver

15.1 Overview

Modules

- I2C CMSIS Driver
- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver

15.2 I2C Driver

15.2.1 Overview

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

The I2C driver includes functional APIs and transactional APIs.

Functional APIs target the low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires knowing the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs target the high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions I2C_MasterTransferNonBlocking() set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

15.2.2 Typical use case

15.2.2.1 Master Operation in functional method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/i2c

15.2.2.2 Master Operation in interrupt transactional method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/i2c

15.2.2.3 Master Operation in DMA transactional method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/i2c

15.2.2.4 Slave Operation in functional method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/i2c

15.2.2.5 Slave Operation in interrupt transactional method

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/i2c

Data Structures

- struct i2c_master_config_t
  I2C master user configuration. More...
- struct i2c_slave_config_t
  I2C slave user configuration. More...
- struct i2c_master_transfer_t
  I2C master transfer structure. More...
- struct i2c_master_handle_t
  I2C master handle structure. More...
- struct i2c_slave_transfer_t
  I2C slave transfer structure. More...
- struct i2c_slave_handle_t
  I2C slave handle structure. More...

Macros

- #define I2C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.
- #define I2C_MASTER_FACK_CONTROL 0U /∗ Default defines to zero means master will send ack automatically. ∗/
  Mater Fast ack control, control if master needs to manually write ack, this is used to low the speed of transfer for SoCs with feature FSL_FEATURE_I2C_HAS_DOUBLE_BUFFERING.

Typedefs

- typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t status, void ∗userData)
  I2C master transfer callback typedef.
- typedef void(∗ i2c_slave_transfer_callback_t)(I2C_Type ∗base, i2c_slave_transfer_t ∗xfer, void ∗userData)
  I2C slave transfer callback typedef.

Enumerations

- enum {
  kStatus_I2C_Busy = MAKE_STATUS(kStatusGroup_I2C, 0),
  kStatus_I2C_Idle = MAKE_STATUS(kStatusGroup_I2C, 1),
  kStatus_I2C_Nak = MAKE_STATUS(kStatusGroup_I2C, 2),
  kStatus_I2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_I2C, 3),
  kStatus_I2C_Timeout = MAKE_STATUS(kStatusGroup_I2C, 4),
  kStatus_I2C_Addr_Nak = MAKE_STATUS(kStatusGroup_I2C, 5) }
  I2C status return codes.

- enum _i2c_flags {
  kI2C_ReceiveNakFlag = I2C_S_RXAK_MASK,
  kI2C_IntPendingFlag = I2C_S_IICIF_MASK,
  kI2C_TransferDirectionFlag = I2C_S_SRW_MASK,
  kI2C_RangeAddressMatchFlag = I2C_S_RAM_MASK,
  kI2C_ArbitrationLostFlag = I2C_S_ARBL_MASK,
  kI2C_BusBusyFlag = I2C_S_BUSY_MASK,
  kI2C_AddressMatchFlag = I2C_S_IAAS_MASK,
  kI2C_TransferCompleteFlag = I2C_S_TCF_MASK,
  kI2C_StopDetectFlag = I2C_FLT_STOPF_MASK << 8,
  kI2C_StartDetectFlag = I2C_FLT_STARTF_MASK << 8 }

  I2C peripheral flags.
- enum _i2c_interrupt_enable {
  kI2C_GlobalInterruptEnable = I2C_C1_IICIE_MASK,
  kI2C_StartStopDetectInterruptEnable = I2C_FLT_SSIE_MASK }

  I2C feature interrupt source.
- enum i2c_direction_t {
  kI2C_Write = 0x0U,
  kI2C_Read = 0x1U }

  The direction of master and slave transfers.
- enum i2c_slave_address_mode_t {
  kI2C_Address7bit = 0x0U,
  kI2C_RangeMatch = 0X2U }

  Addressing mode.
- enum _i2c_master_transfer_flags {
  kI2C_TransferDefaultFlag = 0x0U,
  kI2C_TransferNoStartFlag = 0x1U,
  kI2C_TransferRepeatedStartFlag = 0x2U,
  kI2C_TransferNoStopFlag = 0x4U }

  I2C transfer control flag.
- enum i2c_slave_transfer_event_t {
  kI2C_SlaveAddressMatchEvent = 0x01U,
  kI2C_SlaveTransmitEvent = 0x02U,
  kI2C_SlaveReceiveEvent = 0x04U,
  kI2C_SlaveTransmitAckEvent = 0x08U,
  kI2C_SlaveStartEvent = 0x10U,
  kI2C_SlaveCompletionEvent = 0x20U,
  kI2C_SlaveGenaralcallEvent = 0x40U,
  kI2C_SlaveAllEvents }

  Set of events sent to the callback for nonblocking slave transfers.
- enum { kClearFlags = kI2C_ArbitrationLostFlag | kI2C_IntPendingFlag | kI2C_StartDetectFlag | kI2C_StopDetectFlag }

  Common sets of flags used by the driver.

Driver version

- #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))
  I2C driver version.

Initialization and deinitialization

- void I2C_MasterInit (I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes the I2C peripheral.
- void I2C_SlaveInit (I2C_Type ∗base, const i2c_slave_config_t ∗slaveConfig, uint32_t srcClock_Hz)
  Initializes the I2C peripheral.
- void I2C_MasterDeinit (I2C_Type ∗base)
  De-initializes the I2C master peripheral.
- void I2C_SlaveDeinit (I2C_Type ∗base)
  De-initializes the I2C slave peripheral.
- uint32_t I2C_GetInstance (I2C_Type ∗base)
  Get instance number for I2C module.
- void I2C_MasterGetDefaultConfig (i2c_master_config_t ∗masterConfig)
  Sets the I2C master configuration structure to default values.
- void I2C_SlaveGetDefaultConfig (i2c_slave_config_t ∗slaveConfig)
  Sets the I2C slave configuration structure to default values.
- static void I2C_Enable (I2C_Type ∗base, bool enable)
  Enables or disables the I2C peripheral operation.

Status

- uint32_t I2C_MasterGetStatusFlags (I2C_Type ∗base)
  Gets the I2C status flags.
- static uint32_t I2C_SlaveGetStatusFlags (I2C_Type ∗base)
  Gets the I2C status flags.
- static void I2C_MasterClearStatusFlags (I2C_Type ∗base, uint32_t statusMask)
  Clears the I2C status flag state.
- static void I2C_SlaveClearStatusFlags (I2C_Type ∗base, uint32_t statusMask)
  Clears the I2C status flag state.

Interrupts

- void I2C_EnableInterrupts (I2C_Type ∗base, uint32_t mask)
  Enables I2C interrupt requests.
- void I2C_DisableInterrupts (I2C_Type ∗base, uint32_t mask)
  Disables I2C interrupt requests.

DMA Control

- static void I2C_EnableDMA (I2C_Type *base, bool enable)
  Enables/disables the I2C DMA interrupt.
- static uint32_t I2C_GetDataRegAddr (I2C_Type *base)
  Gets the I2C tx/rx data register address.

Bus Operations

- void I2C_MasterSetBaudRate (I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the I2C master transfer baud rate.
- status_t I2C_MasterStart (I2C_Type *base, uint8_t address, i2c_direction_t direction)
  Sends a START on the I2C bus.
- status_t I2C_MasterStop (I2C_Type *base)
  Sends a STOP signal on the I2C bus.
- status_t I2C_MasterRepeatedStart (I2C_Type *base, uint8_t address, i2c_direction_t direction)
  Sends a REPEATED START on the I2C bus.
- status_t I2C_MasterWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize, uint32_t flags)
  Performs a polling send transaction on the I2C bus.
- status_t I2C_MasterReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize, uint32_t flags)
  Performs a polling receive transaction on the I2C bus.
- status_t I2C_SlaveWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize)
  Performs a polling send transaction on the I2C bus.
- status_t I2C_SlaveReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize)
  Performs a polling receive transaction on the I2C bus.
- status_t I2C_MasterTransferBlocking (I2C_Type *base, i2c_master_transfer_t *xfer)
  Performs a master polling transfer on the I2C bus.

Transactional

- void I2C_MasterTransferCreateHandle (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_callback_t callback, void *userData)
  Initializes the I2C handle which is used in transactional functions.
- status_t I2C_MasterTransferNonBlocking (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_t *xfer)
  Performs a master interrupt non-blocking transfer on the I2C bus.
- status_t I2C_MasterTransferGetCount (I2C_Type *base, i2c_master_handle_t *handle, size_t *count)
  Gets the master transfer status during a interrupt non-blocking transfer.
- status_t I2C_MasterTransferAbort (I2C_Type *base, i2c_master_handle_t *handle)
  Aborts an interrupt non-blocking transfer early.
- void I2C_MasterTransferHandleIRQ (I2C_Type *base, void *i2cHandle)
  Master interrupt handler.
- void I2C_SlaveTransferCreateHandle (I2C_Type *base, i2c_slave_handle_t *handle, i2c_slave_transfer_callback_t callback, void *userData)
  Initializes the I2C handle which is used in transactional functions.

- status_t I2C_SlaveTransferNonBlocking (I2C_Type ∗base, i2c_slave_handle_t ∗handle, uint32_t eventMask)
  Starts accepting slave transfers.
- void I2C_SlaveTransferAbort (I2C_Type ∗base, i2c_slave_handle_t ∗handle)
  Aborts the slave transfer.
- status_t I2C_SlaveTransferGetCount (I2C_Type ∗base, i2c_slave_handle_t ∗handle, size_t ∗count)
  Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.
- void I2C_SlaveTransferHandleIRQ (I2C_Type ∗base, void ∗i2cHandle)
  Slave interrupt handler.

15.2.3 Data Structure Documentation

15.2.3.1 struct i2c_master_config_t

Data Fields

- bool enableMaster
  Enables the I2C peripheral at initialization time.
- bool enableStopHold
  Controls the stop hold enable.
- uint32_t baudRate_Bps
  Baud rate configuration of I2C peripheral.
- uint8_t glitchFilterWidth
  Controls the width of the glitch.

Field Documentation

(1) bool i2c_master_config_t::enableMaster

(2) bool i2c_master_config_t::enableStopHold

(3) uint32_t i2c_master_config_t::baudRate_Bps

(4) uint8_t i2c_master_config_t::glitchFilterWidth

15.2.3.2 struct i2c_slave_config_t

Data Fields

- bool enableSlave
  Enables the I2C peripheral at initialization time.
- bool enableGeneralCall
  Enables the general call addressing mode.
- bool enableWakeUp
  Enables/disables waking up MCU from low-power mode.
- bool enableBaudRateCtl
  Enables/disables independent slave baud rate on SCL in very fast I2C modes.
- uint16_t slaveAddress
  A slave address configuration.

- uint16_t upperAddress
  A maximum boundary slave address used in a range matching mode.
- i2c_slave_address_mode_t addressingMode
  An addressing mode configuration of i2c_slave_address_mode_config_t.
- uint32_t sclStopHoldTime_ns
  the delay from the rising edge of SCL (I2C clock) to the rising edge of SDA (I2C data) while SCL is high (stop condition), SDA hold time and SCL start hold time are also configured according to the SCL stop hold time.

Field Documentation

(1) bool i2c_slave_config_t::enableSlave

(2) bool i2c_slave_config_t::enableGeneralCall

(3) bool i2c_slave_config_t::enableWakeUp

(4) bool i2c_slave_config_t::enableBaudRateCtl

(5) uint16_t i2c_slave_config_t::slaveAddress

(6) uint16_t i2c_slave_config_t::upperAddress

(7) i2c_slave_address_mode_t i2c_slave_config_t::addressingMode

(8) uint32_t i2c_slave_config_t::sclStopHoldTime_ns

15.2.3.3 struct i2c_master_transfer_t

Data Fields

- uint32_t flags
  A transfer flag which controls the transfer.
- uint8_t slaveAddress
  7-bit slave address.
- i2c_direction_t direction
  A transfer direction, read or write.
- uint32_t subaddress
  A sub address.
- uint8_t subaddressSize
  A size of the command buffer.
- uint8_t *volatile data
  A transfer buffer.
- volatile size_t dataSize
  A transfer size.

Field Documentation

(1) uint32_t i2c_master_transfer_t::flags

(2) uint8_t i2c_master_transfer_t::slaveAddress

(3) i2c_direction_t i2c_master_transfer_t::direction

(4) uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

(5) uint8_t i2c_master_transfer_t::subaddressSize

(6) uint8_t* volatile i2c_master_transfer_t::data

(7) volatile size_t i2c_master_transfer_t::dataSize

15.2.3.4 struct _i2c_master_handle

I2C master handle typedef.

Data Fields

- i2c_master_transfer_t transfer
  I2C master transfer copy.
- size_t transferSize
  Total bytes to be transferred.
- uint8_t state
  A transfer state maintained during transfer.
- i2c_master_transfer_callback_t completionCallback
  A callback function called when the transfer is finished.
- void * userData
  A callback parameter passed to the callback function.

Field Documentation

(1) i2c_master_transfer_t i2c_master_handle_t::transfer

(2) size_t i2c_master_handle_t::transferSize

(3) uint8_t i2c_master_handle_t::state

(4) i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

(5) void* i2c_master_handle_t::userData

15.2.3.5 struct i2c_slave_transfer_t

Data Fields

- i2c_slave_transfer_event_t event
  A reason that the callback is invoked.
- uint8_t *volatile data

A transfer buffer.
- volatile size_t dataSize
  A transfer size.
- status_t completionStatus
  Success or error code describing how the transfer completed.
- size_t transferredCount
  A number of bytes actually transferred since the start or since the last repeated start.

Field Documentation

(1) i2c_slave_transfer_event_t i2c_slave_transfer_t::event

(2) uint8_t∗ volatile i2c_slave_transfer_t::data

(3) volatile size_t i2c_slave_transfer_t::dataSize

(4) status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

(5) size_t i2c_slave_transfer_t::transferredCount

15.2.3.6 struct _i2c_slave_handle

I2C slave handle typedef.

Data Fields

- volatile bool isBusy
  Indicates whether a transfer is busy.
- i2c_slave_transfer_t transfer
  I2C slave transfer copy.
- uint32_t eventMask
  A mask of enabled events.
- i2c_slave_transfer_callback_t callback
  A callback function called at the transfer event.
- void ∗ userData
  A callback parameter passed to the callback.

Field Documentation

(1) volatile bool i2c_slave_handle_t::isBusy

(2) i2c_slave_transfer_t i2c_slave_handle_t::transfer

(3) uint32_t i2c_slave_handle_t::eventMask

(4) i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

(5) void∗ i2c_slave_handle_t::userData

15.2.4 Macro Definition Documentation

15.2.4.1 #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))

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

15.2.5 Typedef Documentation

15.2.5.1 typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *userData)

15.2.5.2 typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, i2c_slave_transfer_t *xfer, void *userData)

15.2.6 Enumeration Type Documentation

15.2.6.1 anonymous enum

Enumerator

kStatus_I2C_Busy I2C is busy with current transfer.
kStatus_I2C_Idle Bus is Idle.
kStatus_I2C_Nak NAK received during transfer.
kStatus_I2C_ArbitrationLost Arbitration lost during transfer.
kStatus_I2C_Timeout Timeout polling status flags.
kStatus_I2C_Addr_Nak NAK received during the address probe.

15.2.6.2 enum _i2c_flags

Note

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

Enumerator

kI2C_ReceiveNakFlag I2C receive NAK flag.
kI2C_IntPendingFlag I2C interrupt pending flag. This flag can be cleared.
kI2C_TransferDirectionFlag I2C transfer direction flag.
kI2C_RangeAddressMatchFlag I2C range address match flag.
kI2C_ArbitrationLostFlag I2C arbitration lost flag. This flag can be cleared.
kI2C_BusBusyFlag I2C bus busy flag.
kI2C_AddressMatchFlag I2C address match flag.
kI2C_TransferCompleteFlag I2C transfer complete flag.

kI2C_StopDetectFlag I2C stop detect flag. This flag can be cleared.
kI2C_StartDetectFlag I2C start detect flag. This flag can be cleared.

15.2.6.3 enum _i2c_interrupt_enable

Enumerator

kI2C_GlobalInterruptEnable I2C global interrupt.
kI2C_StartStopDetectInterruptEnable I2C start&stop detect interrupt.

15.2.6.4 enum i2c_direction_t

Enumerator

kI2C_Write Master transmits to the slave.
kI2C_Read Master receives from the slave.

15.2.6.5 enum i2c_slave_address_mode_t

Enumerator

kI2C_Address7bit 7-bit addressing mode.
kI2C_RangeMatch Range address match addressing mode.

15.2.6.6 enum _i2c_master_transfer_flags

Enumerator

kI2C_TransferDefaultFlag A transfer starts with a start signal, stops with a stop signal.
kI2C_TransferNoStartFlag A transfer starts without a start signal, only support write only or write+read with no start flag, do not support read only with no start flag.
kI2C_TransferRepeatedStartFlag A transfer starts with a repeated start signal.
kI2C_TransferNoStopFlag A transfer ends without a stop signal.

15.2.6.7 enum i2c_slave_transfer_event_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I2C_SlaveTransferNonBlocking() to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

kI2C_SlaveAddressMatchEvent Received the slave address after a start or repeated start.
kI2C_SlaveTransmitEvent A callback is requested to provide data to transmit (slave-transmitter role).
kI2C_SlaveReceiveEvent A callback is requested to provide a buffer in which to place received data (slave-receiver role).
kI2C_SlaveTransmitAckEvent A callback needs to either transmit an ACK or NACK.
kI2C_SlaveStartEvent A start/repeated start was detected.
kI2C_SlaveCompletionEvent A stop was detected or finished transfer, completing the transfer.
kI2C_SlaveGenaralcallEvent Received the general call address after a start or repeated start.
kI2C_SlaveAllEvents A bit mask of all available events.

15.2.6.8 anonymous enum

Enumerator

kClearFlags All flags which are cleared by the driver upon starting a transfer.

15.2.7 Function Documentation

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

Call this API to ungate the I2C clock and configure the I2C with master configuration.

Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the I2C_MasterGetDefaultConfig(). After calling this API, the master is ready to transfer. This is an example.

```
* i2c_master_config_t config = {
* .enableMaster = true,
* .enableStopHold = false,
* .highDrive = false,
* .baudRate_Bps = 100000,
* .glitchFilterWidth = 0
* };
* I2C_MasterInit(I2C0, &config, 12000000U);
*
```

Parameters

base	I2C base pointer
masterConfig	A pointer to the master configuration structure
srcClock_Hz	I2C peripheral clock frequency in Hz

15.2.7.2 void I2C_SlaveInit (I2C_Type * *base*, const i2c_slave_config_t * *slaveConfig*, uint32_t *srcClock_Hz*)

Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by I2C_SlaveGetDefaultConfig() or it can be custom filled by the user. This is an example.

```
* i2c_slave_config_t config = {
* .enableSlave = true,
* .enableGeneralCall = false,
* .addressingMode = kI2C_Address7bit,
* .slaveAddress = 0x1DU,
* .enableWakeUp = false,
* .enablehighDrive = false,
* .enableBaudRateCtl = false,
* .sclStopHoldTime_ns = 4000
* };
* I2C_SlaveInit(I2C0, &config, 12000000U);
*
```

Parameters

base	I2C base pointer
slaveConfig	A pointer to the slave configuration structure
srcClock_Hz	I2C peripheral clock frequency in Hz

15.2.7.3 void I2C_MasterDeinit (I2C_Type * *base*)

Call this API to gate the I2C clock. The I2C master module can't work unless the I2C_MasterInit is called.

Parameters

base	I2C base pointer

15.2.7.4 void I2C_SlaveDeinit (I2C_Type * *base*)

Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C_SlaveInit is called to enable the clock.

Parameters

base	I2C base pointer

15.2.7.5 uint32_t I2C_GetInstance (I2C_Type * *base*)

Parameters

base	I2C peripheral base address.

15.2.7.6 void I2C_MasterGetDefaultConfig (i2c_master_config_t * *masterConfig*)

The purpose of this API is to get the configuration structure initialized for use in the I2C_MasterConfigure(). Use the initialized structure unchanged in the I2C_MasterConfigure() or modify the structure before calling the I2C_MasterConfigure(). This is an example.

```
* i2c_master_config_t config;
* I2C_MasterGetDefaultConfig(&config);
*
```

Parameters

masterConfig	A pointer to the master configuration structure.

15.2.7.7 void I2C_SlaveGetDefaultConfig (i2c_slave_config_t * *slaveConfig*)

The purpose of this API is to get the configuration structure initialized for use in the I2C_SlaveConfigure(). Modify fields of the structure before calling the I2C_SlaveConfigure(). This is an example.

```
* i2c_slave_config_t config;
* I2C_SlaveGetDefaultConfig(&config);
*
```

Parameters

slaveConfig	A pointer to the slave configuration structure.

15.2.7.8 static void I2C_Enable (I2C_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	I2C base pointer
enable	Pass true to enable and false to disable the module.

15.2.7.9 uint32_t I2C_MasterGetStatusFlags (I2C_Type * *base*)

Parameters

base	I2C base pointer

Returns

status flag, use status flag to AND _i2c_flags to get the related status.

15.2.7.10 static uint32_t I2C_SlaveGetStatusFlags (I2C_Type * *base*) [inline], [static]

Parameters

base	I2C base pointer

Returns

status flag, use status flag to AND _i2c_flags to get the related status.

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

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag.

Parameters

base	I2C base pointer
statusMask	The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values: • kI2C_StartDetectFlag (if available) • kI2C_StopDetectFlag (if available) • kI2C_ArbitrationLostFlag • kI2C_IntPendingFlagFlag

15.2.7.12 static void I2C_SlaveClearStatusFlags (I2C_Type ∗ *base,* uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag

Parameters

base	I2C base pointer
statusMask	The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values: • kI2C_StartDetectFlag (if available) • kI2C_StopDetectFlag (if available) • kI2C_ArbitrationLostFlag • kI2C_IntPendingFlagFlag

15.2.7.13 void I2C_EnableInterrupts (I2C_Type ∗ *base,* uint32_t *mask*)

Parameters

base	I2C base pointer
mask	interrupt source The parameter can be combination of the following source if defined: • kI2C_GlobalInterruptEnable • kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable • kI2C_SdaTimeoutInterruptEnable

15.2.7.14 void I2C_DisableInterrupts (I2C_Type ∗ *base,* uint32_t *mask*)

Parameters

base	I2C base pointer
mask	interrupt source The parameter can be combination of the following source if defined: • kI2C_GlobalInterruptEnable • kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable • kI2C_SdaTimeoutInterruptEnable

15.2.7.15 static void I2C_EnableDMA (I2C_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	I2C base pointer
enable	true to enable, false to disable

15.2.7.16 static uint32_t I2C_GetDataRegAddr (I2C_Type * *base*) [inline], [static]

This API is used to provide a transfer address for I2C DMA transfer configuration.

Parameters

base	I2C base pointer

Returns

data register address

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

Parameters

base	I2C base pointer
baudRate_Bps	the baud rate value in bps
srcClock_Hz	Source clock

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

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

Parameters

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

Return values

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

15.2.7.19 status_t I2C_MasterStop (I2C_Type * *base*)

Return values

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

15.2.7.20 status_t I2C_MasterRepeatedStart (I2C_Type * *base*, uint8_t *address*, i2c_direction_t *direction*)

Parameters

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

Return values

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

15.2.7.21 status_t I2C_MasterWriteBlocking (I2C_Type * *base*, const uint8_t * *txBuff*, size_t *txSize*, uint32_t *flags*)

Parameters

base	The I2C peripheral base pointer.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.
flags	Transfer control flag to decide whether need to send a stop, use kI2C_Transfer-DefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

15.2.7.22 status_t I2C_MasterReadBlocking (I2C_Type * *base*, uint8_t * *rxBuff*, size_t *rxSize*, uint32_t *flags*)

Note

The I2C_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

Parameters

base	I2C peripheral base pointer.
rxBuff	The pointer to the data to store the received data.
rxSize	The length in bytes of the data to be received.
flags	Transfer control flag to decide whether need to send a stop, use kI2C_Transfer-DefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Timeout	Send stop signal failed, timeout.

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

Parameters

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

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

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

Parameters

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

Return values

kStatus_Success	Successfully complete data receive.
kStatus_I2C_Timeout	Wait status flag timeout.

15.2.7.25 status_t I2C_MasterTransferBlocking (I2C_Type * *base*, i2c_master_transfer_t * *xfer*)

Note

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

Parameters

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

Return values

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

15.2.7.26 void I2C_MasterTransferCreateHandle (I2C_Type * *base*, i2c_master_handle_t * *handle*, i2c_master_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure to store the transfer state.
callback	pointer to user callback function.
userData	user parameter passed to the callback function.

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

Note

Calling the API returns immediately after transfer initiates. The user needs to call I2C_MasterGet-TransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_I2C_Busy, the transfer is finished.

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure which stores the transfer state.
xfer	pointer to i2c_master_transfer_t structure.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_I2C_Busy	Previous transmission still not finished.
kStatus_I2C_Timeout	Transfer error, wait signal timeout.

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

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure which stores the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

15.2.7.29 status_t I2C_MasterTransferAbort (I2C_Type ∗ *base*, i2c_master_handle_t ∗ *handle*)

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure which stores the transfer state

Return values

kStatus_I2C_Timeout	Timeout during polling flag.
kStatus_Success	Successfully abort the transfer.

15.2.7.30 void I2C_MasterTransferHandleIRQ (I2C_Type * *base*, void * *i2cHandle*)

Parameters

base	I2C base pointer.
i2cHandle	pointer to i2c_master_handle_t structure.

15.2.7.31 void I2C_SlaveTransferCreateHandle (I2C_Type * *base*, i2c_slave_handle_t * *handle*, i2c_slave_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_slave_handle_t structure to store the transfer state.
callback	pointer to user callback function.
userData	user parameter passed to the callback function.

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

Call this API after calling the I2C_SlaveInit() and I2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to I2C_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 i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

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

Return values

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

15.2.7.33 void I2C_SlaveTransferAbort (I2C_Type ∗ *base,* i2c_slave_handle_t ∗ *handle*)

Note

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

Parameters

base	I2C base pointer.
handle	pointer to i2c_slave_handle_t structure which stores the transfer state.

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

Parameters

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

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

15.2.7.35 void I2C_SlaveTransferHandleIRQ (I2C_Type * *base*, void * *i2cHandle*)

Parameters

base	I2C base pointer.
i2cHandle	pointer to i2c_slave_handle_t structure which stores the transfer state

15.3 I2C DMA Driver

15.3.1 Overview

Data Structures

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

Typedefs

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

Driver version

- #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))
  I2C DMA driver version.

I2C Block DMA Transfer Operation

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

15.3.2 Data Structure Documentation

15.3.2.1 struct _i2c_master_dma_handle

Retry times for waiting flag.

I2C master DMA handle typedef.

Data Fields

- i2c_master_transfer_t transfer
  I2C master transfer struct.
- size_t transferSize
  Total bytes to be transferred.
- uint8_t state
  I2C master transfer status.
- dma_handle_t * dmaHandle
  The DMA handler used.
- i2c_master_dma_transfer_callback_t completionCallback
  A callback function called after the DMA transfer finished.
- void * userData
  A callback parameter passed to the callback function.

Field Documentation

(1) i2c_master_transfer_t i2c_master_dma_handle_t::transfer

(2) size_t i2c_master_dma_handle_t::transferSize

(3) uint8_t i2c_master_dma_handle_t::state

(4) dma_handle_t* i2c_master_dma_handle_t::dmaHandle

(5) i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completionCallback

(6) void* i2c_master_dma_handle_t::userData

15.3.3 Macro Definition Documentation

15.3.3.1 #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))

15.3.4 Typedef Documentation

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

15.3.5 Function Documentation

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

Parameters

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

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

Parameters

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

Return values

kStatus_Success	Successfully completes the data transmission.
kStatus_I2C_Busy	A previous transmission is still not finished.
kStatus_I2C_Timeout	A transfer error, waits for the signal timeout.
kStatus_I2C_Arbitration-Lost	A transfer error, arbitration lost.
kStataus_I2C_Nak	A transfer error, receives NAK during transfer.

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

Parameters

base	I2C peripheral base address
handle	A pointer to the i2c_master_dma_handle_t structure

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

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

Parameters

base	I2C peripheral base address
handle	A pointer to the i2c_master_dma_handle_t structure.

15.4 I2C FreeRTOS Driver

15.4.1 Overview

Driver version

- #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))
  I2C FreeRTOS driver version 2.0.9.

I2C RTOS Operation

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

15.4.2 Macro Definition Documentation

15.4.2.1 #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))

15.4.3 Function Documentation

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

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

Parameters

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

Returns

status of the operation.

15.4.3.2 status_t I2C_RTOS_Deinit (i2c_rtos_handle_t * *handle*)

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

Parameters

handle	The RTOS I2C handle.

15.4.3.3 status_t I2C_RTOS_Transfer (i2c_rtos_handle_t ∗ *handle*, i2c_master_transfer_t ∗ *transfer*)

This function performs the I2C transfer according to the data given in the transfer structure.

Parameters

handle	The RTOS I2C handle.
transfer	A structure specifying the transfer parameters.

Returns

status of the operation.

15.5 I2C CMSIS Driver

This section describes the programming interface of the I2C Cortex Microcontroller Software Interface Standard (CMSIS) driver. This driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage methord see http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The I2C CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

15.5.1 I2C CMSIS Driver

15.5.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}
/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
Driver_I2C0.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

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

15.5.1.2 Master Operation in DMA transactional method

```
    void I2C_MasterSignalEvent_t(uint32_t event)
    {
        /*  Transfer done */
        if (event == ARM_I2C_EVENT_TRANSFER_DONE)
        {
            g_MasterCompletionFlag = true;
        }
    }

/* Init DMAMUX and DMA/EDMA. */
    DMAMUX_Init(EXAMPLE_I2C_DMAMUX_BASEADDR)
```

```
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
    DMA_Init(EXAMPLE_I2C_DMA_BASEADDR);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
    edma_config_t edmaConfig;

    EDMA_GetDefaultConfig(&edmaConfig);
    EDMA_Init(EXAMPLE_I2C_DMA_BASEADDR, &edmaConfig);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

    /*Init I2C0*/
    Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

    Driver_I2C0.PowerControl(ARM_POWER_FULL);

    /*config transmit speed*/
    Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

    /*start transfer*/
    Driver_I2C0.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

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

15.5.1.3 Slave Operation in interrupt transactional method

```
void I2C_SlaveSignalEvent_t(uint32_t event)
{
    /*  Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_SlaveCompletionFlag = true;
    }
}

/*Init I2C1*/
Driver_I2C1.Initialize(I2C_SlaveSignalEvent_t);

Driver_I2C1.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
Driver_I2C1.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
Driver_I2C1.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

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

Chapter 16
IRTC: IRTC Driver

16.1 Overview

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

Data Structures

- struct irtc_datetime_t
  Structure is used to hold the date and time. More...
- struct irtc_daylight_time_t
  Structure is used to hold the daylight saving time. More...
- struct irtc_tamper_config_t
  Structure is used to define the parameters to configure a RTC tamper event. More...
- struct irtc_config_t
  RTC config structure. More...

Enumerations

- enum irtc_filter_clock_source_t {
  kIRTC_32K = 0x0U,
  kIRTC_512 = 0x1U,
  kIRTC_128 = 0x2U,
  kIRTC_64 = 0x3U,
  kIRTC_16 = 0x4U,
  kIRTC_8 = 0x5U,
  kIRTC_4 = 0x6U,
  kIRTC_2 = 0x7U }
  IRTC filter clock source options.
- enum irtc_tamper_pins_t {
  kIRTC_Tamper_0 = 0U,
  kIRTC_Tamper_1,
  kIRTC_Tamper_2,
  kIRTC_Tamper_3 }
  IRTC Tamper pins.
- enum irtc_interrupt_enable_t {

kIRTC_TamperInterruptEnable = RTC_IER_TAMPER_IE_MASK,
kIRTC_AlarmInterruptEnable = RTC_IER_ALM_IE_MASK,
kIRTC_DayInterruptEnable = RTC_IER_DAY_IE_MASK,
kIRTC_HourInterruptEnable = RTC_IER_HOUR_IE_MASK,
kIRTC_MinInterruptEnable = RTC_IER_MIN_IE_MASK,
kIRTC_1hzInterruptEnable = RTC_IER_IE_1HZ_MASK,
kIRTC_2hzInterruptEnable = RTC_IER_IE_2HZ_MASK,
kIRTC_4hzInterruptEnable = RTC_IER_IE_4HZ_MASK,
kIRTC_8hzInterruptEnable = RTC_IER_IE_8HZ_MASK,
kIRTC_16hzInterruptEnable = RTC_IER_IE_16HZ_MASK,
kIRTC_32hzInterruptEnable = RTC_IER_IE_32HZ_MASK,
kIRTC_64hzInterruptEnable = RTC_IER_IE_64HZ_MASK,
kIRTC_128hzInterruptEnable = RTC_IER_IE_128HZ_MASK,
kIRTC_256hzInterruptEnable = RTC_IER_IE_256HZ_MASK,
kIRTC_512hzInterruptEnable = RTC_IER_IE_512HZ_MASK }
List of IRTC interrupts.

- enum irtc_status_flags_t {
kIRTC_TamperFlag = RTC_ISR_TAMPER_IS_MASK,
kIRTC_AlarmFlag = RTC_ISR_ALM_IS_MASK,
kIRTC_DayFlag = RTC_ISR_DAY_IS_MASK,
kIRTC_HourFlag = RTC_ISR_HOUR_IS_MASK,
kIRTC_MinFlag = RTC_ISR_MIN_IS_MASK,
kIRTC_1hzFlag = RTC_ISR_IS_1HZ_MASK,
kIRTC_2hzFlag = RTC_ISR_IS_2HZ_MASK,
kIRTC_4hzFlag = RTC_ISR_IS_4HZ_MASK,
kIRTC_8hzFlag = RTC_ISR_IS_8HZ_MASK,
kIRTC_16hzFlag = RTC_ISR_IS_16HZ_MASK,
kIRTC_32hzFlag = RTC_ISR_IS_32HZ_MASK,
kIRTC_64hzFlag = RTC_ISR_IS_64HZ_MASK,
kIRTC_128hzFlag = RTC_ISR_IS_128HZ_MASK,
kIRTC_256hzFlag = RTC_ISR_IS_256HZ_MASK,
kIRTC_512hzFlag = RTC_ISR_IS_512HZ_MASK,
kIRTC_InvalidFlag = (RTC_STATUS_INVAL_BIT_MASK << 16U),
kIRTC_WriteProtFlag = (RTC_STATUS_WRITE_PROT_EN_MASK << 16U),
kIRTC_CpuLowVoltFlag = (RTC_STATUS_CPU_LOW_VOLT_MASK << 16U),
kIRTC_ResetSrcFlag = (RTC_STATUS_RST_SRC_MASK << 16U),
kIRTC_CmpIntFlag = (RTC_STATUS_CMP_INT_MASK << 16U),
kIRTC_BusErrFlag = (RTC_STATUS_BUS_ERR_MASK << 16U),
kIRTC_CmpDoneFlag = (RTC_STATUS_CMP_DONE_MASK << 16U) }
List of IRTC flags.

- enum irtc_alarm_match_t {
kRTC_MatchSecMinHr = 0U,
kRTC_MatchSecMinHrDay = 1U,
kRTC_MatchSecMinHrDayMnth = 2U,
kRTC_MatchSecMinHrDayMnthYr = 3U }

IRTC alarm match options.
- enum irtc_osc_cap_load_t {
kIRTC_Capacitor2p = (1U << 1U),
kIRTC_Capacitor4p = (1U << 2U),
kIRTC_Capacitor8p = (1U << 3U),
kIRTC_Capacitor16p = (1U << 4U) }
List of RTC Oscillator capacitor load settings.
- enum irtc_clockout_sel_t {
kIRTC_ClkoutNo = 0U,
kIRTC_ClkoutFine1Hz,
kIRTC_Clkout32kHz,
kIRTC_ClkoutCoarse1Hz }
IRTC clockout select.

Functions

- static void IRTC_SetOscCapLoad (RTC_Type ∗base, uint16_t capLoad)
This function sets the specified capacitor configuration for the RTC oscillator.
- status_t IRTC_SetWriteProtection (RTC_Type ∗base, bool lock)
Locks or unlocks IRTC registers for write access.
- static void IRTC_Reset (RTC_Type ∗base)
Performs a software reset on the IRTC module.
- static void IRTC_Enable32kClkDuringRegisterWrite (RTC_Type ∗base, bool enable)
Enable/disable 32 kHz RTC OSC clock during RTC register write.
- void IRTC_ConfigClockOut (RTC_Type ∗base, irtc_clockout_sel_t clkOut)
Select which clock to output from RTC.
- static uint8_t IRTC_GetTamperStatusFlag (RTC_Type ∗base)
Gets the IRTC Tamper status flags.
- static void IRTC_ClearTamperStatusFlag (RTC_Type ∗base)
Gets the IRTC Tamper status flags.
- static void IRTC_SetTamperConfigurationOver (RTC_Type ∗base)
Set tamper configuration over.

Driver version

- #define FSL_IRTC_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
Version.

Initialization and deinitialization

- status_t IRTC_Init (RTC_Type ∗base, const irtc_config_t ∗config)
Ungates the IRTC clock and configures the peripheral for basic operation.
- static void IRTC_Deinit (RTC_Type ∗base)
Gate the IRTC clock.
- void IRTC_GetDefaultConfig (irtc_config_t ∗config)
Fill in the IRTC config struct with the default settings.

Current Time & Alarm

- status_t IRTC_SetDatetime (RTC_Type ∗base, const irtc_datetime_t ∗datetime)

Sets the IRTC date and time according to the given time structure.
- void IRTC_GetDatetime (RTC_Type ∗base, irtc_datetime_t ∗datetime)
  Gets the IRTC time and stores it in the given time structure.
- status_t IRTC_SetAlarm (RTC_Type ∗base, const irtc_datetime_t ∗alarmTime)
  Sets the IRTC alarm time.
- void IRTC_GetAlarm (RTC_Type ∗base, irtc_datetime_t ∗datetime)
  Returns the IRTC alarm time.

Interrupt Interface

- static void IRTC_EnableInterrupts (RTC_Type ∗base, uint32_t mask)
  Enables the selected IRTC interrupts.
- static void IRTC_DisableInterrupts (RTC_Type ∗base, uint32_t mask)
  Disables the selected IRTC interrupts.
- static uint32_t IRTC_GetEnabledInterrupts (RTC_Type ∗base)
  Gets the enabled IRTC interrupts.

Status Interface

- static uint32_t IRTC_GetStatusFlags (RTC_Type ∗base)
  Gets the IRTC status flags.
- static void IRTC_ClearStatusFlags (RTC_Type ∗base, uint32_t mask)
  Clears the IRTC status flags.

Daylight Savings Interface

- void IRTC_SetDaylightTime (RTC_Type ∗base, const irtc_daylight_time_t ∗datetime)
  Sets the IRTC daylight savings start and stop date and time.
- void IRTC_GetDaylightTime (RTC_Type ∗base, irtc_daylight_time_t ∗datetime)
  Gets the IRTC daylight savings time and stores it in the given time structure.

Time Compensation Interface

- void IRTC_SetCoarseCompensation (RTC_Type ∗base, uint8_t compensationValue, uint8_t compensationInterval)
  Enables the coarse compensation and sets the value in the IRTC compensation register.
- void IRTC_SetFineCompensation (RTC_Type ∗base, uint8_t integralValue, uint8_t fractionValue, bool accumulateFractional)
  Enables the fine compensation and sets the value in the IRTC compensation register.

Tamper Interface

- void IRTC_SetTamperParams (RTC_Type ∗base, irtc_tamper_pins_t tamperNumber, const irtc_tamper_config_t ∗tamperConfig)
  This function allows configuring the four tamper inputs.

16.2 Data Structure Documentation

16.2.1 struct irtc_datetime_t

Data Fields

- uint16_t year
  Range from 1984 to 2239.
- uint8_t month
  Range from 1 to 12.
- uint8_t day
  Range from 1 to 31 (depending on month).
- uint8_t weekDay
  Range from 0(Sunday) to 6(Saturday).
- 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 irtc_datetime_t::year

(2) uint8_t irtc_datetime_t::month

(3) uint8_t irtc_datetime_t::day

(4) uint8_t irtc_datetime_t::weekDay

(5) uint8_t irtc_datetime_t::hour

(6) uint8_t irtc_datetime_t::minute

(7) uint8_t irtc_datetime_t::second

16.2.2 struct irtc_daylight_time_t

Data Fields

- uint8_t startMonth
  Range from 1 to 12.
- uint8_t endMonth
  Range from 1 to 12.
- uint8_t startDay
  Range from 1 to 31 (depending on month)
- uint8_t endDay
  Range from 1 to 31 (depending on month)
- uint8_t startHour
  Range from 0 to 23.

- uint8_t endHour
  Range from 0 to 23.

16.2.3 struct irtc_tamper_config_t

Data Fields

- bool pinPolarity
  true: tamper has active low polarity; false: active high polarity
- irtc_filter_clock_source_t filterClk
  Clock source for the tamper filter.
- uint8_t filterDuration
  Tamper filter duration.

Field Documentation

(1) uint8_t irtc_tamper_config_t::filterDuration

16.2.4 struct irtc_config_t

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the IRTC_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: Tamper pin 0 is used to wakeup the chip; false: Tamper pin 0 is used as the tamper pin
- bool timerStdMask
  true: Sampling clocks gated in standby mode; false: Sampling clocks not gated
- irtc_alarm_match_t alrmMatch
  Pick one option from enumeration :: irtc_alarm_match_t.

16.3 Macro Definition Documentation

16.3.1 #define FSL_IRTC_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

16.4 Enumeration Type Documentation

16.4.1 enum irtc_filter_clock_source_t

Enumerator

kIRTC_32K Use 32 kHz clock source for the tamper filter.

kIRTC_512 Use 512 Hz clock source for the tamper filter.
kIRTC_128 Use 128 Hz clock source for the tamper filter.
kIRTC_64 Use 64 Hz clock source for the tamper filter.
kIRTC_16 Use 16 Hz clock source for the tamper filter.
kIRTC_8 Use 8 Hz clock source for the tamper filter.
kIRTC_4 Use 4 Hz clock source for the tamper filter.
kIRTC_2 Use 2 Hz clock source for the tamper filter.

16.4.2 enum irtc_tamper_pins_t

Enumerator

kIRTC_Tamper_0 External Tamper 0.
kIRTC_Tamper_1 External Tamper 1.
kIRTC_Tamper_2 External Tamper 2.
kIRTC_Tamper_3 Internal tamper, does not have filter configuration.

16.4.3 enum irtc_interrupt_enable_t

Enumerator

kIRTC_TamperInterruptEnable Tamper Interrupt Enable.
kIRTC_AlarmInterruptEnable Alarm Interrupt Enable.
kIRTC_DayInterruptEnable Days Interrupt Enable.
kIRTC_HourInterruptEnable Hours Interrupt Enable.
kIRTC_MinInterruptEnable Minutes Interrupt Enable.
kIRTC_1hzInterruptEnable 1 Hz interval Interrupt Enable
kIRTC_2hzInterruptEnable 2 Hz interval Interrupt Enable
kIRTC_4hzInterruptEnable 4 Hz interval Interrupt Enable
kIRTC_8hzInterruptEnable 8 Hz interval Interrupt Enable
kIRTC_16hzInterruptEnable 16 Hz interval Interrupt Enable
kIRTC_32hzInterruptEnable 32 Hz interval Interrupt Enable
kIRTC_64hzInterruptEnable 64 Hz interval Interrupt Enable
kIRTC_128hzInterruptEnable 128 Hz interval Interrupt Enable
kIRTC_256hzInterruptEnable 256 Hz interval Interrupt Enable
kIRTC_512hzInterruptEnable 512 Hz interval Interrupt Enable

16.4.4 enum irtc_status_flags_t

Enumerator

kIRTC_TamperFlag Tamper Status flag.

kIRTC_AlarmFlag Alarm Status flag.
kIRTC_DayFlag Days Status flag.
kIRTC_HourFlag Hour Status flag.
kIRTC_MinFlag Minutes Status flag.
kIRTC_1hzFlag 1 Hz interval status flag
kIRTC_2hzFlag 2 Hz interval status flag
kIRTC_4hzFlag 4 Hz interval status flag
kIRTC_8hzFlag 8 Hz interval status flag
kIRTC_16hzFlag 16 Hz interval status flag
kIRTC_32hzFlag 32 Hz interval status flag
kIRTC_64hzFlag 64 Hz interval status flag
kIRTC_128hzFlag 128 Hz interval status flag
kIRTC_256hzFlag 256 Hz interval status flag
kIRTC_512hzFlag 512 Hz interval status flag
kIRTC_InvalidFlag Indicates if time/date counters are invalid.
kIRTC_WriteProtFlag Write protect enable status flag.
kIRTC_CpuLowVoltFlag CPU low voltage warning flag.
kIRTC_ResetSrcFlag Reset source flag.
kIRTC_CmpIntFlag Compensation interval status flag.
kIRTC_BusErrFlag Bus error flag.
kIRTC_CmpDoneFlag Compensation done flag.

16.4.5 enum irtc_alarm_match_t

Enumerator

kRTC_MatchSecMinHr Only match second, minute and hour.
kRTC_MatchSecMinHrDay Only match second, minute, hour and day.
kRTC_MatchSecMinHrDayMnth Only match second, minute, hour, day and month.
kRTC_MatchSecMinHrDayMnthYr Only match second, minute, hour, day, month and year.

16.4.6 enum irtc_osc_cap_load_t

Enumerator

kIRTC_Capacitor2p 2pF capacitor load
kIRTC_Capacitor4p 4pF capacitor load
kIRTC_Capacitor8p 8pF capacitor load
kIRTC_Capacitor16p 16pF capacitor load

16.4.7 enum irtc_clockout_sel_t

Enumerator

kIRTC_ClkoutNo No clock out.
kIRTC_ClkoutFine1Hz clock out fine 1Hz
kIRTC_Clkout32kHz clock out 32.768kHz
kIRTC_ClkoutCoarse1Hz clock out coarse 1Hz

16.5 Function Documentation

16.5.1 status_t IRTC_Init (RTC_Type * *base*, const irtc_config_t * *config*)

This function initiates a soft-reset of the IRTC module, this has not effect on DST, calendaring, standby time and tamper detect registers.

Note

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

Parameters

base	IRTC peripheral base address
config	Pointer to user's IRTC config structure.

Returns

kStatus_Fail if we cannot disable register write protection

16.5.2 static void IRTC_Deinit (RTC_Type * *base*) [inline], [static]

Parameters

base	IRTC peripheral base address

16.5.3 void IRTC_GetDefaultConfig (irtc_config_t * *config*)

The default values are:

```
*    config->wakeupSelect = true;
*    config->timerStdMask = false;
*    config->alrmMatch = kRTC_MatchSecMinHr;
*
```

Parameters

config	Pointer to user's IRTC config structure.

16.5.4 status_t IRTC_SetDatetime (RTC_Type ∗ *base*, const irtc_datetime_t ∗ *datetime*)

The IRTC counter is started after the time is set.

Parameters

base	IRTC peripheral base address
datetime	Pointer to structure where the date and time details to set are stored

Returns

kStatus_Success: success in setting the time and starting the IRTC kStatus_InvalidArgument-: failure. An error occurs because the datetime format is incorrect.

16.5.5 void IRTC_GetDatetime (RTC_Type ∗ *base*, irtc_datetime_t ∗ *datetime*)

Parameters

base	IRTC peripheral base address
datetime	Pointer to structure where the date and time details are stored.

16.5.6 status_t IRTC_SetAlarm (RTC_Type ∗ *base*, const irtc_datetime_t ∗ *alarmTime*)

Parameters

base	RTC peripheral base address
alarmTime	Pointer to structure where the alarm time is stored.

Note

weekDay field of alarmTime is not used during alarm match and should be set to 0

Returns

kStatus_Success: success in setting the alarm kStatus_InvalidArgument: error in setting the alarm. Error occurs because the alarm datetime format is incorrect.

16.5.7 void IRTC_GetAlarm (RTC_Type ∗ *base*, irtc_datetime_t ∗ *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to structure where the alarm date and time details are stored.

16.5.8 static void IRTC_EnableInterrupts (RTC_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	IRTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration irtc_interrupt_enable_t

16.5.9 static void IRTC_DisableInterrupts (RTC_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	IRTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration irtc_interrupt_enable_t

16.5.10 static uint32_t IRTC_GetEnabledInterrupts (RTC_Type ∗ *base*) [inline], [static]

Parameters

base	IRTC peripheral base address

Returns

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

16.5.11 static uint32_t IRTC_GetStatusFlags (RTC_Type * *base*) [inline], [static]

Parameters

base	IRTC peripheral base address

Returns

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

16.5.12 static void IRTC_ClearStatusFlags (RTC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

16.5.13 static void IRTC_SetOscCapLoad (RTC_Type * *base*, uint16_t *capLoad*) [inline], [static]

Parameters

base	IRTC peripheral base address
capLoad	Oscillator loads to enable. This is a logical OR of members of the enumeration irtc_-osc_cap_load_t

16.5.14 status_t IRTC_SetWriteProtection (RTC_Type ∗ *base*, bool *lock*)

Note

When the registers are unlocked, they remain in unlocked state for 2 seconds, after which they are locked automatically. After power-on-reset, the registers come out unlocked and they are locked automatically 15 seconds after power on.

Parameters

base	IRTC peripheral base address
lock	true: Lock IRTC registers; false: Unlock IRTC registers.

Returns

kStatus_Success: if lock or unlock operation is successful kStatus_Fail: if lock or unlock operation fails even after multiple retry attempts

16.5.15 static void IRTC_Reset (RTC_Type ∗ *base*) `[inline]`, `[static]`

Clears contents of alarm, interrupt (status and enable except tamper interrupt enable bit) registers, STATUS[CMP_DONE] and STATUS[BUS_ERR]. This has no effect on DST, calendaring, standby time and tamper detect registers.

Parameters

base	IRTC peripheral base address

16.5.16 static void IRTC_Enable32kClkDuringRegisterWrite (RTC_Type ∗ *base*, bool *enable*) `[inline]`, `[static]`

Parameters

base	IRTC peripheral base address
enable	Enable/disable 32 kHz RTC OSC clock. • true: Enables the oscillator. • false: Disables the oscillator.

16.5.17 void IRTC_ConfigClockOut (RTC_Type ∗ *base*, irtc_clockout_sel_t *clkOut*)

Select which clock to output from RTC for other modules to use inside SoC, for example, RTC subsystem needs RTC to output 1HZ clock for sub-second counter.

Parameters

base	IRTC peripheral base address
cloOut	select clock to use for output,

16.5.18 static uint8_t IRTC_GetTamperStatusFlag (RTC_Type ∗ *base*) [inline], [static]

Parameters

base	IRTC peripheral base address

Returns

The Tamper status value.

16.5.19 static void IRTC_ClearTamperStatusFlag (RTC_Type ∗ *base*) [inline], [static]

Parameters

base	IRTC peripheral base address

16.5.20 static void IRTC_SetTamperConfigurationOver (RTC_Type * *base*) [inline], [static]

Note that this API is neeeded after call IRTC_SetTamperParams to configure tamper events to notify IRTC module that tamper configuration process is over.

Parameters

base	IRTC peripheral base address

16.5.21 void IRTC_SetDaylightTime (RTC_Type * *base,* const irtc_daylight_time_t * *datetime*)

It also enables the daylight saving bit in the IRTC control register

Parameters

base	IRTC peripheral base address
datetime	Pointer to a structure where the date and time details are stored.

16.5.22 void IRTC_GetDaylightTime (RTC_Type * *base,* irtc_daylight_time_t * *datetime*)

Parameters

base	IRTC peripheral base address
datetime	Pointer to a structure where the date and time details are stored.

16.5.23 void IRTC_SetCoarseCompensation (RTC_Type * *base,* uint8_t *compensationValue,* uint8_t *compensationInterval*)

Parameters

base	IRTC peripheral base address
compensation-Value	Compensation value is a 2's complement value.
compensation-Interval	Compensation interval.

16.5.24 void IRTC_SetFineCompensation (RTC_Type * *base,* uint8_t *integralValue,* uint8_t *fractionValue,* bool *accumulateFractional*)

Parameters

base	The IRTC peripheral base address
integralValue	Compensation integral value; twos complement value of the integer part
fractionValue	Compensation fraction value expressed as number of clock cycles of a fixed 4.-194304Mhz clock that have to be added.
accumulate-Fractional	Flag indicating if we want to add to previous fractional part; true: Add to previously accumulated fractional part, false: Start afresh and overwrite current value

16.5.25 void IRTC_SetTamperParams (RTC_Type ∗ *base,* irtc_tamper_pins_t *tamperNumber,* const irtc_tamper_config_t ∗ *tamperConfig*)

The function configures the filter properties for the three external tampers. It also sets up active/passive and direction of the tamper bits, which are not available on all platforms.

Note

> This function programs the tamper filter parameters. The user must gate the 32K clock to the RTC before calling this function. It is assumed that the time and date are set after this and the tamper parameters do not require to be changed again later.

Parameters

base	The IRTC peripheral base address
tamperNumber	The IRTC tamper input to configure
tamperConfig	The IRTC tamper properties

Chapter 17
LLWU: Low-Leakage Wakeup Unit Driver

17.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low-Leakage Wakeup Unit (LLWU) module of MCUXpresso SDK devices. The LLWU module allows the user to select external pin sources and internal modules as a wake-up source from low-leakage power modes.

17.2 External wakeup pins configurations

Configures the external wakeup pins' working modes, gets, and clears the wake pin flags. External wakeup pins are accessed by the `pinIndex`, which is started from 1. Numbers of the external pins depend on the SoC configuration.

17.3 Internal wakeup modules configurations

Enables/disables the internal wakeup modules and gets the module flags. Internal modules are accessed by `moduleIndex`, which is started from 1. Numbers of external pins depend the on SoC configuration.

17.4 Digital pin filter for external wakeup pin configurations

Configures the digital pin filter of the external wakeup pins' working modes, gets, and clears the pin filter flags. Digital pin filters are accessed by the `filterIndex`, which is started from 1. Numbers of external pins depend on the SoC configuration.

Data Structures

- struct llwu_external_pin_filter_mode_t
  An external input pin filter control structure. More...

Enumerations

- enum llwu_external_pin_mode_t {
  kLLWU_ExternalPinDisable = 0U,
  kLLWU_ExternalPinRisingEdge = 1U,
  kLLWU_ExternalPinFallingEdge = 2U,
  kLLWU_ExternalPinAnyEdge = 3U }
  External input pin control modes.
- enum llwu_pin_filter_mode_t {
  kLLWU_PinFilterDisable = 0U,
  kLLWU_PinFilterRisingEdge = 1U,
  kLLWU_PinFilterFallingEdge = 2U,
  kLLWU_PinFilterAnyEdge = 3U }
  Digital filter control modes.

Driver version

- #define FSL_LLWU_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  LLWU driver version.

Low-Leakage Wakeup Unit Control APIs

- void LLWU_SetExternalWakeupPinMode (LLWU_Type *base, uint32_t pinIndex, llwu_external_pin_mode_t pinMode)
  Sets the external input pin source mode.
- bool LLWU_GetExternalWakeupPinFlag (LLWU_Type *base, uint32_t pinIndex)
  Gets the external wakeup source flag.
- void LLWU_ClearExternalWakeupPinFlag (LLWU_Type *base, uint32_t pinIndex)
  Clears the external wakeup source flag.
- static void LLWU_EnableInternalModuleInterruptWakup (LLWU_Type *base, uint32_t moduleIndex, bool enable)
  Enables/disables the internal module source.
- static bool LLWU_GetInternalWakeupModuleFlag (LLWU_Type *base, uint32_t moduleIndex)
  Gets the external wakeup source flag.
- void LLWU_SetPinFilterMode (LLWU_Type *base, uint32_t filterIndex, llwu_external_pin_filter_mode_t filterMode)
  Sets the pin filter configuration.
- bool LLWU_GetPinFilterFlag (LLWU_Type *base, uint32_t filterIndex)
  Gets the pin filter configuration.
- void LLWU_ClearPinFilterFlag (LLWU_Type *base, uint32_t filterIndex)
  Clears the pin filter configuration.
- #define **INTERNAL_WAKEUP_MODULE_FLAG_REG** F3

17.5 Data Structure Documentation

17.5.1 struct llwu_external_pin_filter_mode_t

Data Fields

- uint32_t pinIndex
  A pin number.
- llwu_pin_filter_mode_t filterMode
  Filter mode.

17.6 Macro Definition Documentation

17.6.1 #define FSL_LLWU_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

17.7 Enumeration Type Documentation

17.7.1 enum llwu_external_pin_mode_t

Enumerator

kLLWU_ExternalPinDisable Pin disabled as a wakeup input.

kLLWU_ExternalPinRisingEdge Pin enabled with the rising edge detection.
kLLWU_ExternalPinFallingEdge Pin enabled with the falling edge detection.
kLLWU_ExternalPinAnyEdge Pin enabled with any change detection.

17.7.2 enum llwu_pin_filter_mode_t

Enumerator

kLLWU_PinFilterDisable Filter disabled.
kLLWU_PinFilterRisingEdge Filter positive edge detection.
kLLWU_PinFilterFallingEdge Filter negative edge detection.
kLLWU_PinFilterAnyEdge Filter any edge detection.

17.8 Function Documentation

17.8.1 void LLWU_SetExternalWakeupPinMode (LLWU_Type ∗ *base*, uint32_t *pinIndex*, llwu_external_pin_mode_t *pinMode*)

This function sets the external input pin source mode that is used as a wake up source.

Parameters

base	LLWU peripheral base address.
pinIndex	A pin index to be enabled as an external wakeup source starting from 1.
pinMode	A pin configuration mode defined in the llwu_external_pin_modes_t.

17.8.2 bool LLWU_GetExternalWakeupPinFlag (LLWU_Type ∗ *base*, uint32_t *pinIndex*)

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

Parameters

base	LLWU peripheral base address.
pinIndex	A pin index, which starts from 1.

Returns

True if the specific pin is a wakeup source.

17.8.3 void LLWU_ClearExternalWakeupPinFlag (LLWU_Type * *base*, uint32_t *pinIndex*)

This function clears the external wakeup source flag for a specific pin.

Parameters

base	LLWU peripheral base address.
pinIndex	A pin index, which starts from 1.

17.8.4 static void LLWU_EnableInternalModuleInterruptWakup (LLWU_Type * *base*, uint32_t *moduleIndex*, bool *enable*) [inline], [static]

This function enables/disables the internal module source mode that is used as a wake up source.

Parameters

base	LLWU peripheral base address.
moduleIndex	A module index to be enabled as an internal wakeup source starting from 1.
enable	An enable or a disable setting

17.8.5 static bool LLWU_GetInternalWakeupModuleFlag (LLWU_Type * *base*, uint32_t *moduleIndex*) [inline], [static]

This function checks the external pin flag to detect whether the system is woken up by the specific pin.

Parameters

base	LLWU peripheral base address.
moduleIndex	A module index, which starts from 1.

Returns

True if the specific pin is a wake up source.

17.8.6 void LLWU_SetPinFilterMode (LLWU_Type * *base*, uint32_t *filterIndex*, llwu_external_pin_filter_mode_t *filterMode*)

This function sets the pin filter configuration.

Parameters

base	LLWU peripheral base address.
filterIndex	A pin filter index used to enable/disable the digital filter, starting from 1.
filterMode	A filter mode configuration

17.8.7 bool LLWU_GetPinFilterFlag (LLWU_Type * *base*, uint32_t *filterIndex*)

This function gets the pin filter flag.

Parameters

base	LLWU peripheral base address.
filterIndex	A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

17.8.8 void LLWU_ClearPinFilterFlag (LLWU_Type * *base*, uint32_t *filterIndex*)

This function clears the pin filter flag.

Parameters

base	LLWU peripheral base address.
filterIndex	A pin filter index to clear the flag, starting from 1.

Chapter 18
LPTMR: Low-Power Timer

18.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

18.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

18.2.1 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.

18.2.2 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.

18.2.3 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.

18.2.4 Status

Provides functions to get and clear the LPTMR status.

18.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

18.3 Typical use case

18.3.1 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.

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.

18.4 Data Structure Documentation

18.4.1 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.

18.5 Enumeration Type Documentation

18.5.1 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.

18.5.2 enum lptmr_pin_polarity_t

Enumerator

kLPTMR_PinPolarityActiveHigh Pulse Counter input source is active-high.
kLPTMR_PinPolarityActiveLow Pulse Counter input source is active-low.

18.5.3 enum lptmr_timer_mode_t

Enumerator

kLPTMR_TimerModeTimeCounter Time Counter mode.
kLPTMR_TimerModePulseCounter Pulse Counter mode.

18.5.4 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.

18.5.5 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.

18.5.6 enum lptmr_interrupt_enable_t

Enumerator

kLPTMR_TimerInterruptEnable Timer interrupt enable.

18.5.7 enum lptmr_status_flags_t

Enumerator

kLPTMR_TimerCompareFlag Timer compare flag.

18.6 Function Documentation

18.6.1 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.

18.6.2 void LPTMR_Deinit (LPTMR_Type * *base*)

Parameters

base	LPTMR peripheral base address

18.6.3 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.

18.6.4 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

18.6.5 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.

18.6.6 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

18.6.7 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

18.6.8 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.

18.6.9 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.

18.6.10 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

18.6.11 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

18.6.12 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 19
PIT: Periodic Interrupt Timer

19.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

19.2 Function groups

The PIT driver supports operating the module as a time counter.

19.2.1 Initialization and deinitialization

The function PIT_Init() initializes the PIT with specified configurations. The function PIT_GetDefaultConfig() gets the default configurations. The initialization function configures the PIT operation in debug mode.

The function PIT_SetTimerChainMode() configures the chain mode operation of each PIT channel.

The function PIT_Deinit() disables the PIT timers and disables the module clock.

19.2.2 Timer period Operations

The function PITR_SetTimerPeriod() sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function PIT_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. Users can call the utility macros provided in fsl_common.h to convert to microseconds or milliseconds.

19.2.3 Start and Stop timer operations

The function PIT_StartTimer() starts the timer counting. After calling this function, the timer loads the period value set earlier via the PIT_SetPeriod() function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function PIT_StopTimer() stops the timer counting.

19.2.4 Status

Provides functions to get and clear the PIT status.

19.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

19.3 Typical use case

19.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pit

Data Structures

- struct pit_config_t
  PIT configuration structure. More...

Enumerations

- enum pit_chnl_t {
  kPIT_Chnl_0 = 0U,
  kPIT_Chnl_1,
  kPIT_Chnl_2,
  kPIT_Chnl_3 }
  List of PIT channels.
- enum pit_interrupt_enable_t { kPIT_TimerInterruptEnable = PIT_TCTRL_TIE_MASK }
  List of PIT interrupts.
- enum pit_status_flags_t { kPIT_TimerFlag = PIT_TFLG_TIF_MASK }
  List of PIT status flags.

Driver version

- #define FSL_PIT_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  PIT Driver Version 2.0.4.

Initialization and deinitialization

- void PIT_Init (PIT_Type *base, const pit_config_t *config)
  Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.
- void PIT_Deinit (PIT_Type *base)
  Gates the PIT clock and disables the PIT module.
- static void PIT_GetDefaultConfig (pit_config_t *config)
  Fills in the PIT configuration structure with the default settings.
- static void PIT_SetTimerChainMode (PIT_Type *base, pit_chnl_t channel, bool enable)
  Enables or disables chaining a timer with the previous timer.

Interrupt Interface

- static void PIT_EnableInterrupts (PIT_Type *base, pit_chnl_t channel, uint32_t mask)
  Enables the selected PIT interrupts.
- static void PIT_DisableInterrupts (PIT_Type *base, pit_chnl_t channel, uint32_t mask)
  Disables the selected PIT interrupts.
- static uint32_t PIT_GetEnabledInterrupts (PIT_Type *base, pit_chnl_t channel)
  Gets the enabled PIT interrupts.

Status Interface

- static uint32_t PIT_GetStatusFlags (PIT_Type *base, pit_chnl_t channel)
  Gets the PIT status flags.
- static void PIT_ClearStatusFlags (PIT_Type *base, pit_chnl_t channel, uint32_t mask)
  Clears the PIT status flags.

Read and Write the timer period

- static void PIT_SetTimerPeriod (PIT_Type *base, pit_chnl_t channel, uint32_t count)
  Sets the timer period in units of count.
- static uint32_t PIT_GetCurrentTimerCount (PIT_Type *base, pit_chnl_t channel)
  Reads the current timer counting value.

Timer Start and Stop

- static void PIT_StartTimer (PIT_Type *base, pit_chnl_t channel)
  Starts the timer counting.
- static void PIT_StopTimer (PIT_Type *base, pit_chnl_t channel)
  Stops the timer counting.

19.4 Data Structure Documentation

19.4.1 struct pit_config_t

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

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

Data Fields

- bool enableRunInDebug
  true: Timers run in debug mode; false: Timers stop in debug mode

19.5 Enumeration Type Documentation

19.5.1 enum pit_chnl_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPIT_Chnl_0 PIT channel number 0.
kPIT_Chnl_1 PIT channel number 1.
kPIT_Chnl_2 PIT channel number 2.
kPIT_Chnl_3 PIT channel number 3.

19.5.2 enum pit_interrupt_enable_t

Enumerator

kPIT_TimerInterruptEnable Timer interrupt enable.

19.5.3 enum pit_status_flags_t

Enumerator

kPIT_TimerFlag Timer flag.

19.6 Function Documentation

19.6.1 void PIT_Init (PIT_Type * *base*, const pit_config_t * *config*)

Note

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

Parameters

base	PIT peripheral base address
config	Pointer to the user's PIT config structure

19.6.2 void PIT_Deinit (PIT_Type * *base*)

Parameters

base	PIT peripheral base address

19.6.3 static void PIT_GetDefaultConfig (pit_config_t * *config*) [inline], [static]

The default values are as follows.

```
*     config->enableRunInDebug = false;
*
```

Parameters

config	Pointer to the configuration structure.

19.6.4 static void PIT_SetTimerChainMode (PIT_Type * *base*, pit_chnl_t *channel*, bool *enable*) [inline], [static]

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

base	PIT peripheral base address
channel	Timer channel number which is chained with the previous timer
enable	Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers.

19.6.5 static void PIT_EnableInterrupts (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number
mask	The interrupts to enable. This is a logical OR of members of the enumeration pit_-interrupt_enable_t

19.6.6 static void PIT_DisableInterrupts (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number
mask	The interrupts to disable. This is a logical OR of members of the enumeration pit_-interrupt_enable_t

19.6.7 static uint32_t PIT_GetEnabledInterrupts (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number

Returns

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

19.6.8 static uint32_t PIT_GetStatusFlags (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number

Returns

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

19.6.9 static void PIT_ClearStatusFlags (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number
mask	The status flags to clear. This is a logical OR of members of the enumeration pit_-status_flags_t

19.6.10 static void PIT_SetTimerPeriod (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *count*) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note

Users can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

base	PIT peripheral base address
channel	Timer channel number

count	Timer period in units of ticks

19.6.11 static uint32_t PIT_GetCurrentTimerCount (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

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

Note

Users can call the utility macros provided in fsl_common.h to convert ticks to usec or msec.

Parameters

base	PIT peripheral base address
channel	Timer channel number

Returns

Current timer counting value in ticks

19.6.12 static void PIT_StartTimer (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

base	PIT peripheral base address
channel	Timer channel number.

19.6.13 static void PIT_StopTimer (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

base	PIT peripheral base address
channel	Timer channel number.

Chapter 20
PMC: Power Management Controller

20.1 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...
- struct pmc_bandgap_buffer_config_t
  Bandgap Buffer configuration. More...

Enumerations

- enum pmc_low_volt_detect_volt_select_t {
  kPMC_LowVoltDetectLowTrip = 0U,
  kPMC_LowVoltDetectHighTrip = 1U }
  Low-voltage Detect Voltage Select.
- enum pmc_low_volt_warning_volt_select_t {
  kPMC_LowVoltWarningLowTrip = 0U,
  kPMC_LowVoltWarningMid1Trip = 1U,
  kPMC_LowVoltWarningMid2Trip = 2U,
  kPMC_LowVoltWarningHighTrip = 3U }
  Low-voltage Warning Voltage Select.
- enum pmc_bandgap_buffer_drive_select_t {
  kPMC_BandgapBufferDriveLow = 0U,
  kPMC_BandgapBufferDriveHigh = 1U }
  Bandgap Buffer Drive Select.

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.
- void PMC_ConfigureBandgapBuffer (PMC_Type ∗base, const pmc_bandgap_buffer_config_t ∗config)
  Configures the PMC bandgap.
- static bool PMC_GetPeriphIOIsolationFlag (PMC_Type ∗base)
  Gets the acknowledge Peripherals and I/O pads isolation flag.
- static void PMC_ClearPeriphIOIsolationFlag (PMC_Type ∗base)
  Acknowledges the isolation flag to Peripherals and I/O pads.
- static bool PMC_IsRegulatorInRunRegulation (PMC_Type ∗base)
  Gets the regulator regulation status.

20.2 Data Structure Documentation

20.2.1 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.
- pmc_low_volt_detect_volt_select_t voltSelect
  Low-voltage detect trip point voltage selection.

20.2.2 struct pmc_low_volt_warning_config_t

Data Fields

- bool enableInt
  Enable interrupt when low-voltage warning.
- pmc_low_volt_warning_volt_select_t voltSelect
  Low-voltage warning trip point voltage selection.

20.2.3 struct pmc_bandgap_buffer_config_t

Data Fields

- bool enable
  Enable bandgap buffer.
- bool enableInLowPowerMode
  Enable bandgap buffer in low-power mode.
- pmc_bandgap_buffer_drive_select_t drive
  Bandgap buffer drive select.

Field Documentation

(1) bool pmc_bandgap_buffer_config_t::enable

(2) bool pmc_bandgap_buffer_config_t::enableInLowPowerMode

(3) pmc_bandgap_buffer_drive_select_t pmc_bandgap_buffer_config_t::drive

20.3 Macro Definition Documentation

20.3.1 #define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Version 2.0.3.

20.4 Enumeration Type Documentation

20.4.1 enum pmc_low_volt_detect_volt_select_t

Enumerator

kPMC_LowVoltDetectLowTrip Low-trip point selected (VLVD = VLVDL)
kPMC_LowVoltDetectHighTrip High-trip point selected (VLVD = VLVDH)

20.4.2 enum pmc_low_volt_warning_volt_select_t

Enumerator

kPMC_LowVoltWarningLowTrip Low-trip point selected (VLVW = VLVW1)
kPMC_LowVoltWarningMid1Trip Mid 1 trip point selected (VLVW = VLVW2)
kPMC_LowVoltWarningMid2Trip Mid 2 trip point selected (VLVW = VLVW3)
kPMC_LowVoltWarningHighTrip High-trip point selected (VLVW = VLVW4)

20.4.3 enum pmc_bandgap_buffer_drive_select_t

Enumerator

kPMC_BandgapBufferDriveLow Low-drive.
kPMC_BandgapBufferDriveHigh High-drive.

20.5 Function Documentation

20.5.1 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.

20.5.2 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

20.5.3 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.

20.5.4 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.

20.5.5 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.

20.5.6 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.

20.5.7 void PMC_ConfigureBandgapBuffer (PMC_Type * *base,* const pmc_bandgap_buffer_config_t * *config*)

This function configures the PMC bandgap, including the drive select and behavior in low-power mode.

Parameters

base	PMC peripheral base address.
config	Pointer to the configuration structure

20.5.8 static bool PMC_GetPeriphIOIsolationFlag (PMC_Type * *base*) [inline], [static]

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

Parameters

base	PMC peripheral base address.
base	Base address for current PMC instance.

Returns

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

20.5.9 static void PMC_ClearPeriphIOIsolationFlag (PMC_Type * *base*) [inline], [static]

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

Parameters

base	PMC peripheral base address.

20.5.10 static bool PMC_IsRegulatorInRunRegulation (PMC_Type * *base*) [inline], [static]

This function returns the regulator to run a regulation status. It provides the current status of the internal voltage regulator.

Parameters

base	PMC peripheral base address.
base	Base address for current PMC instance.

Returns

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.

Chapter 21
PORT: Port Control and Interrupts

21.1 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_slew_rate {
  kPORT_FastSlewRate = 0U,
  kPORT_SlowSlewRate = 1U }
  Slew rate selection.
- 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 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.

21.2 Data Structure Documentation

21.2.1 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.

21.2.2 struct port_pin_config_t

Data Fields

- uint16_t pullSelect: 2
  No-pull/pull-down/pull-up select.
- uint16_t slewRate: 1
  Fast/slow slew rate Configure.
- uint16_t mux: 3
  Pin mux Configure.
- uint16_t lockRegister: 1
  Lock/unlock the PCR field[15:0].

21.3 Macro Definition Documentation

21.3.1 #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

21.4 Enumeration Type Documentation

21.4.1 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.

21.4.2 enum _port_slew_rate

Enumerator

kPORT_FastSlewRate Fast slew rate is configured.
kPORT_SlowSlewRate Slow slew rate is configured.

21.4.3 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.

21.4.4 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.

21.4.5 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.

21.4.6 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.

21.5 Function Documentation

21.5.1 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.

21.5.2 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.

21.5.3 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

21.5.4 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.

21.5.5 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.

21.5.6 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).

21.5.7 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.

21.5.8 static void PORT_ClearPinsInterruptFlags (PORT_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
mask	PORT pin number macro.

Chapter 22
QTMR: Quad Timer Driver

22.1 Overview

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

Data Structures

- struct qtmr_config_t
  Quad Timer config structure. More...

Enumerations

- enum qtmr_primary_count_source_t {
  kQTMR_ClockCounter0InputPin = 0,
  kQTMR_ClockCounter1InputPin,
  kQTMR_ClockCounter2InputPin,
  kQTMR_ClockCounter3InputPin,
  kQTMR_ClockCounter0Output,
  kQTMR_ClockCounter1Output,
  kQTMR_ClockCounter2Output,
  kQTMR_ClockCounter3Output,
  kQTMR_ClockDivide_1,
  kQTMR_ClockDivide_2,
  kQTMR_ClockDivide_4,
  kQTMR_ClockDivide_8,
  kQTMR_ClockDivide_16,
  kQTMR_ClockDivide_32,
  kQTMR_ClockDivide_64,
  kQTMR_ClockDivide_128 }
  Quad Timer primary clock source selection.
- enum qtmr_input_source_t {
  kQTMR_Counter0InputPin = 0,
  kQTMR_Counter1InputPin,
  kQTMR_Counter2InputPin,
  kQTMR_Counter3InputPin }
  Quad Timer input sources selection.
- enum qtmr_counting_mode_t {

kQTMR_NoOperation = 0,
kQTMR_PriSrcRiseEdge,
kQTMR_PriSrcRiseAndFallEdge,
kQTMR_PriSrcRiseEdgeSecInpHigh,
kQTMR_QuadCountMode,
kQTMR_PriSrcRiseEdgeSecDir,
kQTMR_SecSrcTrigPriCnt,
kQTMR_CascadeCount }

Quad Timer counting mode selection.

- enum qtmr_output_mode_t {
kQTMR_AssertWhenCountActive = 0,
kQTMR_ClearOnCompare,
kQTMR_SetOnCompare,
kQTMR_ToggleOnCompare,
kQTMR_ToggleOnAltCompareReg,
kQTMR_SetOnCompareClearOnSecSrcInp,
kQTMR_SetOnCompareClearOnCountRoll,
kQTMR_EnableGateClock }

  Quad Timer output mode selection.
- enum qtmr_input_capture_edge_t {
kQTMR_NoCapture = 0,
kQTMR_RisingEdge,
kQTMR_FallingEdge,
kQTMR_RisingAndFallingEdge }

  Quad Timer input capture edge mode, rising edge, or falling edge.
- enum qtmr_preload_control_t {
kQTMR_NoPreload = 0,
kQTMR_LoadOnComp1,
kQTMR_LoadOnComp2 }

  Quad Timer input capture edge mode, rising edge, or falling edge.
- enum qtmr_debug_action_t {
kQTMR_RunNormalInDebug = 0U,
kQTMR_HaltCounter,
kQTMR_ForceOutToZero,
kQTMR_HaltCountForceOutZero }

  List of Quad Timer run options when in Debug mode.
- enum qtmr_interrupt_enable_t {
kQTMR_CompareInterruptEnable = (1U << 0),
kQTMR_Compare1InterruptEnable = (1U << 1),
kQTMR_Compare2InterruptEnable = (1U << 2),
kQTMR_OverflowInterruptEnable = (1U << 3),
kQTMR_EdgeInterruptEnable = (1U << 4) }

  List of Quad Timer interrupts.
- enum qtmr_status_flags_t {

kQTMR_CompareFlag = (1U << 0),
kQTMR_Compare1Flag = (1U << 1),
kQTMR_Compare2Flag = (1U << 2),
kQTMR_OverflowFlag = (1U << 3),
kQTMR_EdgeFlag = (1U << 4) }
List of Quad Timer flags.

Functions

- status_t QTMR_SetupPwm (TMR_Type *base, uint32_t pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity, uint32_t srcClock_Hz)
  Sets up Quad timer module for PWM signal output.
- void QTMR_SetupInputCapture (TMR_Type *base, qtmr_input_source_t capturePin, bool inputPolarity, bool reloadOnCapture, qtmr_input_capture_edge_t captureMode)
  Allows the user to count the source clock cycles until a capture event arrives.

Driver version

- #define FSL_QTMR_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  Version.

Initialization and deinitialization

- void QTMR_Init (TMR_Type *base, const qtmr_config_t *config)
  Ungates the Quad Timer clock and configures the peripheral for basic operation.
- void QTMR_Deinit (TMR_Type *base)
  Stops the counter and gates the Quad Timer clock.
- void QTMR_GetDefaultConfig (qtmr_config_t *config)
  Fill in the Quad Timer config struct with the default settings.

Interrupt Interface

- void QTMR_EnableInterrupts (TMR_Type *base, uint32_t mask)
  Enables the selected Quad Timer interrupts.
- void QTMR_DisableInterrupts (TMR_Type *base, uint32_t mask)
  Disables the selected Quad Timer interrupts.
- uint32_t QTMR_GetEnabledInterrupts (TMR_Type *base)
  Gets the enabled Quad Timer interrupts.

Status Interface

- uint32_t QTMR_GetStatus (TMR_Type *base)
  Gets the Quad Timer status flags.
- void QTMR_ClearStatusFlags (TMR_Type *base, uint32_t mask)
  Clears the Quad Timer status flags.

Read and Write the timer period

- void QTMR_SetTimerPeriod (TMR_Type *base, uint16_t ticks)

Sets the timer period in ticks.
- static uint16_t QTMR_GetCurrentTimerCount (TMR_Type *base)
  Reads the current timer counting value.

Timer Start and Stop

- static void QTMR_StartTimer (TMR_Type *base, qtmr_counting_mode_t clockSource)
  Starts the Quad Timer counter.
- static void QTMR_StopTimer (TMR_Type *base)
  Stops the Quad Timer counter.

22.2 Data Structure Documentation

22.2.1 struct qtmr_config_t

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

The config struct can be made const so it resides in flash

Data Fields

- qtmr_primary_count_source_t primarySource
  Specify the primary count source.
- qtmr_input_source_t secondarySource
  Specify the secondary count source.
- bool enableMasterMode
  true: Broadcast compare function output to other counters; false no broadcast
- bool enableExternalForce
  true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter
- uint8_t faultFilterCount
  Fault filter count.
- uint8_t faultFilterPeriod
  Fault filter period;value of 0 will bypass the filter.
- qtmr_debug_action_t debugMode
  Operation in Debug mode.

22.3 Macro Definition Documentation

22.3.1 #define FSL_QTMR_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

22.4 Enumeration Type Documentation

22.4.1 enum qtmr_primary_count_source_t

Enumerator

kQTMR_ClockCounter0InputPin Use counter 0 input pin.
kQTMR_ClockCounter1InputPin Use counter 1 input pin.
kQTMR_ClockCounter2InputPin Use counter 2 input pin.
kQTMR_ClockCounter3InputPin Use counter 3 input pin.
kQTMR_ClockCounter0Output Use counter 0 output.
kQTMR_ClockCounter1Output Use counter 1 output.
kQTMR_ClockCounter2Output Use counter 2 output.
kQTMR_ClockCounter3Output Use counter 3 output.
kQTMR_ClockDivide_1 IP bus clock divide by 1 prescaler.
kQTMR_ClockDivide_2 IP bus clock divide by 2 prescaler.
kQTMR_ClockDivide_4 IP bus clock divide by 4 prescaler.
kQTMR_ClockDivide_8 IP bus clock divide by 8 prescaler.
kQTMR_ClockDivide_16 IP bus clock divide by 16 prescaler.
kQTMR_ClockDivide_32 IP bus clock divide by 32 prescaler.
kQTMR_ClockDivide_64 IP bus clock divide by 64 prescaler.
kQTMR_ClockDivide_128 IP bus clock divide by 128 prescaler.

22.4.2 enum qtmr_input_source_t

Enumerator

kQTMR_Counter0InputPin Use counter 0 input pin.
kQTMR_Counter1InputPin Use counter 1 input pin.
kQTMR_Counter2InputPin Use counter 2 input pin.
kQTMR_Counter3InputPin Use counter 3 input pin.

22.4.3 enum qtmr_counting_mode_t

Enumerator

kQTMR_NoOperation No operation.
kQTMR_PriSrcRiseEdge Count rising edges of primary source.
kQTMR_PriSrcRiseAndFallEdge Count rising and falling edges of primary source.
kQTMR_PriSrcRiseEdgeSecInpHigh Count rise edges of pri SRC while sec inp high active.
kQTMR_QuadCountMode Quadrature count mode, uses pri and sec sources.
kQTMR_PriSrcRiseEdgeSecDir Count rising edges of pri SRC; sec SRC specifies dir.
kQTMR_SecSrcTrigPriCnt Edge of sec SRC trigger primary count until compare.
kQTMR_CascadeCount Cascaded count mode (up/down)

22.4.4 enum qtmr_output_mode_t

Enumerator

kQTMR_AssertWhenCountActive Assert OFLAG while counter is active.
kQTMR_ClearOnCompare Clear OFLAG on successful compare.
kQTMR_SetOnCompare Set OFLAG on successful compare.
kQTMR_ToggleOnCompare Toggle OFLAG on successful compare.
kQTMR_ToggleOnAltCompareReg Toggle OFLAG using alternating compare registers.
kQTMR_SetOnCompareClearOnSecSrcInp Set OFLAG on compare, clear on sec SRC input edge.

kQTMR_SetOnCompareClearOnCountRoll Set OFLAG on compare, clear on counter rollover.
kQTMR_EnableGateClock Enable gated clock output while count is active.

22.4.5 enum qtmr_input_capture_edge_t

Enumerator

kQTMR_NoCapture Capture is disabled.
kQTMR_RisingEdge Capture on rising edge (IPS=0) or falling edge (IPS=1)
kQTMR_FallingEdge Capture on falling edge (IPS=0) or rising edge (IPS=1)
kQTMR_RisingAndFallingEdge Capture on both edges.

22.4.6 enum qtmr_preload_control_t

Enumerator

kQTMR_NoPreload Never preload.
kQTMR_LoadOnComp1 Load upon successful compare with value in COMP1.
kQTMR_LoadOnComp2 Load upon successful compare with value in COMP2.

22.4.7 enum qtmr_debug_action_t

Enumerator

kQTMR_RunNormalInDebug Continue with normal operation.
kQTMR_HaltCounter Halt counter.
kQTMR_ForceOutToZero Force output to logic 0.
kQTMR_HaltCountForceOutZero Halt counter and force output to logic 0.

22.4.8 enum qtmr_interrupt_enable_t

Enumerator

kQTMR_CompareInterruptEnable Compare interrupt.
kQTMR_Compare1InterruptEnable Compare 1 interrupt.
kQTMR_Compare2InterruptEnable Compare 2 interrupt.
kQTMR_OverflowInterruptEnable Timer overflow interrupt.
kQTMR_EdgeInterruptEnable Input edge interrupt.

22.4.9 enum qtmr_status_flags_t

Enumerator

kQTMR_CompareFlag Compare flag.
kQTMR_Compare1Flag Compare 1 flag.
kQTMR_Compare2Flag Compare 2 flag.
kQTMR_OverflowFlag Timer overflow flag.
kQTMR_EdgeFlag Input edge flag.

22.5 Function Documentation

22.5.1 void QTMR_Init (TMR_Type ∗ *base*, const qtmr_config_t ∗ *config*)

Note

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

Parameters

base	Quad Timer peripheral base address
config	Pointer to user's Quad Timer config structure

22.5.2 void QTMR_Deinit (TMR_Type ∗ *base*)

Parameters

base	Quad Timer peripheral base address

22.5.3 void QTMR_GetDefaultConfig (qtmr_config_t * *config*)

The default values are:

```
*     config->debugMode = kQTMR_RunNormalInDebug;
*     config->enableExternalForce = false;
*     config->enableMasterMode = false;
*     config->faultFilterCount = 0;
*     config->faultFilterPeriod = 0;
*     config->primarySource = kQTMR_ClockDivide_2;
*     config->secondarySource = kQTMR_Counter0InputPin;
*
```

Parameters

config	Pointer to user's Quad Timer config structure.

22.5.4 status_t QTMR_SetupPwm (TMR_Type * *base*, uint32_t *pwmFreqHz*, uint8_t *dutyCyclePercent*, bool *outputPolarity*, uint32_t *srcClock_Hz*)

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

base	Quad Timer peripheral base address
pwmFreqHz	PWM signal frequency in Hz
dutyCycle-Percent	PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)
outputPolarity	true: invert polarity of the output signal, false: no inversion
srcClock_Hz	Main counter clock in Hz.

Returns

Returns an error if there was error setting up the signal.

22.5.5 void QTMR_SetupInputCapture (TMR_Type * *base*, qtmr_input-_source_t *capturePin*, bool *inputPolarity*, bool *reloadOnCapture*, qtmr_input_capture_edge_t *captureMode*)

The count is stored in the capture register.

Parameters

base	Quad Timer peripheral base address
capturePin	Pin through which we receive the input signal to trigger the capture
inputPolarity	true: invert polarity of the input signal, false: no inversion
reloadOn-Capture	true: reload the counter when an input capture occurs, false: no reload
captureMode	Specifies which edge of the input signal triggers a capture

22.5.6 void QTMR_EnableInterrupts (TMR_Type ∗ *base*, uint32_t *mask*)

Parameters

base	Quad Timer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration qtmr_interrupt_enable_t

22.5.7 void QTMR_DisableInterrupts (TMR_Type ∗ *base*, uint32_t *mask*)

Parameters

base	Quad Timer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration qtmr_interrupt_enable_t

22.5.8 uint32_t QTMR_GetEnabledInterrupts (TMR_Type ∗ *base*)

Parameters

base	Quad Timer peripheral base address

Returns

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

22.5.9 uint32_t QTMR_GetStatus (TMR_Type * *base*)

Parameters

base	Quad Timer peripheral base address

Returns

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

22.5.10 void QTMR_ClearStatusFlags (TMR_Type * *base*, uint32_t *mask*)

Parameters

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

22.5.11 void QTMR_SetTimerPeriod (TMR_Type * *base*, uint16_t *ticks*)

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in fsl_common.h to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.

Parameters

base	Quad Timer peripheral base address
ticks	Timer period in units of ticks

22.5.12 static uint16_t QTMR_GetCurrentTimerCount (TMR_Type * *base*) [inline], [static]

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

Note

User can call the utility macros provided in fsl_common.h to convert ticks to usec or msec

Parameters

base	Quad Timer peripheral base address

Returns

Current counter value in ticks

22.5.13 static void QTMR_StartTimer (TMR_Type ∗ *base*, qtmr_counting_mode_t *clockSource*) [inline], [static]

Parameters

base	Quad Timer peripheral base address
clockSource	Quad Timer clock source

22.5.14 static void QTMR_StopTimer (TMR_Type ∗ *base*) [inline], [static]

Parameters

base	Quad Timer peripheral base address

Chapter 23
RCM: Reset Control Module Driver

23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Reset Control Module (RCM) module of MCUXpresso SDK devices.

Data Structures

- struct rcm_reset_pin_filter_config_t
  Reset pin filter configuration. More...

Enumerations

- enum rcm_reset_source_t {
  kRCM_SourceWakeup = RCM_SRS0_WAKEUP_MASK,
  kRCM_SourceLvd = RCM_SRS0_LVD_MASK,
  kRCM_SourceLoc = RCM_SRS0_LOC_MASK,
  kRCM_SourceLol = RCM_SRS0_LOL_MASK,
  kRCM_SourceWdog = RCM_SRS0_WDOG_MASK,
  kRCM_SourcePin = RCM_SRS0_PIN_MASK,
  kRCM_SourcePor = RCM_SRS0_POR_MASK,
  kRCM_SourceLockup = RCM_SRS1_LOCKUP_MASK << 8U,
  kRCM_SourceSw = RCM_SRS1_SW_MASK << 8U,
  kRCM_SourceMdmap = RCM_SRS1_MDM_AP_MASK << 8U,
  kRCM_SourceSackerr = RCM_SRS1_SACKERR_MASK << 8U }
  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.

Driver version

- #define FSL_RCM_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  RCM driver version 2.0.4.

Reset Control Module APIs

- static uint32_t RCM_GetPreviousResetSources (RCM_Type *base)
  Gets the reset source status which caused a previous reset.

- void RCM_ConfigureResetPinFilter (RCM_Type *base, const rcm_reset_pin_filter_config_t *config)
  Configures the reset pin filter.

23.2 Data Structure Documentation

23.2.1 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

23.3 Macro Definition Documentation

23.3.1 #define FSL_RCM_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

23.4 Enumeration Type Documentation

23.4.1 enum rcm_reset_source_t

Enumerator

kRCM_SourceWakeup Low-leakage wakeup reset.
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.

23.4.2 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.

23.5 Function Documentation

23.5.1 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.

23.5.2 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.

Chapter 24
RNGA: Random Number Generator Accelerator Driver

24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number Generator Accelerator (R-NGA) block of MCUXpresso SDK devices.

24.2 RNGA Initialization

1. To initialize the RNGA module, call the RNGA_Init() function. This function automatically enables the RNGA module and its clock.
2. After calling the RNGA_Init() function, the RNGA is enabled and the counter starts working.
3. To disable the RNGA module, call the RNGA_Deinit() function.

24.3 Get random data from RNGA

1. RNGA_GetRandomData() function gets random data from the RNGA module.

24.4 RNGA Set/Get Working Mode

The RNGA works either in sleep mode or normal mode

1. RNGA_SetMode() function sets the RNGA mode.
2. RNGA_GetMode() function gets the RNGA working mode.

24.5 Seed RNGA

1. RNGA_Seed() function inputs an entropy value that the RNGA can use to seed the pseudo random algorithm.

This example code shows how to initialize and get random data from the RNGA driver:

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

Note

> It is important to note that there is no known cryptographic proof showing this is a secure method for generating random data. In fact, there may be an attack against this random number generator if its output is used directly in a cryptographic application. The attack is based on the linearity of the internal shift registers. Therefore, it is highly recommended that the random data produced by this module be used as an entropy source to provide an input seed to a NIST-approved pseudo-random-number generator based on DES or SHA-1 and defined in NIST FIPS PUB 186-2 Appendix 3 and NIST FIPS PUB SP 800-90. The requirement is needed to maximize the entropy of this input seed. To do this, when data is extracted from RNGA as quickly as the hardware allows, there are one to two bits of added entropy per 32-bit word. Any single bit of that word contains that entropy.

Therefore, when used as an entropy source, a random number should be generated for each bit of entropy required and the least significant bit (any bit would be equivalent) of each word retained. The remainder of each random number should then be discarded. Used this way, even with full knowledge of the internal state of RNGA and all prior random numbers, an attacker is not able to predict the values of the extracted bits. Other sources of entropy can be used along with RNGA to generate the seed to the pseudorandom algorithm. The more random sources combined to create the seed, the better. The following is a list of sources that can be easily combined with the output of this module.

- Current time using highest precision possible
- Real-time system inputs that can be characterized as "random"
- Other entropy supplied directly by the user

Enumerations

- enum rnga_mode_t {
  kRNGA_ModeNormal = 0U,
  kRNGA_ModeSleep = 1U }
  RNGA working mode.

Functions

- void RNGA_Init (RNG_Type ∗base)
  Initializes the RNGA.
- void RNGA_Deinit (RNG_Type ∗base)
  Shuts down the RNGA.
- status_t RNGA_GetRandomData (RNG_Type ∗base, void ∗data, size_t data_size)
  Gets random data.
- void RNGA_Seed (RNG_Type ∗base, uint32_t seed)
  Feeds the RNGA module.
- void RNGA_SetMode (RNG_Type ∗base, rnga_mode_t mode)
  Sets the RNGA in normal mode or sleep mode.
- rnga_mode_t RNGA_GetMode (RNG_Type ∗base)
  Gets the RNGA working mode.

Driver version

- #define FSL_RNGA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  RNGA driver version 2.0.2.

24.6 Macro Definition Documentation

24.6.1 #define FSL_RNGA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

24.7 Enumeration Type Documentation

24.7.1 enum rnga_mode_t

Enumerator

kRNGA_ModeNormal Normal Mode. The ring-oscillator clocks are active; RNGA generates entropy (randomness) from the clocks and stores it in shift registers.

kRNGA_ModeSleep Sleep Mode. The ring-oscillator clocks are inactive; RNGA does not generate entropy.

24.8 Function Documentation

24.8.1 void RNGA_Init (RNG_Type ∗ *base*)

This function initializes the RNGA. When called, the RNGA entropy generation starts immediately.

Parameters

base	RNGA base address

24.8.2 void RNGA_Deinit (RNG_Type ∗ *base*)

This function shuts down the RNGA.

Parameters

base	RNGA base address

24.8.3 status_t RNGA_GetRandomData (RNG_Type ∗ *base,* void ∗ *data,* size_t *data_size*)

This function gets random data from the RNGA.

Parameters

base	RNGA base address
data	pointer to user buffer to be filled by random data
data_size	size of data in bytes

Returns

RNGA status

24.8.4 void RNGA_Seed (RNG_Type * *base*, uint32_t *seed*)

This function inputs an entropy value that the RNGA uses to seed its pseudo-random algorithm.

Parameters

base	RNGA base address
seed	input seed value

24.8.5 void RNGA_SetMode (RNG_Type * *base*, rnga_mode_t *mode*)

This function sets the RNGA in sleep mode or normal mode.

Parameters

base	RNGA base address
mode	normal mode or sleep mode

24.8.6 rnga_mode_t RNGA_GetMode (RNG_Type * *base*)

This function gets the RNGA working mode.

Parameters

base	RNGA base address

Returns

normal mode or sleep mode

Chapter 25
SIM: System Integration Module Driver

25.1 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))

25.2 Data Structure Documentation

25.2.1 struct sim_uid_t

Data Fields

- uint32_t H
  UIDH.
- uint32_t MH
  UIDMH.
- uint32_t ML
  UIDML.
- uint32_t L
  UIDL.

Field Documentation

(1) uint32_t sim_uid_t::H

(2) uint32_t sim_uid_t::MH

(3) uint32_t sim_uid_t::ML

(4) uint32_t sim_uid_t::L

25.3 Enumeration Type Documentation

25.3.1 enum _sim_flash_mode

Enumerator

kSIM_FlashDisableInWait Disable flash in wait mode.
kSIM_FlashDisable Disable flash in normal mode.

25.4 Function Documentation

25.4.1 void SIM_GetUniqueId (sim_uid_t ∗ *uid*)

Parameters

uid	Pointer to the structure to save the UID value.

25.4.2 static void SIM_SetFlashMode (uint8_t *mode*) [inline], [static]

Parameters

mode	The mode to set; see _sim_flash_mode for mode details.

Chapter 26
SLCD: Segment LCD Driver

26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Segment LCD (SLCD) module of MCU-Xpresso SDK devices. The SLCD module is a CMOS charge pump voltage inverter that is designed for low voltage and low-power operation. SLCD is designed to generate the appropriate waveforms to drive multiplexed numeric, alphanumeric, or custom segment LCD panels. SLCD also has several timing and control settings that can be software-configured depending on the application's requirements. Timing and control consists of registers and control logic for the following:

1. LCD frame frequency
2. Duty cycle selection
3. Front plane/back plane selection and enabling
4. Blink modes and frequency
5. Operation in low-power modes

26.2 Plane Setting and Display Control

After the SLCD general initialization, the SLCD_SetBackPlanePhase(), SLCD_SetFrontPlaneSegments(), and SLCD_SetFrontPlaneOnePhase() are used to set the special back/front Plane to make SLCD display correctly. Then, the independent display control APIs, SLCD_StartDisplay() and SLCD_StopDisplay(), start and stop the SLCD display.

The SLCD_StartBlinkMode() and SLCD_StopBlinkMode() are provided for the runtime special blink mode control. To get the SLCD fault detection result, call the SLCD_GetFaultDetectCounter().

26.3 Typical use case

26.3.1 SLCD Initialization operation

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

Data Structures

- struct slcd_fault_detect_config_t
  SLCD fault frame detection configuration structure. More...
- struct slcd_clock_config_t
  SLCD clock configuration structure. More...
- struct slcd_config_t
  SLCD configuration structure. More...

Enumerations

- enum slcd_power_supply_option_t {
  kSLCD_InternalVll3UseChargePump,
  kSLCD_ExternalVll3UseResistorBiasNetwork,
  kSLCD_ExteranlVll3UseChargePump,
  kSLCD_InternalVll1UseChargePump }
  SLCD power supply option.
- enum slcd_regulated_voltage_trim_t {
  kSLCD_RegulatedVolatgeTrim00 = 0U,
  kSLCD_RegulatedVolatgeTrim01,
  kSLCD_RegulatedVolatgeTrim02,
  kSLCD_RegulatedVolatgeTrim03,
  kSLCD_RegulatedVolatgeTrim04,
  kSLCD_RegulatedVolatgeTrim05,
  kSLCD_RegulatedVolatgeTrim06,
  kSLCD_RegulatedVolatgeTrim07,
  kSLCD_RegulatedVolatgeTrim08,
  kSLCD_RegulatedVolatgeTrim09,
  kSLCD_RegulatedVolatgeTrim10,
  kSLCD_RegulatedVolatgeTrim11,
  kSLCD_RegulatedVolatgeTrim12,
  kSLCD_RegulatedVolatgeTrim13,
  kSLCD_RegulatedVolatgeTrim14,
  kSLCD_RegulatedVolatgeTrim15 }
  SLCD regulated voltage trim parameter, be used to meet the desired contrast.
- enum slcd_load_adjust_t {
  kSLCD_LowLoadOrFastestClkSrc = 0U,
  kSLCD_LowLoadOrIntermediateClkSrc,
  kSLCD_HighLoadOrIntermediateClkSrc,
  kSLCD_HighLoadOrSlowestClkSrc }
  SLCD load adjust to handle different LCD glass capacitance or configure the LCD charge pump clock source.
- enum slcd_clock_src_t {
  kSLCD_DefaultClk = 0U,
  kSLCD_AlternateClk1 = 1U }
  SLCD clock source.
- enum slcd_alt_clock_div_t {
  kSLCD_AltClkDivFactor1 = 0U,
  kSLCD_AltClkDivFactor64,
  kSLCD_AltClkDivFactor256,
  kSLCD_AltClkDivFactor512 }
  SLCD alternate clock divider.
- enum slcd_clock_prescaler_t {

kSLCD_ClkPrescaler00 = 0U,
kSLCD_ClkPrescaler01,
kSLCD_ClkPrescaler02,
kSLCD_ClkPrescaler03,
kSLCD_ClkPrescaler04,
kSLCD_ClkPrescaler05,
kSLCD_ClkPrescaler06,
kSLCD_ClkPrescaler07 }
SLCD clock prescaler to generate frame frequency.
- enum slcd_duty_cycle_t {
kSLCD_1Div1DutyCycle = 0U,
kSLCD_1Div2DutyCycle,
kSLCD_1Div3DutyCycle,
kSLCD_1Div4DutyCycle,
kSLCD_1Div5DutyCycle,
kSLCD_1Div6DutyCycle,
kSLCD_1Div7DutyCycle,
kSLCD_1Div8DutyCycle }
SLCD duty cycle.
- enum slcd_phase_type_t {
kSLCD_NoPhaseActivate = 0x00U,
kSLCD_PhaseAActivate = 0x01U,
kSLCD_PhaseBActivate = 0x02U,
kSLCD_PhaseCActivate = 0x04U,
kSLCD_PhaseDActivate = 0x08U,
kSLCD_PhaseEActivate = 0x10U,
kSLCD_PhaseFActivate = 0x20U,
kSLCD_PhaseGActivate = 0x40U,
kSLCD_PhaseHActivate = 0x80U }
SLCD segment phase type.
- enum slcd_phase_index_t {
kSLCD_PhaseAIndex = 0x0U,
kSLCD_PhaseBIndex = 0x1U,
kSLCD_PhaseCIndex = 0x2U,
kSLCD_PhaseDIndex = 0x3U,
kSLCD_PhaseEIndex = 0x4U,
kSLCD_PhaseFIndex = 0x5U,
kSLCD_PhaseGIndex = 0x6U,
kSLCD_PhaseHIndex = 0x7U }
SLCD segment phase bit index.
- enum slcd_display_mode_t {
kSLCD_NormalMode = 0U,
kSLCD_AlternateMode,
kSLCD_BlankMode }
SLCD display mode.
- enum slcd_blink_mode_t {

kSLCD_BlankDisplayBlink = 0U,
kSLCD_AltDisplayBlink }
SLCD blink mode.
- enum slcd_blink_rate_t {
kSLCD_BlinkRate00 = 0U,
kSLCD_BlinkRate01,
kSLCD_BlinkRate02,
kSLCD_BlinkRate03,
kSLCD_BlinkRate04,
kSLCD_BlinkRate05,
kSLCD_BlinkRate06,
kSLCD_BlinkRate07 }
SLCD blink rate.
- enum slcd_fault_detect_clock_prescaler_t {
kSLCD_FaultSampleFreqDivider1 = 0U,
kSLCD_FaultSampleFreqDivider2,
kSLCD_FaultSampleFreqDivider4,
kSLCD_FaultSampleFreqDivider8,
kSLCD_FaultSampleFreqDivider16,
kSLCD_FaultSampleFreqDivider32,
kSLCD_FaultSampleFreqDivider64,
kSLCD_FaultSampleFreqDivider128 }
SLCD fault detect clock prescaler.
- enum slcd_fault_detect_sample_window_width_t {
kSLCD_FaultDetectWindowWidth4SampleClk = 0U,
kSLCD_FaultDetectWindowWidth8SampleClk,
kSLCD_FaultDetectWindowWidth16SampleClk,
kSLCD_FaultDetectWindowWidth32SampleClk,
kSLCD_FaultDetectWindowWidth64SampleClk,
kSLCD_FaultDetectWindowWidth128SampleClk,
kSLCD_FaultDetectWindowWidth256SampleClk,
kSLCD_FaultDetectWindowWidth512SampleClk }
SLCD fault detect sample window width.
- enum slcd_interrupt_enable_t {
kSLCD_FaultDetectCompleteInterrupt = 1U,
kSLCD_FrameFreqInterrupt = 2U }
SLCD interrupt source.
- enum slcd_lowpower_behavior {
kSLCD_EnabledInWaitStop = 0,
kSLCD_EnabledInWaitOnly,
kSLCD_EnabledInStopOnly,
kSLCD_DisabledInWaitStop }
SLCD behavior in low power mode.

Driver version

- #define FSL_SLCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

SLCD driver version.

Initialization and deinitialization

- void SLCD_Init (LCD_Type ∗base, slcd_config_t ∗configure)
  Initializes the SLCD, ungates the module clock, initializes the power setting, enables all used plane pins, and sets with interrupt and work mode with the configuration.
- void SLCD_Deinit (LCD_Type ∗base)
  Deinitializes the SLCD module, gates the module clock, disables an interrupt, and displays the SLCD.
- void SLCD_GetDefaultConfig (slcd_config_t ∗configure)
  Gets the SLCD default configuration structure.

Plane Setting and Display Control

- static void SLCD_StartDisplay (LCD_Type ∗base)
  Enables the SLCD controller, starts generation, and displays the front plane and back plane waveform.
- static void SLCD_StopDisplay (LCD_Type ∗base)
  Stops the SLCD controller.
- void SLCD_StartBlinkMode (LCD_Type ∗base, slcd_blink_mode_t mode, slcd_blink_rate_t rate)
  Starts the SLCD blink mode.
- static void SLCD_StopBlinkMode (LCD_Type ∗base)
  Stops the SLCD blink mode.
- static void SLCD_SetBackPlanePhase (LCD_Type ∗base, uint32_t pinIndx, slcd_phase_type_t phase)
  Sets the SLCD back plane pin phase.
- static void SLCD_SetFrontPlaneSegments (LCD_Type ∗base, uint32_t pinIndx, uint8_t operation)
  Sets the SLCD front plane segment operation for a front plane pin.
- static void SLCD_SetFrontPlaneOnePhase (LCD_Type ∗base, uint32_t pinIndx, slcd_phase_index_t phaseIndx, bool enable)
  Sets one SLCD front plane pin for one phase.
- static uint32_t SLCD_GetFaultDetectCounter (LCD_Type ∗base)
  Gets the SLCD fault detect counter.

Interrupts.

- void SLCD_EnableInterrupts (LCD_Type ∗base, uint32_t mask)
  Enables the SLCD interrupt.
- void SLCD_DisableInterrupts (LCD_Type ∗base, uint32_t mask)
  Disables the SLCD interrupt.
- uint32_t SLCD_GetInterruptStatus (LCD_Type ∗base)
  Gets the SLCD interrupt status flag.
- void SLCD_ClearInterruptStatus (LCD_Type ∗base, uint32_t mask)
  Clears the SLCD interrupt events status flag.

26.4 Data Structure Documentation

26.4.1 struct slcd_fault_detect_config_t

Data Fields

- bool faultDetectIntEnable
  Fault frame detection interrupt enable flag.
- bool faultDetectBackPlaneEnable
  True means the pin id fault detected is back plane otherwise front plane.
- uint8_t faultDetectPinIndex
  Fault detected pin id from 0 to 63.
- slcd_fault_detect_clock_prescaler_t faultPrescaler
  Fault detect clock prescaler.
- slcd_fault_detect_sample_window_width_t width
  Fault detect sample window width.

Field Documentation

(1) bool slcd_fault_detect_config_t::faultDetectIntEnable

(2) bool slcd_fault_detect_config_t::faultDetectBackPlaneEnable

(3) uint8_t slcd_fault_detect_config_t::faultDetectPinIndex

(4) slcd_fault_detect_clock_prescaler_t slcd_fault_detect_config_t::faultPrescaler

(5) slcd_fault_detect_sample_window_width_t slcd_fault_detect_config_t::width

26.4.2 struct slcd_clock_config_t

Data Fields

- slcd_clock_src_t clkSource
  Clock source.
- slcd_alt_clock_div_t altClkDivider
  The divider to divide the alternate clock used for alternate clock source.
- slcd_clock_prescaler_t clkPrescaler
  Clock prescaler.

Field Documentation

(1) slcd_clock_src_t slcd_clock_config_t::clkSource

"slcd_clock_src_t" is recommended to be used. The SLCD is optimized to operate using a 32.768kHz clock input.

(2) slcd_alt_clock_div_t slcd_clock_config_t::altClkDivider

(3) slcd_clock_prescaler_t slcd_clock_config_t::clkPrescaler

26.4.3 struct slcd_config_t

Data Fields

- slcd_power_supply_option_t powerSupply
  Power supply option.
- slcd_regulated_voltage_trim_t voltageTrim
  Regulated voltage trim used for the internal regulator VIREG to adjust to facilitate contrast control.
- slcd_clock_config_t * clkConfig
  Clock configure.
- slcd_display_mode_t displayMode
  SLCD display mode.
- slcd_load_adjust_t loadAdjust
  Load adjust to handle glass capacitance.
- slcd_duty_cycle_t dutyCycle
  Duty cycle.
- slcd_lowpower_behavior lowPowerBehavior
  SLCD behavior in low power mode.
- bool frameFreqIntEnable
  Frame frequency interrupt enable flag.
- uint32_t slcdLowPinEnabled
  Setting enabled SLCD pin 0 ∼ pin 31.
- uint32_t slcdHighPinEnabled
  Setting enabled SLCD pin 32 ∼ pin 63.
- uint32_t backPlaneLowPin
  Setting back plane pin 0 ∼ pin 31.
- uint32_t backPlaneHighPin
  Setting back plane pin 32 ∼ pin 63.
- slcd_fault_detect_config_t * faultConfig
  Fault frame detection configure.

Field Documentation

(1) slcd_power_supply_option_t slcd_config_t::powerSupply

(2) slcd_regulated_voltage_trim_t slcd_config_t::voltageTrim

(3) slcd_clock_config_t* slcd_config_t::clkConfig

(4) slcd_display_mode_t slcd_config_t::displayMode

(5) slcd_load_adjust_t slcd_config_t::loadAdjust

(6) slcd_duty_cycle_t slcd_config_t::dutyCycle

(7) slcd_lowpower_behavior slcd_config_t::lowPowerBehavior

(8) bool slcd_config_t::frameFreqIntEnable

(9) uint32_t slcd_config_t::slcdLowPinEnabled

Setting bit n to 1 means enable pin n.

(10) uint32_t slcd_config_t::slcdHighPinEnabled

Setting bit n to 1 means enable pin (n + 32).

(11) uint32_t slcd_config_t::backPlaneLowPin

Setting bit n to 1 means setting pin n as back plane. It should never have the same bit setting as the frontPlane Pin.

(12) uint32_t slcd_config_t::backPlaneHighPin

Setting bit n to 1 means setting pin (n + 32) as back plane. It should never have the same bit setting as the frontPlane Pin.

(13) slcd_fault_detect_config_t* slcd_config_t::faultConfig

If not requirement, set to NULL.

26.5 Macro Definition Documentation

26.5.1 #define FSL_SLCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

26.6 Enumeration Type Documentation

26.6.1 enum slcd_power_supply_option_t

Enumerator

kSLCD_InternalVll3UseChargePump VLL3 connected to VDD internally, charge pump is used to generate VLL1 and VLL2.

kSLCD_ExternalVll3UseResistorBiasNetwork VLL3 is driven externally and resistor bias network is used to generate VLL1 and VLL2.

kSLCD_ExteranlVll3UseChargePump VLL3 is driven externally and charge pump is used to generate VLL1 and VLL2.

kSLCD_InternalVll1UseChargePump VIREG is connected to VLL1 internally and charge pump is used to generate VLL2 and VLL3.

26.6.2 enum slcd_regulated_voltage_trim_t

Enumerator

kSLCD_RegulatedVolatgeTrim00 Increase the voltage to 0.91 V.

kSLCD_RegulatedVolatgeTrim01 Increase the voltage to 1.01 V.

kSLCD_RegulatedVolatgeTrim02 Increase the voltage to 0.96 V.
kSLCD_RegulatedVolatgeTrim03 Increase the voltage to 1.06 V.
kSLCD_RegulatedVolatgeTrim04 Increase the voltage to 0.93 V.
kSLCD_RegulatedVolatgeTrim05 Increase the voltage to 1.02 V.
kSLCD_RegulatedVolatgeTrim06 Increase the voltage to 0.98 V.
kSLCD_RegulatedVolatgeTrim07 Increase the voltage to 1.08 V.
kSLCD_RegulatedVolatgeTrim08 Increase the voltage to 0.92 V.
kSLCD_RegulatedVolatgeTrim09 Increase the voltage to 1.02 V.
kSLCD_RegulatedVolatgeTrim10 Increase the voltage to 0.97 V.
kSLCD_RegulatedVolatgeTrim11 Increase the voltage to 1.07 V.
kSLCD_RegulatedVolatgeTrim12 Increase the voltage to 0.94 V.
kSLCD_RegulatedVolatgeTrim13 Increase the voltage to 1.05 V.
kSLCD_RegulatedVolatgeTrim14 Increase the voltage to 0.99 V.
kSLCD_RegulatedVolatgeTrim15 Increase the voltage to 1.09 V.

26.6.3 enum slcd_load_adjust_t

Adjust the LCD glass capacitance if resistor bias network is enabled: kSLCD_LowLoadOrFastestClkSrc - Low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,V-LL2,Vcap1 and Vcap2 pins) kSLCD_LowLoadOrIntermediateClkSrc - low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) kSLCD-_HighLoadOrIntermediateClkSrc - high load (LCD glass capacitance 8000pF or lower. LCD or GPIO function can be used on Vcap1 and Vcap2 pins) kSLCD_HighLoadOrSlowestClkSrc - high load (LC-D glass capacitance 8000pF or lower LCD or GPIO function can be used on Vcap1 and Vcap2 pins) Adjust clock for charge pump if charge pump is enabled: kSLCD_LowLoadOrFastestClkSrc - Fasten clock source (LCD glass capacitance 8000pF or 4000pF or lower if Fast Frame Rate is set) kSLCD_-LowLoadOrIntermediateClkSrc - Intermediate clock source (LCD glass capacitance 4000pF or 2000pF or lower if Fast Frame Rate is set) kSLCD_HighLoadOrIntermediateClkSrc - Intermediate clock source (LCD glass capacitance 2000pF or 1000pF or lower if Fast Frame Rate is set) kSLCD_HighLoadOr-SlowestClkSrc - slowest clock source (LCD glass capacitance 1000pF or 500pF or lower if Fast Frame Rate is set)

Enumerator

kSLCD_LowLoadOrFastestClkSrc Adjust in low load or selects fastest clock.
kSLCD_LowLoadOrIntermediateClkSrc Adjust in low load or selects intermediate clock.
kSLCD_HighLoadOrIntermediateClkSrc Adjust in high load or selects intermediate clock.
kSLCD_HighLoadOrSlowestClkSrc Adjust in high load or selects slowest clock.

26.6.4 enum slcd_clock_src_t

Enumerator

kSLCD_DefaultClk Select default clock ERCLK32K.

kSLCD_AlternateClk1 Select alternate clock source 1 : MCGIRCLK.

26.6.5 enum slcd_alt_clock_div_t

Enumerator

kSLCD_AltClkDivFactor1 No divide for alternate clock.
kSLCD_AltClkDivFactor64 Divide alternate clock with factor 64.
kSLCD_AltClkDivFactor256 Divide alternate clock with factor 256.
kSLCD_AltClkDivFactor512 Divide alternate clock with factor 512.

26.6.6 enum slcd_clock_prescaler_t

Enumerator

kSLCD_ClkPrescaler00 Prescaler 0.
kSLCD_ClkPrescaler01 Prescaler 1.
kSLCD_ClkPrescaler02 Prescaler 2.
kSLCD_ClkPrescaler03 Prescaler 3.
kSLCD_ClkPrescaler04 Prescaler 4.
kSLCD_ClkPrescaler05 Prescaler 5.
kSLCD_ClkPrescaler06 Prescaler 6.
kSLCD_ClkPrescaler07 Prescaler 7.

26.6.7 enum slcd_duty_cycle_t

Enumerator

kSLCD_1Div1DutyCycle LCD use 1 BP 1/1 duty cycle.
kSLCD_1Div2DutyCycle LCD use 2 BP 1/2 duty cycle.
kSLCD_1Div3DutyCycle LCD use 3 BP 1/3 duty cycle.
kSLCD_1Div4DutyCycle LCD use 4 BP 1/4 duty cycle.
kSLCD_1Div5DutyCycle LCD use 5 BP 1/5 duty cycle.
kSLCD_1Div6DutyCycle LCD use 6 BP 1/6 duty cycle.
kSLCD_1Div7DutyCycle LCD use 7 BP 1/7 duty cycle.
kSLCD_1Div8DutyCycle LCD use 8 BP 1/8 duty cycle.

26.6.8 enum slcd_phase_type_t

Enumerator

kSLCD_NoPhaseActivate LCD wareform no phase activates.
kSLCD_PhaseAActivate LCD waveform phase A activates.
kSLCD_PhaseBActivate LCD waveform phase B activates.
kSLCD_PhaseCActivate LCD waveform phase C activates.
kSLCD_PhaseDActivate LCD waveform phase D activates.
kSLCD_PhaseEActivate LCD waveform phase E activates.
kSLCD_PhaseFActivate LCD waveform phase F activates.
kSLCD_PhaseGActivate LCD waveform phase G activates.
kSLCD_PhaseHActivate LCD waveform phase H activates.

26.6.9 enum slcd_phase_index_t

Enumerator

kSLCD_PhaseAIndex LCD phase A bit index.
kSLCD_PhaseBIndex LCD phase B bit index.
kSLCD_PhaseCIndex LCD phase C bit index.
kSLCD_PhaseDIndex LCD phase D bit index.
kSLCD_PhaseEIndex LCD phase E bit index.
kSLCD_PhaseFIndex LCD phase F bit index.
kSLCD_PhaseGIndex LCD phase G bit index.
kSLCD_PhaseHIndex LCD phase H bit index.

26.6.10 enum slcd_display_mode_t

Enumerator

kSLCD_NormalMode LCD Normal display mode.
kSLCD_AlternateMode LCD Alternate display mode. For four back planes or less.
kSLCD_BlankMode LCD Blank display mode.

26.6.11 enum slcd_blink_mode_t

Enumerator

kSLCD_BlankDisplayBlink Display blank during the blink period.
kSLCD_AltDisplayBlink Display alternate display during the blink period if duty cycle is lower than 5.

26.6.12 enum slcd_blink_rate_t

Enumerator

kSLCD_BlinkRate00 SLCD blink rate is LCD clock/((2^12)).
kSLCD_BlinkRate01 SLCD blink rate is LCD clock/((2^13)).
kSLCD_BlinkRate02 SLCD blink rate is LCD clock/((2^14)).
kSLCD_BlinkRate03 SLCD blink rate is LCD clock/((2^15)).
kSLCD_BlinkRate04 SLCD blink rate is LCD clock/((2^16)).
kSLCD_BlinkRate05 SLCD blink rate is LCD clock/((2^17)).
kSLCD_BlinkRate06 SLCD blink rate is LCD clock/((2^18)).
kSLCD_BlinkRate07 SLCD blink rate is LCD clock/((2^19)).

26.6.13 enum slcd_fault_detect_clock_prescaler_t

Enumerator

kSLCD_FaultSampleFreqDivider1 Fault detect sample clock frequency is 1/1 bus clock.
kSLCD_FaultSampleFreqDivider2 Fault detect sample clock frequency is 1/2 bus clock.
kSLCD_FaultSampleFreqDivider4 Fault detect sample clock frequency is 1/4 bus clock.
kSLCD_FaultSampleFreqDivider8 Fault detect sample clock frequency is 1/8 bus clock.
kSLCD_FaultSampleFreqDivider16 Fault detect sample clock frequency is 1/16 bus clock.
kSLCD_FaultSampleFreqDivider32 Fault detect sample clock frequency is 1/32 bus clock.
kSLCD_FaultSampleFreqDivider64 Fault detect sample clock frequency is 1/64 bus clock.
kSLCD_FaultSampleFreqDivider128 Fault detect sample clock frequency is 1/128 bus clock.

26.6.14 enum slcd_fault_detect_sample_window_width_t

Enumerator

kSLCD_FaultDetectWindowWidth4SampleClk Sample window width is 4 sample clock cycles.
kSLCD_FaultDetectWindowWidth8SampleClk Sample window width is 8 sample clock cycles.
kSLCD_FaultDetectWindowWidth16SampleClk Sample window width is 16 sample clock cycles.
kSLCD_FaultDetectWindowWidth32SampleClk Sample window width is 32 sample clock cycles.
kSLCD_FaultDetectWindowWidth64SampleClk Sample window width is 64 sample clock cycles.
kSLCD_FaultDetectWindowWidth128SampleClk Sample window width is 128 sample clock cycles.
kSLCD_FaultDetectWindowWidth256SampleClk Sample window width is 256 sample clock cycles.
kSLCD_FaultDetectWindowWidth512SampleClk Sample window width is 512 sample clock cycles.

26.6.15 enum slcd_interrupt_enable_t

Enumerator

kSLCD_FaultDetectCompleteInterrupt SLCD fault detection complete interrupt source.
kSLCD_FrameFreqInterrupt SLCD frame frequency interrupt source. Not available in all low-power modes.

26.6.16 enum slcd_lowpower_behavior

Enumerator

kSLCD_EnabledInWaitStop SLCD works in wait and stop mode.
kSLCD_EnabledInWaitOnly SLCD works in wait mode and is disabled in stop mode.
kSLCD_EnabledInStopOnly SLCD works in stop mode and is disabled in wait mode.
kSLCD_DisabledInWaitStop SLCD is disabled in stop mode and wait mode.

26.7 Function Documentation

26.7.1 void SLCD_Init (LCD_Type * *base,* slcd_config_t * *configure*)

Parameters

base	SLCD peripheral base address.
configure	SLCD configuration pointer. For the configuration structure, many parameters have the default setting and the SLCD_Getdefaultconfig() is provided to get them. Use it verified for their applications. The others have no default settings, such as "clk-Config", and must be provided by the application before calling the SLCD_Init() API.

26.7.2 void SLCD_Deinit (LCD_Type * *base*)

Parameters

base	SLCD peripheral base address.

26.7.3 void SLCD_GetDefaultConfig (slcd_config_t * *configure*)

The purpose of this API is to get default parameters of the configuration structure for the SLCD_Init(). Use these initialized parameters unchanged in SLCD_Init() or modify fields of the structure before the calling SLCD_Init(). All default parameters of the configure structuration are listed.

```
config.displayMode         = kSLCD_NormalMode;
config.powerSupply         = kSLCD_InternalVll3UseChargePump;
config.voltageTrim         = kSLCD_RegulatedVolatgeTrim00;
config.lowPowerBehavior    = kSLCD_EnabledInWaitStop;
config.interruptSrc        = 0;
config.faultConfig         = NULL;
config.frameFreqIntEnable =  false;
```

Parameters

configure	The SLCD configuration structure pointer.

26.7.4 static void SLCD_StartDisplay (LCD_Type ∗ *base*) [inline], [static]

Parameters

base	SLCD peripheral base address.

26.7.5 static void SLCD_StopDisplay (LCD_Type ∗ *base*) [inline], [static]

There is no waveform generator and all enabled pins only output a low value.

Parameters

base	SLCD peripheral base address.

26.7.6 void SLCD_StartBlinkMode (LCD_Type ∗ *base*, slcd_blink_mode_t *mode*, slcd_blink_rate_t *rate*)

Parameters

base	SLCD peripheral base address.
mode	SLCD blink mode.
rate	SLCD blink rate.

26.7.7 static void SLCD_StopBlinkMode (LCD_Type ∗ *base*) [inline], [static]

Parameters

base	SLCD peripheral base address.

26.7.8 static void SLCD_SetBackPlanePhase (LCD_Type * *base*, uint32_t *pinIndx*, slcd_phase_type_t *phase*) [inline], [static]

This function sets the SLCD back plane pin phase. "kSLCD_PhaseXActivate" setting means the phase X is active for the back plane pin. "kSLCD_NoPhaseActivate" setting means there is no phase active for the back plane pin. For example, set the back plane pin 20 for phase A.

```
* SLCD_SetBackPlanePhase(LCD, 20, kSLCD_PhaseAActivate);
*
```

Parameters

base	SLCD peripheral base address.
pinIndx	SLCD back plane pin index. Range from 0 to 63.
phase	The phase activates for the back plane pin.

26.7.9 static void SLCD_SetFrontPlaneSegments (LCD_Type * *base*, uint32_t *pinIndx*, uint8_t *operation*) [inline], [static]

This function sets the SLCD front plane segment on or off operation. Each bit turns on or off the segments associated with the front plane pin in the following pattern: HGFEDCBA (most significant bit controls segment H and least significant bit controls segment A). For example, turn on the front plane pin 20 for phase B and phase C.

```
* SLCD_SetFrontPlaneSegments(LCD, 20, (
      kSLCD_PhaseBActivate | kSLCD_PhaseCActivate));
*
```

Parameters

base	SLCD peripheral base address.

pinIndx	SLCD back plane pin index. Range from 0 to 63.
operation	The operation for the segment on the front plane pin. This is a logical OR of the enumeration :: slcd_phase_type_t.

26.7.10 static void SLCD_SetFrontPlaneOnePhase (LCD_Type * *base*, uint32_t *pinIndx*, slcd_phase_index_t *phaseIndx*, bool *enable*) [inline], [static]

This function can be used to set one phase on or off for the front plane pin. It can be call many times to set the plane pin for different phase indexes. For example, turn on the front plane pin 20 for phase B and phase C.

```
* SLCD_SetFrontPlaneOnePhase(LCD, 20,
      kSLCD_PhaseBIndex, true);
* SLCD_SetFrontPlaneOnePhase(LCD, 20,
      kSLCD_PhaseCIndex, true);
*
```

Parameters

base	SLCD peripheral base address.
pinIndx	SLCD back plane pin index. Range from 0 to 63.
phaseIndx	The phase bit index slcd_phase_index_t.
enable	True to turn on the segment for phaseIndx phase false to turn off the segment for phaseIndx phase.

26.7.11 static uint32_t SLCD_GetFaultDetectCounter (LCD_Type * *base*) [inline], [static]

This function gets the number of samples inside the fault detection sample window.

Parameters

base	SLCD peripheral base address.

Returns

The fault detect counter. The maximum return value is 255. If the maximum 255 returns, the overflow may happen. Reconfigure the fault detect sample window and fault detect clock prescaler for proper sampling.

26.7.12 void SLCD_EnableInterrupts (LCD_Type * *base*, uint32_t *mask*)

For example, to enable fault detect complete interrupt and frame frequency interrupt, for FSL_FEATURE_SLCD_HAS_FRAME_FREQUENCY_INTERRUPT enabled case, do the following.

```
*     SLCD_EnableInterrupts(LCD,
      kSLCD_FaultDetectCompleteInterrupt |
      kSLCD_FrameFreqInterrupt);
*
```

Parameters

base	SLCD peripheral base address.
mask	SLCD interrupts to enable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t.

26.7.13 void SLCD_DisableInterrupts (LCD_Type * *base*, uint32_t *mask*)

For example, to disable fault detect complete interrupt and frame frequency interrupt, for FSL_FEATURE_SLCD_HAS_FRAME_FREQUENCY_INTERRUPT enabled case, do the following.

```
*     SLCD_DisableInterrupts(LCD,
      kSLCD_FaultDetectCompleteInterrupt |
      kSLCD_FrameFreqInterrupt);
*
```

Parameters

base	SLCD peripheral base address.
mask	SLCD interrupts to disable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t.

26.7.14 uint32_t SLCD_GetInterruptStatus (LCD_Type * *base*)

Parameters

base	SLCD peripheral base address.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: slcd_interrupt_enable_t.

26.7.15 void SLCD_ClearInterruptStatus (LCD_Type * *base*, uint32_t *mask*)

Parameters

base	SLCD peripheral base address.
mask	SLCD interrupt source to be cleared. This is the logical OR of members of the enumeration :: slcd_interrupt_enable_t.

Chapter 27
SMC: System Mode Controller Driver

27.1 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.

27.2 Typical use case

27.2.1 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_power_mode_vlls_config_t
  SMC Very Low-Leakage Stop power mode configuration. More...

Enumerations

- enum smc_power_mode_protection_t {
  kSMC_AllowPowerModeVlls = SMC_PMPROT_AVLLS_MASK,
  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,
  kSMC_PowerStateVlls = 0x01U << 6U }
  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,
  kSMC_StopVlls = 4U }
  Stop mode definition.
- enum smc_stop_submode_t {
  kSMC_StopSub0 = 0U,
  kSMC_StopSub1 = 1U,
  kSMC_StopSub2 = 2U,
  kSMC_StopSub3 = 3U }
  VLLS/LLS stop sub 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_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.
- status_t SMC_SetPowerModeVlls (SMC_Type ∗base, const smc_power_mode_vlls_config_t ∗config)
  Configures the system to VLLS power mode.

27.3 Data Structure Documentation

27.3.1 struct smc_power_mode_vlls_config_t

Data Fields

- smc_stop_submode_t subMode
  Very Low-leakage Stop sub-mode.
- bool enablePorDetectInVlls0
  Enable Power on reset detect in VLLS mode.

27.4 Enumeration Type Documentation

27.4.1 enum smc_power_mode_protection_t

Enumerator

kSMC_AllowPowerModeVlls Allow Very-low-leakage Stop Mode.
kSMC_AllowPowerModeVlp Allow Very-Low-power Mode.
kSMC_AllowPowerModeAll Allow all power mode.

27.4.2 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
kSMC_PowerStateVlls 0100_0000 - Current power mode is VLLS

27.4.3 enum smc_run_mode_t

Enumerator

kSMC_RunNormal Normal RUN mode.
kSMC_RunVlpr Very-low-power RUN mode.

27.4.4 enum smc_stop_mode_t

Enumerator

kSMC_StopNormal Normal STOP mode.
kSMC_StopVlps Very-low-power STOP mode.
kSMC_StopVlls Very-low-leakage Stop mode.

27.4.5 enum smc_stop_submode_t

Enumerator

kSMC_StopSub0 Stop submode 0, for VLLS0/LLS0.
kSMC_StopSub1 Stop submode 1, for VLLS1/LLS1.
kSMC_StopSub2 Stop submode 2, for VLLS2/LLS2.
kSMC_StopSub3 Stop submode 3, for VLLS3/LLS3.

27.4.6 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.

27.4.7 anonymous enum

Enumerator

kStatus_SMC_StopAbort Entering Stop mode is abort.

27.5 Function Documentation

27.5.1 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.

27.5.2 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.

27.5.3 void SMC_PreEnterStopModes (void)

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

27.5.4 void SMC_PostExitStopModes (void)

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with SMC-_PreEnterStopModes.

27.5.5 void SMC_PreEnterWaitModes (void)

This function should be called before entering WAIT/VLPW modes.

27.5.6 void SMC_PostExitWaitModes (void)

This function should be called after wake up from WAIT/VLPW modes. It is used with SMC_PreEnter-WaitModes.

27.5.7 status_t SMC_SetPowerModeRun (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

27.5.8 status_t SMC_SetPowerModeWait (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

27.5.9 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.

27.5.10 status_t SMC_SetPowerModeVlpr (SMC_Type ∗ *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

27.5.11 status_t SMC_SetPowerModeVlpw (SMC_Type ∗ *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

27.5.12 status_t SMC_SetPowerModeVlps (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

27.5.13 status_t SMC_SetPowerModeVlls (SMC_Type * *base*, const smc_power_mode_vlls_config_t * *config*)

Parameters

base	SMC peripheral base address.
config	The VLLS power mode configuration structure.

Returns

SMC configuration error code.

Chapter 28
SPI: Serial Peripheral Interface Driver

28.1 Overview

Modules

- SPI CMSIS driver
- SPI DMA Driver
- SPI Driver
- SPI FreeRTOS driver

28.2 SPI Driver

28.2.1 Overview

SPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional A-PI implementation and write a custom code. All transactional APIs use the spi_handle_t as the first parameter. Initialize the handle by calling the SPI_MasterTransferCreateHandle() or SPI_SlaveTransfer-CreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SPI_MasterTransferNon-Blocking() and SPI_SlaveTransferNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_SPI_Idle status.

28.2.2 Typical use case

28.2.2.1 SPI master transfer using an interrupt method

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

28.2.2.2 SPI Send/receive using a DMA method

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

Data Structures

- struct spi_master_config_t
  SPI master user configure structure. More...
- struct spi_slave_config_t
  SPI slave user configure structure. More...
- struct spi_transfer_t
  SPI transfer structure. More...
- struct spi_master_handle_t
  SPI transfer handle structure. More...

Macros

- #define SPI_DUMMYDATA (0xFFU)
  SPI dummy transfer data, the data is sent while txBuff is NULL.
- #define SPI_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef spi_master_handle_t spi_slave_handle_t
  Slave handle is the same with master handle.
- typedef void(∗ spi_master_callback_t)(SPI_Type ∗base, spi_master_handle_t ∗handle, status_t status, void ∗userData)
  SPI master callback for finished transmit.
- typedef void(∗ spi_slave_callback_t)(SPI_Type ∗base, spi_slave_handle_t ∗handle, status_t status, void ∗userData)
  SPI master callback for finished transmit.

Enumerations

- enum {
  kStatus_SPI_Busy = MAKE_STATUS(kStatusGroup_SPI, 0),
  kStatus_SPI_Idle = MAKE_STATUS(kStatusGroup_SPI, 1),
  kStatus_SPI_Error = MAKE_STATUS(kStatusGroup_SPI, 2),
  kStatus_SPI_Timeout = MAKE_STATUS(kStatusGroup_SPI, 3) }
  Return status for the SPI driver.
- enum spi_clock_polarity_t {
  kSPI_ClockPolarityActiveHigh = 0x0U,
  kSPI_ClockPolarityActiveLow }
  SPI clock polarity configuration.
- enum spi_clock_phase_t {
  kSPI_ClockPhaseFirstEdge = 0x0U,
  kSPI_ClockPhaseSecondEdge }
  SPI clock phase configuration.
- enum spi_shift_direction_t {
  kSPI_MsbFirst = 0x0U,
  kSPI_LsbFirst }
  SPI data shifter direction options.
- enum spi_ss_output_mode_t {
  kSPI_SlaveSelectAsGpio = 0x0U,
  kSPI_SlaveSelectFaultInput = 0x2U,
  kSPI_SlaveSelectAutomaticOutput = 0x3U }
  SPI slave select output mode options.
- enum spi_pin_mode_t {

kSPI_PinModeNormal = 0x0U,
kSPI_PinModeInput = 0x1U,
kSPI_PinModeOutput = 0x3U }
SPI pin mode options.

- enum spi_data_bitcount_mode_t {
kSPI_8BitMode = 0x0U,
kSPI_16BitMode }
SPI data length mode options.
- enum _spi_interrupt_enable {
kSPI_RxFullAndModfInterruptEnable = 0x1U,
kSPI_TxEmptyInterruptEnable = 0x2U,
kSPI_MatchInterruptEnable = 0x4U,
kSPI_RxFifoNearFullInterruptEnable = 0x8U,
kSPI_TxFifoNearEmptyInterruptEnable = 0x10U }
SPI interrupt sources.
- enum _spi_flags {
kSPI_RxBufferFullFlag = SPI_S_SPRF_MASK,
kSPI_MatchFlag = SPI_S_SPMF_MASK,
kSPI_TxBufferEmptyFlag = SPI_S_SPTEF_MASK,
kSPI_ModeFaultFlag = SPI_S_MODF_MASK,
kSPI_RxFifoNearFullFlag = SPI_S_RNFULLF_MASK,
kSPI_TxFifoNearEmptyFlag = SPI_S_TNEAREF_MASK,
kSPI_TxFifoFullFlag = SPI_S_TXFULLF_MASK,
kSPI_RxFifoEmptyFlag = SPI_S_RFIFOEF_MASK,
kSPI_TxFifoError = SPI_CI_TXFERR_MASK << 8U,
kSPI_RxFifoError = SPI_CI_RXFERR_MASK << 8U,
kSPI_TxOverflow = SPI_CI_TXFOF_MASK << 8U,
kSPI_RxOverflow = SPI_CI_RXFOF_MASK << 8U }
SPI status flags.
- enum spi_w1c_interrupt_t {
kSPI_RxFifoFullClearInterrupt = SPI_CI_SPRFCI_MASK,
kSPI_TxFifoEmptyClearInterrupt = SPI_CI_SPTEFCI_MASK,
kSPI_RxNearFullClearInterrupt = SPI_CI_RNFULLFCI_MASK,
kSPI_TxNearEmptyClearInterrupt = SPI_CI_TNEAREFCI_MASK }
SPI FIFO write-1-to-clear interrupt flags.
- enum spi_txfifo_watermark_t {
kSPI_TxFifoOneFourthEmpty = 0,
kSPI_TxFifoOneHalfEmpty = 1 }
SPI TX FIFO watermark settings.
- enum spi_rxfifo_watermark_t {
kSPI_RxFifoThreeFourthsFull = 0,
kSPI_RxFifoOneHalfFull = 1 }
SPI RX FIFO watermark settings.
- enum _spi_dma_enable_t {
kSPI_TxDmaEnable = SPI_C2_TXDMAE_MASK,
kSPI_RxDmaEnable = SPI_C2_RXDMAE_MASK,

kSPI_DmaAllEnable = (SPI_C2_TXDMAE_MASK | SPI_C2_RXDMAE_MASK) }
SPI DMA source.

Variables

- volatile uint8_t g_spiDummyData [ ]
  Global variable for dummy data value setting.

Driver version

- #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  SPI driver version.

Initialization and deinitialization

- void SPI_MasterGetDefaultConfig (spi_master_config_t ∗config)
  Sets the SPI master configuration structure to default values.
- void SPI_MasterInit (SPI_Type ∗base, const spi_master_config_t ∗config, uint32_t srcClock_Hz)
  Initializes the SPI with master configuration.
- void SPI_SlaveGetDefaultConfig (spi_slave_config_t ∗config)
  Sets the SPI slave configuration structure to default values.
- void SPI_SlaveInit (SPI_Type ∗base, const spi_slave_config_t ∗config)
  Initializes the SPI with slave configuration.
- void SPI_Deinit (SPI_Type ∗base)
  De-initializes the SPI.
- static void SPI_Enable (SPI_Type ∗base, bool enable)
  Enables or disables the SPI.

Status

- uint32_t SPI_GetStatusFlags (SPI_Type ∗base)
  Gets the status flag.
- static void SPI_ClearInterrupt (SPI_Type ∗base, uint8_t mask)
  Clear the interrupt if enable INCTLR.

Interrupts

- void SPI_EnableInterrupts (SPI_Type ∗base, uint32_t mask)
  Enables the interrupt for the SPI.
- void SPI_DisableInterrupts (SPI_Type ∗base, uint32_t mask)
  Disables the interrupt for the SPI.

DMA Control

- static void SPI_EnableDMA (SPI_Type ∗base, uint8_t mask, bool enable)
  Enables the DMA source for SPI.
- static uint32_t SPI_GetDataRegisterAddress (SPI_Type ∗base)
  Gets the SPI tx/rx data register address.

Bus Operations

- uint32_t SPI_GetInstance (SPI_Type ∗base)
  Get the instance for SPI module.
- static void SPI_SetPinMode (SPI_Type ∗base, spi_pin_mode_t pinMode)
  Sets the pin mode for transfer.
- void SPI_MasterSetBaudRate (SPI_Type ∗base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the baud rate for SPI transfer.
- static void SPI_SetMatchData (SPI_Type ∗base, uint32_t matchData)
  Sets the match data for SPI.
- void SPI_EnableFIFO (SPI_Type ∗base, bool enable)
  Enables or disables the FIFO if there is a FIFO.
- status_t SPI_WriteBlocking (SPI_Type ∗base, uint8_t ∗buffer, size_t size)
  Sends a buffer of data bytes using a blocking method.
- void SPI_WriteData (SPI_Type ∗base, uint16_t data)
  Writes a data into the SPI data register.
- uint16_t SPI_ReadData (SPI_Type ∗base)
  Gets a data from the SPI data register.
- void SPI_SetDummyData (SPI_Type ∗base, uint8_t dummyData)
  Set up the dummy data.

Transactional

- void SPI_MasterTransferCreateHandle (SPI_Type ∗base, spi_master_handle_t ∗handle, spi_master_callback_t callback, void ∗userData)
  Initializes the SPI master handle.
- status_t SPI_MasterTransferBlocking (SPI_Type ∗base, spi_transfer_t ∗xfer)
  Transfers a block of data using a polling method.
- status_t SPI_MasterTransferNonBlocking (SPI_Type ∗base, spi_master_handle_t ∗handle, spi_transfer_t ∗xfer)
  Performs a non-blocking SPI interrupt transfer.
- status_t SPI_MasterTransferGetCount (SPI_Type ∗base, spi_master_handle_t ∗handle, size_t ∗count)
  Gets the bytes of the SPI interrupt transferred.
- void SPI_MasterTransferAbort (SPI_Type ∗base, spi_master_handle_t ∗handle)
  Aborts an SPI transfer using interrupt.
- void SPI_MasterTransferHandleIRQ (SPI_Type ∗base, spi_master_handle_t ∗handle)
  Interrupts the handler for the SPI.
- void SPI_SlaveTransferCreateHandle (SPI_Type ∗base, spi_slave_handle_t ∗handle, spi_slave_callback_t callback, void ∗userData)
  Initializes the SPI slave handle.

- status_t SPI_SlaveTransferNonBlocking (SPI_Type ∗base, spi_slave_handle_t ∗handle, spi_transfer_t ∗xfer)
  Performs a non-blocking SPI slave interrupt transfer.
- static status_t SPI_SlaveTransferGetCount (SPI_Type ∗base, spi_slave_handle_t ∗handle, size_t ∗count)
  Gets the bytes of the SPI interrupt transferred.
- static void SPI_SlaveTransferAbort (SPI_Type ∗base, spi_slave_handle_t ∗handle)
  Aborts an SPI slave transfer using interrupt.
- void SPI_SlaveTransferHandleIRQ (SPI_Type ∗base, spi_slave_handle_t ∗handle)
  Interrupts a handler for the SPI slave.

28.2.3 Data Structure Documentation

28.2.3.1 struct spi_master_config_t

Data Fields

- bool enableMaster
  Enable SPI at initialization time.
- bool enableStopInWaitMode
  SPI stop in wait mode.
- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- spi_data_bitcount_mode_t dataMode
  8bit or 16bit mode
- spi_txfifo_watermark_t txWatermark
  Tx watermark settings.
- spi_rxfifo_watermark_t rxWatermark
  Rx watermark settings.
- spi_ss_output_mode_t outputMode
  SS pin setting.
- spi_pin_mode_t pinMode
  SPI pin mode select.
- uint32_t baudRate_Bps
  Baud Rate for SPI in Hz.

28.2.3.2 struct spi_slave_config_t

Data Fields

- bool enableSlave
  Enable SPI at initialization time.
- bool enableStopInWaitMode
  SPI stop in wait mode.

- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- spi_data_bitcount_mode_t dataMode
  8bit or 16bit mode
- spi_txfifo_watermark_t txWatermark
  Tx watermark settings.
- spi_rxfifo_watermark_t rxWatermark
  Rx watermark settings.
- spi_pin_mode_t pinMode
  SPI pin mode select.

28.2.3.3 struct 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, useless to SPI.

Field Documentation

(1) uint32_t spi_transfer_t::flags

28.2.3.4 struct _spi_master_handle

Data Fields

- uint8_t ∗volatile txData
  Transfer buffer.
- uint8_t ∗volatile rxData
  Receive buffer.
- volatile size_t txRemainingBytes
  Send data remaining in bytes.
- volatile size_t rxRemainingBytes
  Receive data remaining in bytes.
- volatile uint32_t state
  SPI internal state.
- size_t transferSize
  Bytes to be transferred.
- uint8_t bytePerFrame
  SPI mode, 2bytes or 1byte in a frame.

- uint8_t watermark
  Watermark value for SPI transfer.
- spi_master_callback_t callback
  SPI callback.
- void * userData
  Callback parameter.

28.2.4 Macro Definition Documentation

28.2.4.1 #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

28.2.4.2 #define SPI_DUMMYDATA (0xFFU)

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

28.2.5 Enumeration Type Documentation

28.2.5.1 anonymous enum

Enumerator

kStatus_SPI_Busy SPI bus is busy.
kStatus_SPI_Idle SPI is idle.
kStatus_SPI_Error SPI error.
kStatus_SPI_Timeout SPI timeout polling status flags.

28.2.5.2 enum spi_clock_polarity_t

Enumerator

kSPI_ClockPolarityActiveHigh Active-high SPI clock (idles low).
kSPI_ClockPolarityActiveLow Active-low SPI clock (idles high).

28.2.5.3 enum spi_clock_phase_t

Enumerator

kSPI_ClockPhaseFirstEdge First edge on SPSCK occurs at the middle of the first cycle of a data transfer.
kSPI_ClockPhaseSecondEdge First edge on SPSCK occurs at the start of the first cycle of a data transfer.

28.2.5.4 enum spi_shift_direction_t

Enumerator

kSPI_MsbFirst Data transfers start with most significant bit.
kSPI_LsbFirst Data transfers start with least significant bit.

28.2.5.5 enum spi_ss_output_mode_t

Enumerator

kSPI_SlaveSelectAsGpio Slave select pin configured as GPIO.
kSPI_SlaveSelectFaultInput Slave select pin configured for fault detection.
kSPI_SlaveSelectAutomaticOutput Slave select pin configured for automatic SPI output.

28.2.5.6 enum spi_pin_mode_t

Enumerator

kSPI_PinModeNormal Pins operate in normal, single-direction mode.
kSPI_PinModeInput Bidirectional mode. Master: MOSI pin is input; Slave: MISO pin is input.
kSPI_PinModeOutput Bidirectional mode. Master: MOSI pin is output; Slave: MISO pin is output.

28.2.5.7 enum spi_data_bitcount_mode_t

Enumerator

kSPI_8BitMode 8-bit data transmission mode
kSPI_16BitMode 16-bit data transmission mode

28.2.5.8 enum _spi_interrupt_enable

Enumerator

kSPI_RxFullAndModfInterruptEnable Receive buffer full (SPRF) and mode fault (MODF) interrupt.
kSPI_TxEmptyInterruptEnable Transmit buffer empty interrupt.
kSPI_MatchInterruptEnable Match interrupt.
kSPI_RxFifoNearFullInterruptEnable Receive FIFO nearly full interrupt.
kSPI_TxFifoNearEmptyInterruptEnable Transmit FIFO nearly empty interrupt.

28.2.5.9 enum _spi_flags

Enumerator

kSPI_RxBufferFullFlag Read buffer full flag.
kSPI_MatchFlag Match flag.
kSPI_TxBufferEmptyFlag Transmit buffer empty flag.
kSPI_ModeFaultFlag Mode fault flag.
kSPI_RxFifoNearFullFlag Rx FIFO near full.
kSPI_TxFifoNearEmptyFlag Tx FIFO near empty.
kSPI_TxFifoFullFlag Tx FIFO full.
kSPI_RxFifoEmptyFlag Rx FIFO empty.
kSPI_TxFifoError Tx FIFO error.
kSPI_RxFifoError Rx FIFO error.
kSPI_TxOverflow Tx FIFO Overflow.
kSPI_RxOverflow Rx FIFO Overflow.

28.2.5.10 enum spi_w1c_interrupt_t

Enumerator

kSPI_RxFifoFullClearInterrupt Receive FIFO full interrupt.
kSPI_TxFifoEmptyClearInterrupt Transmit FIFO empty interrupt.
kSPI_RxNearFullClearInterrupt Receive FIFO nearly full interrupt.
kSPI_TxNearEmptyClearInterrupt Transmit FIFO nearly empty interrupt.

28.2.5.11 enum spi_txfifo_watermark_t

Enumerator

kSPI_TxFifoOneFourthEmpty SPI tx watermark at 1/4 FIFO size.
kSPI_TxFifoOneHalfEmpty SPI tx watermark at 1/2 FIFO size.

28.2.5.12 enum spi_rxfifo_watermark_t

Enumerator

kSPI_RxFifoThreeFourthsFull SPI rx watermark at 3/4 FIFO size.
kSPI_RxFifoOneHalfFull SPI rx watermark at 1/2 FIFO size.

28.2.5.13 enum _spi_dma_enable_t

Enumerator

kSPI_TxDmaEnable Tx DMA request source.
kSPI_RxDmaEnable Rx DMA request source.
kSPI_DmaAllEnable All DMA request source.

28.2.6 Function Documentation

28.2.6.1 void SPI_MasterGetDefaultConfig (spi_master_config_t * *config*)

The purpose of this API is to get the configuration structure initialized for use in SPI_MasterInit(). User may use the initialized structure unchanged in SPI_MasterInit(), or modify some fields of the structure before calling SPI_MasterInit(). After calling this API, the master is ready to transfer. Example:

```
spi_master_config_t config;
SPI_MasterGetDefaultConfig(&config);
```

Parameters

config	pointer to master config structure

28.2.6.2 void SPI_MasterInit (SPI_Type * *base,* const spi_master_config_t * *config,* uint32_t *srcClock_Hz*)

The configuration structure can be filled by user from scratch, or be set with default values by SPI_Master-GetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
spi_master_config_t config = {
.baudRate_Bps = 400000,
...
};
SPI_MasterInit(SPI0, &config);
```

Parameters

base	SPI base pointer

config	pointer to master configuration structure
srcClock_Hz	Source clock frequency.

28.2.6.3 void SPI_SlaveGetDefaultConfig (spi_slave_config_t * *config*)

The purpose of this API is to get the configuration structure initialized for use in SPI_SlaveInit(). Modify some fields of the structure before calling SPI_SlaveInit(). Example:

```
spi_slave_config_t config;
SPI_SlaveGetDefaultConfig(&config);
```

Parameters

config	pointer to slave configuration structure

28.2.6.4 void SPI_SlaveInit (SPI_Type * *base*, const spi_slave_config_t * *config*)

The configuration structure can be filled by user from scratch or be set with default values by SPI_Slave-GetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
spi_slave_config_t config = {
.polarity = kSPIClockPolarity_ActiveHigh;
.phase = kSPIClockPhase_FirstEdge;
.direction = kSPIMsbFirst;
...
};
SPI_MasterInit(SPI0, &config);
```

Parameters

base	SPI base pointer
config	pointer to master configuration structure

28.2.6.5 void SPI_Deinit (SPI_Type * *base*)

Calling this API resets the SPI module, gates the SPI clock. The SPI module can't work unless calling the SPI_MasterInit/SPI_SlaveInit to initialize module.

Parameters

base	SPI base pointer

28.2.6.6 static void SPI_Enable (SPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	SPI base pointer
enable	pass true to enable module, false to disable module

28.2.6.7 uint32_t SPI_GetStatusFlags (SPI_Type * *base*)

Parameters

base	SPI base pointer

Returns

SPI Status, use status flag to AND _spi_flags could get the related status.

28.2.6.8 static void SPI_ClearInterrupt (SPI_Type * *base*, uint8_t *mask*) [inline], [static]

Parameters

base	SPI base pointer
mask	Interrupt need to be cleared The parameter could be any combination of the following values: • kSPI_RxFullAndModfInterruptEnable • kSPI_TxEmptyInterruptEnable • kSPI_MatchInterruptEnable • kSPI_RxFifoNearFullInterruptEnable • kSPI_TxFifoNearEmptyInterruptEnable

28.2.6.9 void SPI_EnableInterrupts (SPI_Type * *base*, uint32_t *mask*)

Parameters

base	SPI base pointer
mask	SPI interrupt source. The parameter can be any combination of the following values: • kSPI_RxFullAndModfInterruptEnable • kSPI_TxEmptyInterruptEnable • kSPI_MatchInterruptEnable • kSPI_RxFifoNearFullInterruptEnable • kSPI_TxFifoNearEmptyInterruptEnable

28.2.6.10 void SPI_DisableInterrupts (SPI_Type * *base,* uint32_t *mask*)

Parameters

base	SPI base pointer
mask	SPI interrupt source. The parameter can be any combination of the following values: • kSPI_RxFullAndModfInterruptEnable • kSPI_TxEmptyInterruptEnable • kSPI_MatchInterruptEnable • kSPI_RxFifoNearFullInterruptEnable • kSPI_TxFifoNearEmptyInterruptEnable

28.2.6.11 static void SPI_EnableDMA (SPI_Type * *base,* uint8_t *mask,* bool *enable*) [inline], [static]

Parameters

base	SPI base pointer
mask	SPI DMA source.
enable	True means enable DMA, false means disable DMA

28.2.6.12 static uint32_t SPI_GetDataRegisterAddress (SPI_Type * *base*) [inline], [static]

This API is used to provide a transfer address for the SPI DMA transfer configuration.

Parameters

base	SPI base pointer

Returns

data register address

28.2.6.13 uint32_t SPI_GetInstance (SPI_Type ∗ *base*)

Parameters

base	SPI base address

28.2.6.14 static void SPI_SetPinMode (SPI_Type ∗ *base*, spi_pin_mode_t *pinMode*) [inline], [static]

Parameters

base	SPI base pointer
pinMode	pin mode for transfer AND _spi_pin_mode could get the related configuration.

28.2.6.15 void SPI_MasterSetBaudRate (SPI_Type ∗ *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

This is only used in master.

Parameters

base	SPI base pointer
baudRate_Bps	baud rate needed in Hz.
srcClock_Hz	SPI source clock frequency in Hz.

28.2.6.16 static void SPI_SetMatchData (SPI_Type ∗ *base*, uint32_t *matchData*) [inline], [static]

The match data is a hardware comparison value. When the value received in the SPI receive data buffer equals the hardware comparison value, the SPI Match Flag in the S register (S[SPMF]) sets. This can also generate an interrupt if the enable bit sets.

Parameters

base	SPI base pointer
matchData	Match data.

28.2.6.17 void SPI_EnableFIFO (SPI_Type * *base*, bool *enable*)

Parameters

base	SPI base pointer
enable	True means enable FIFO, false means disable FIFO.

28.2.6.18 status_t SPI_WriteBlocking (SPI_Type * *base*, uint8_t * *buffer*, size_t *size*)

Note

This function blocks via polling until all bytes have been sent.

Parameters

base	SPI base pointer
buffer	The data bytes to send
size	The number of data bytes to send

Returns

kStatus_SPI_Timeout The transfer timed out and was aborted.

28.2.6.19 void SPI_WriteData (SPI_Type * *base*, uint16_t *data*)

Parameters

base	SPI base pointer
data	needs to be write.

28.2.6.20 uint16_t SPI_ReadData (SPI_Type * *base*)

Parameters

base	SPI base pointer

Returns

Data in the register.

28.2.6.21 void SPI_SetDummyData (SPI_Type * *base*, uint8_t *dummyData*)

Parameters

base	SPI peripheral address.
dummyData	Data to be transferred when tx buffer is NULL.

28.2.6.22 void SPI_MasterTransferCreateHandle (SPI_Type * *base*, spi_master_handle_t * *handle*, spi_master_callback_t *callback*, void * *userData*)

This function initializes the SPI master handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	Callback function.
userData	User data.

28.2.6.23 status_t SPI_MasterTransferBlocking (SPI_Type * *base*, spi_transfer_t * *xfer*)

Parameters

base	SPI base pointer
xfer	pointer to spi_xfer_config_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.

28.2.6.24 status_t SPI_MasterTransferNonBlocking (SPI_Type * *base*, spi_master_handle_t * *handle*, spi_transfer_t * *xfer*)

Note

The API immediately returns after transfer initialization is finished. Call SPI_GetStatusIRQ() to get the transfer status.
If SPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

Parameters

base	SPI peripheral base address.
handle	pointer to spi_master_handle_t structure which stores the transfer state
xfer	pointer to spi_xfer_config_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

28.2.6.25 status_t SPI_MasterTransferGetCount (SPI_Type * *base*, spi_master_handle_t * *handle*, size_t * *count*)

Parameters

base	SPI peripheral base address.
handle	Pointer to SPI transfer handle, this should be a static variable.
count	Transferred bytes of SPI master.

Return values

kStatus_SPI_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

28.2.6.26 void SPI_MasterTransferAbort (SPI_Type * *base*, spi_master_handle_t * *handle*)

Parameters

base	SPI peripheral base address.
handle	Pointer to SPI transfer handle, this should be a static variable.

28.2.6.27 void SPI_MasterTransferHandleIRQ (SPI_Type * *base*, spi_master_handle_t * *handle*)

Parameters

base	SPI peripheral base address.
handle	pointer to spi_master_handle_t structure which stores the transfer state.

28.2.6.28 void SPI_SlaveTransferCreateHandle (SPI_Type * *base*, spi_slave_handle_t * *handle*, spi_slave_callback_t *callback*, void * *userData*)

This function initializes the SPI slave handle which can be used for other SPI slave transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	Callback function.

userData	User data.

28.2.6.29 status_t SPI_SlaveTransferNonBlocking (SPI_Type * *base*, spi_slave_handle_t * *handle*, spi_transfer_t * *xfer*)

Note

The API returns immediately after the transfer initialization is finished. Call SPI_GetStatusIRQ() to get the transfer status.
If SPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

Parameters

base	SPI peripheral base address.
handle	pointer to spi_slave_handle_t structure which stores the transfer state
xfer	pointer to spi_xfer_config_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

28.2.6.30 static status_t SPI_SlaveTransferGetCount (SPI_Type * *base*, spi_slave_handle_t * *handle*, size_t * *count*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	Pointer to SPI transfer handle, this should be a static variable.
count	Transferred bytes of SPI slave.

Return values

kStatus_SPI_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

28.2.6.31 static void SPI_SlaveTransferAbort (SPI_Type * *base*, spi_slave_handle_t * *handle*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	Pointer to SPI transfer handle, this should be a static variable.

28.2.6.32 void SPI_SlaveTransferHandleIRQ (SPI_Type * *base*, spi_slave_handle_t * *handle*)

Parameters

base	SPI peripheral base address.
handle	pointer to spi_slave_handle_t structure which stores the transfer state

28.2.7 Variable Documentation

28.2.7.1 volatile uint8_t g_spiDummyData[ ]

28.3 SPI DMA Driver

28.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

Data Structures

- struct spi_dma_handle_t
  SPI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ spi_dma_callback_t)(SPI_Type ∗base, spi_dma_handle_t ∗handle, status_t status, void ∗userData)
  SPI DMA callback called at the end of transfer.

Driver version

- #define FSL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  SPI DMA driver version.

DMA Transactional

- void SPI_MasterTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI master DMA handle.
- status_t SPI_MasterTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.
- void SPI_MasterTransferAbortDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle)
  Abort a SPI transfer using DMA.
- status_t SPI_MasterTransferGetCountDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, size_t ∗count)
  Get the transferred bytes for SPI slave DMA.
- static void SPI_SlaveTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI slave DMA handle.
- static status_t SPI_SlaveTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.
- static void SPI_SlaveTransferAbortDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle)
  Abort a SPI transfer using DMA.
- static status_t SPI_SlaveTransferGetCountDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, size_t ∗count)

Get the transferred bytes for SPI slave DMA.

28.3.2 Data Structure Documentation

28.3.2.1 struct _spi_dma_handle

Data Fields

- bool txInProgress
  Send transfer finished.
- bool rxInProgress
  Receive transfer finished.
- dma_handle_t ∗ txHandle
  DMA handler for SPI send.
- dma_handle_t ∗ rxHandle
  DMA handler for SPI receive.
- uint8_t bytesPerFrame
  Bytes in a frame for SPI transfer.
- spi_dma_callback_t callback
  Callback for SPI DMA transfer.
- void ∗ userData
  User Data for SPI DMA callback.
- uint32_t state
  Internal state of SPI DMA transfer.
- size_t transferSize
  Bytes need to be transfer.

28.3.3 Macro Definition Documentation

28.3.3.1 #define FSL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

28.3.4 Typedef Documentation

28.3.4.1 typedef void(∗ spi_dma_callback_t)(SPI_Type ∗base, spi_dma_handle_t ∗handle, status_t status, void ∗userData)

28.3.5 Function Documentation

28.3.5.1 void SPI_MasterTransferCreateHandleDMA (SPI_Type ∗ *base,* spi_dma_handle_t ∗ *handle,* spi_dma_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *txHandle,* dma_handle_t ∗ *rxHandle*)

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

28.3.5.2 status_t SPI_MasterTransferDMA (SPI_Type ∗ *base,* spi_dma_handle_t ∗ *handle,* spi_transfer_t ∗ *xfer*)

Note

This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

28.3.5.3 void SPI_MasterTransferAbortDMA (SPI_Type ∗ *base,* spi_dma_handle_t ∗ *handle*)

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

28.3.5.4 status_t SPI_MasterTransferGetCountDMA (SPI_Type ∗ *base*, spi_dma_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
count	Transferred bytes.

Return values

kStatus_SPI_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

28.3.5.5 static void SPI_SlaveTransferCreateHandleDMA (SPI_Type ∗ *base*, spi_dma_handle_t ∗ *handle*, spi_dma_callback_t *callback*, void ∗ *userData*, dma_handle_t ∗ *txHandle*, dma_handle_t ∗ *rxHandle*) [inline], [static]

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

28.3.5.6 static status_t SPI_SlaveTransferDMA (SPI_Type ∗ *base*, spi_dma_handle_t ∗ *handle*, spi_transfer_t ∗ *xfer*) [inline], [static]

Note

This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

28.3.5.7 static void SPI_SlaveTransferAbortDMA (SPI_Type * *base,* spi_dma_handle_t * *handle*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

28.3.5.8 static status_t SPI_SlaveTransferGetCountDMA (SPI_Type * *base,* spi_dma_handle_t * *handle,* size_t * *count*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
count	Transferred bytes.

Return values

kStatus_SPI_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

28.4 SPI FreeRTOS driver

28.4.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

Driver version

- #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  SPI FreeRTOS driver version.

SPI RTOS Operation

- status_t SPI_RTOS_Init (spi_rtos_handle_t *handle, SPI_Type *base, const spi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  Initializes SPI.
- status_t SPI_RTOS_Deinit (spi_rtos_handle_t *handle)
  Deinitializes the SPI.
- status_t SPI_RTOS_Transfer (spi_rtos_handle_t *handle, spi_transfer_t *transfer)
  Performs SPI transfer.

28.4.2 Macro Definition Documentation

28.4.2.1 #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

28.4.3 Function Documentation

28.4.3.1 status_t SPI_RTOS_Init (spi_rtos_handle_t * *handle*, SPI_Type * *base*, const spi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

This function initializes the SPI module and related RTOS context.

Parameters

handle	The RTOS SPI handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the SPI instance to initialize.
masterConfig	Configuration structure to set-up SPI in master mode.

srcClock_Hz	Frequency of input clock of the SPI module.

Returns

status of the operation.

28.4.3.2 status_t SPI_RTOS_Deinit (spi_rtos_handle_t ∗ *handle*)

This function deinitializes the SPI module and related RTOS context.

Parameters

handle	The RTOS SPI handle.

28.4.3.3 status_t SPI_RTOS_Transfer (spi_rtos_handle_t ∗ *handle,* spi_transfer_t ∗ *transfer*)

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

handle	The RTOS SPI handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

28.5 SPI CMSIS driver

This section describes the programming interface of the SPI Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage methord please refer to http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

28.5.1 Function groups

28.5.1.1 SPI CMSIS GetVersion Operation

This function group will return the SPI CMSIS Driver version to user.

28.5.1.2 SPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

28.5.1.3 SPI CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the instance in master mode or slave mode. And this API must be called before you configure an instance or after you Deinit an instance.The right steps to start an instance is that you must initialize the instance which been slected firstly,then you can power on the instance.After these all have been done,you can configure the instance by using control operation.If you want to Uninitialize the instance, you must power off the instance first.

28.5.1.4 SPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

28.5.1.5 SPI CMSIS Status Operation

This function group gets the SPI transfer status.

28.5.1.6 SPI CMSIS Control Operation

This function can configure instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer,set transfer data bits and other control command.

28.5.2 Typical use case

28.5.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*SPI master init*/
Driver_SPI0.Initialize(SPI_MasterSignalEvent_t);
Driver_SPI0.PowerControl(ARM_POWER_FULL);
Driver_SPI0.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
Driver_SPI0.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
Driver_SPI0.PowerControl(ARM_POWER_OFF);

/* Master uninitialize */
Driver_SPI0.Uninitialize();
```

28.5.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*SPI slave init*/
Driver_SPI1.Initialize(SPI_SlaveSignalEvent_t);
Driver_SPI1.PowerControl(ARM_POWER_FULL);
Driver_SPI1.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
Driver_SPI1.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
Driver_SPI1.PowerControl(ARM_POWER_OFF);

/* slave uninitialize */
Driver_SPI1.Uninitialize();
```

Chapter 29
SYSMPU: System Memory Protection Unit

29.1 Overview

The SYSMPU driver provides hardware access control for all memory references generated in the device. Use the SYSMPU driver to program the region descriptors that define memory spaces and their access rights. After initialization, the SYSMPU concurrently monitors the system bus transactions and evaluates their appropriateness.

29.2 Initialization and Deinitialization

To initialize the SYSMPU module, call the SYSMPU_Init() function and provide the user configuration data structure. This function sets the configuration of the SYSMPU module automatically and enables the SYSMPU module.

Note that the configuration start address, end address, the region valid value, and the debugger's access permission for the SYSMPU region 0 cannot be changed.

This is an example code to configure the SYSMPU driver.

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

29.3 Basic Control Operations

SYSMPU can be enabled/disabled for the entire memory protection region by calling the SYSMPU_Enable() function. To save the power for any unused special regions when the entire memory protection region is disabled, call the SYSMPU_RegionEnable().

After SYSMPU initialization, the SYSMPU_SetRegionLowMasterAccessRights() and SYSMPU_SetRegionHighMasterAccessRights() can be used to change the access rights for special master ports and for special region numbers. The SYSMPU_SetRegionConfig can be used to set the whole region with the start/end address with access rights.

The SYSMPU_GetHardwareInfo() API is provided to get the hardware information for the device. The SYSMPU_GetSlavePortErrorStatus() API is provided to get the error status of a special slave port. When an error happens in this port, the SYSMPU_GetDetailErrorAccessInfo() API is provided to get the detailed error information.

Data Structures

- struct sysmpu_hardware_info_t
  SYSMPU hardware basic information. More...
- struct sysmpu_access_err_info_t
  SYSMPU detail error access information. More...
- struct sysmpu_rwxrights_master_access_control_t

SYSMPU read/write/execute rights control for bus master 0 ∼ 3. More...
- struct sysmpu_rwrights_master_access_control_t
  SYSMPU read/write access control for bus master 4 ∼ 7. More...
- struct sysmpu_region_config_t
  SYSMPU region configuration structure. More...
- struct sysmpu_config_t
  The configuration structure for the SYSMPU initialization. More...

Macros

- #define SYSMPU_MASTER_RWATTRIBUTE_START_PORT (4U)
  define the start master port with read and write attributes.
- #define SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n) ((n)∗6U)
  SYSMPU the bit shift for masters with privilege rights: read write and execute.
- #define SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(n) (0x1FUL << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))
  SYSMPU masters with read, write and execute rights bit mask.
- #define SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH 5U
  SYSMPU masters with read, write and execute rights bit width.
- #define SYSMPU_REGION_RWXRIGHTS_MASTER(n, x) (((uint32_t)(((uint32_t)(x)) << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(n))
  SYSMPU masters with read, write and execute rights priority setting.
- #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n) ((n)∗6U + SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH)
  SYSMPU masters with read, write and execute rights process enable bit shift.
- #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK(n) (0x1UL << SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))
  SYSMPU masters with read, write and execute rights process enable bit mask.
- #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE(n, x)
  SYSMPU masters with read, write and execute rights process enable setting.
- #define SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n) (((n)-SYSMPU_MASTER_RWATTRIBUTE_START_PORT) ∗ 2U + 24U)
  SYSMPU masters with normal read write permission bit shift.
- #define SYSMPU_REGION_RWRIGHTS_MASTER_MASK(n) (0x3UL << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))
  SYSMPU masters with normal read write rights bit mask.
- #define SYSMPU_REGION_RWRIGHTS_MASTER(n, x) (((uint32_t)(((uint32_t)(x)) << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWRIGHTS_MASTER_MASK(n))
  SYSMPU masters with normal read write rights priority setting.

Enumerations

- enum sysmpu_region_total_num_t {
  kSYSMPU_8Regions = 0x0U,
  kSYSMPU_12Regions = 0x1U,
  kSYSMPU_16Regions = 0x2U }
  Describes the number of SYSMPU regions.

- enum sysmpu_slave_t {
kSYSMPU_Slave0 = 0U,
kSYSMPU_Slave1 = 1U,
kSYSMPU_Slave2 = 2U,
kSYSMPU_Slave3 = 3U,
kSYSMPU_Slave4 = 4U }
SYSMPU slave port number.
- enum sysmpu_err_access_control_t {
kSYSMPU_NoRegionHit = 0U,
kSYSMPU_NoneOverlappRegion = 1U,
kSYSMPU_OverlappRegion = 2U }
SYSMPU error access control detail.
- enum sysmpu_err_access_type_t {
kSYSMPU_ErrTypeRead = 0U,
kSYSMPU_ErrTypeWrite = 1U }
SYSMPU error access type.
- enum sysmpu_err_attributes_t {
kSYSMPU_InstructionAccessInUserMode = 0U,
kSYSMPU_DataAccessInUserMode = 1U,
kSYSMPU_InstructionAccessInSupervisorMode = 2U,
kSYSMPU_DataAccessInSupervisorMode = 3U }
SYSMPU access error attributes.
- enum sysmpu_supervisor_access_rights_t {
kSYSMPU_SupervisorReadWriteExecute = 0U,
kSYSMPU_SupervisorReadExecute = 1U,
kSYSMPU_SupervisorReadWrite = 2U,
kSYSMPU_SupervisorEqualToUsermode = 3U }
SYSMPU access rights in supervisor mode for bus master 0 ∼ 3.
- enum sysmpu_user_access_rights_t {
kSYSMPU_UserNoAccessRights = 0U,
kSYSMPU_UserExecute = 1U,
kSYSMPU_UserWrite = 2U,
kSYSMPU_UserWriteExecute = 3U,
kSYSMPU_UserRead = 4U,
kSYSMPU_UserReadExecute = 5U,
kSYSMPU_UserReadWrite = 6U,
kSYSMPU_UserReadWriteExecute = 7U }
SYSMPU access rights in user mode for bus master 0 ∼ 3.

Driver version

- #define FSL_SYSMPU_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))
SYSMPU driver version 2.2.3.

Initialization and deinitialization

- void SYSMPU_Init (SYSMPU_Type ∗base, const sysmpu_config_t ∗config)

Initializes the SYSMPU with the user configuration structure.
- void SYSMPU_Deinit (SYSMPU_Type ∗base)
  Deinitializes the SYSMPU regions.

Basic Control Operations

- static void SYSMPU_Enable (SYSMPU_Type ∗base, bool enable)
  Enables/disables the SYSMPU globally.
- static void SYSMPU_RegionEnable (SYSMPU_Type ∗base, uint32_t number, bool enable)
  Enables/disables the SYSMPU for a special region.
- void SYSMPU_GetHardwareInfo (SYSMPU_Type ∗base, sysmpu_hardware_info_t ∗hardwareInform)
  Gets the SYSMPU basic hardware information.
- void SYSMPU_SetRegionConfig (SYSMPU_Type ∗base, const sysmpu_region_config_t ∗regionConfig)
  Sets the SYSMPU region.
- void SYSMPU_SetRegionAddr (SYSMPU_Type ∗base, uint32_t regionNum, uint32_t startAddr, uint32_t endAddr)
  Sets the region start and end address.
- void SYSMPU_SetRegionRwxMasterAccessRights (SYSMPU_Type ∗base, uint32_t regionNum, uint32_t masterNum, const sysmpu_rwxrights_master_access_control_t ∗accessRights)
  Sets the SYSMPU region access rights for masters with read, write, and execute rights.
- bool SYSMPU_GetSlavePortErrorStatus (SYSMPU_Type ∗base, sysmpu_slave_t slaveNum)
  Gets the numbers of slave ports where errors occur.
- void SYSMPU_GetDetailErrorAccessInfo (SYSMPU_Type ∗base, sysmpu_slave_t slaveNum, sysmpu_access_err_info_t ∗errInform)
  Gets the SYSMPU detailed error access information.

29.4 Data Structure Documentation

29.4.1 struct sysmpu_hardware_info_t

Data Fields

- uint8_t hardwareRevisionLevel
  Specifies the SYSMPU's hardware and definition reversion level.
- uint8_t slavePortsNumbers
  Specifies the number of slave ports connected to SYSMPU.
- sysmpu_region_total_num_t regionsNumbers
  Indicates the number of region descriptors implemented.

Field Documentation

(1) uint8_t sysmpu_hardware_info_t::hardwareRevisionLevel

(2) uint8_t sysmpu_hardware_info_t::slavePortsNumbers

(3) sysmpu_region_total_num_t sysmpu_hardware_info_t::regionsNumbers

29.4.2 struct sysmpu_access_err_info_t

Data Fields

- uint32_t master
  Access error master.
- sysmpu_err_attributes_t attributes
  Access error attributes.
- sysmpu_err_access_type_t accessType
  Access error type.
- sysmpu_err_access_control_t accessControl
  Access error control.
- uint32_t address
  Access error address.
- uint8_t processorIdentification
  Access error processor identification.

Field Documentation

(1) uint32_t sysmpu_access_err_info_t::master

(2) sysmpu_err_attributes_t sysmpu_access_err_info_t::attributes

(3) sysmpu_err_access_type_t sysmpu_access_err_info_t::accessType

(4) sysmpu_err_access_control_t sysmpu_access_err_info_t::accessControl

(5) uint32_t sysmpu_access_err_info_t::address

(6) uint8_t sysmpu_access_err_info_t::processorIdentification

29.4.3 struct sysmpu_rwxrights_master_access_control_t

Data Fields

- sysmpu_supervisor_access_rights_t superAccessRights
  Master access rights in supervisor mode.
- sysmpu_user_access_rights_t userAccessRights
  Master access rights in user mode.
- bool processIdentifierEnable
  Enables or disables process identifier.

Field Documentation

(1) sysmpu_supervisor_access_rights_t sysmpu_rwxrights_master_access_control_t::superAccessRights

(2) sysmpu_user_access_rights_t sysmpu_rwxrights_master_access_control_t::userAccessRights

(3) bool sysmpu_rwxrights_master_access_control_t::processIdentifierEnable

29.4.4 struct sysmpu_rwrights_master_access_control_t

Data Fields

- bool writeEnable
  Enables or disables write permission.
- bool readEnable
  Enables or disables read permission.

Field Documentation

(1) bool sysmpu_rwrights_master_access_control_t::writeEnable

(2) bool sysmpu_rwrights_master_access_control_t::readEnable

29.4.5 struct sysmpu_region_config_t

This structure is used to configure the regionNum region. The accessRights1[0] ∼ accessRights1[3] are used to configure the bus master 0 ∼ 3 with the privilege rights setting. The accessRights2[0] ∼ accessRights2[3] are used to configure the high master 4 ∼ 7 with the normal read write permission. The master port assignment is the chip configuration. Normally, the core is the master 0, debugger is the master 1. Note that the SYSMPU assigns a priority scheme where the debugger is treated as the highest priority master followed by the core and then all the remaining masters. SYSMPU protection does not allow writes from the core to affect the "regionNum 0" start and end address nor the permissions associated with the debugger. It can only write the permission fields associated with the other masters. This protection guarantees that the debugger always has access to the entire address space and those rights can't be changed by the core or any other bus master. Prepare the region configuration when regionNum is 0.

Data Fields

- uint32_t regionNum
  SYSMPU region number, range form 0 ∼ FSL_FEATURE_SYSMPU_DESCRIPTOR_COUNT - 1.
- uint32_t startAddress
  Memory region start address.
- uint32_t endAddress
  Memory region end address.
- sysmpu_rwxrights_master_access_control_t accessRights1 [4]
  Masters with read, write and execute rights setting.
- sysmpu_rwrights_master_access_control_t accessRights2 [4]
  Masters with normal read write rights setting.
- uint8_t processIdentifier

Process identifier used when "processIdentifierEnable" set with true.
- uint8_t processIdMask
  Process identifier mask.

Field Documentation

(1) uint32_t sysmpu_region_config_t::regionNum

(2) uint32_t sysmpu_region_config_t::startAddress

Note: bit0 ∼ bit4 always be marked as 0 by SYSMPU. The actual start address is 0-modulo-32 byte address.

(3) uint32_t sysmpu_region_config_t::endAddress

Note: bit0 ∼ bit4 always be marked as 1 by SYSMPU. The actual end address is 31-modulo-32 byte address.

(4) sysmpu_rwxrights_master_access_control_t sysmpu_region_config_t::accessRights1[4]

(5) sysmpu_rwrights_master_access_control_t sysmpu_region_config_t::accessRights2[4]

(6) uint8_t sysmpu_region_config_t::processIdentifier

(7) uint8_t sysmpu_region_config_t::processIdMask

The setting bit will ignore the same bit in process identifier.

29.4.6 struct sysmpu_config_t

This structure is used when calling the SYSMPU_Init function.

Data Fields

- sysmpu_region_config_t regionConfig
  Region access permission.
- struct _sysmpu_config ∗ next
  Pointer to the next structure.

Field Documentation

(1) sysmpu_region_config_t sysmpu_config_t::regionConfig

(2) struct _sysmpu_config∗ sysmpu_config_t::next

29.5 Macro Definition Documentation

29.5.1 #define FSL_SYSMPU_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))

29.5.2 #define SYSMPU_MASTER_RWATTRIBUTE_START_PORT (4U)

29.5.3 #define SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(*n*) ((n)*6U)

29.5.4 #define SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(*n*) (0x1FUL << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))

29.5.5 #define SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH 5U

29.5.6 #define SYSMPU_REGION_RWXRIGHTS_MASTER(*n,* *x*) (((uint32_t)(((uint32_t)(x)) << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(n))

29.5.7 #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(*n*) ((n)*6U + SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH)

29.5.8 #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK(*n*) (0x1UL << SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))

29.5.9 #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE(*n,* *x*)

Value:

```
(((uint32_t)(((uint32_t)(x)) << SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT
    (n))) & \
    SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK(n))
```

29.5.10 #define SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(*n*) (((n)-SYSMPU_MASTER_RWATTRIBUTE_START_PORT) * 2U + 24U)

29.5.11 #define SYSMPU_REGION_RWRIGHTS_MASTER_MASK(*n*) (0x3UL << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))

29.5.12 #define SYSMPU_REGION_RWRIGHTS_MASTER(*n,* *x*) (((uint32_t)(((uint32_t)(x)) << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWRIGHTS_MASTER_MASK(n))

29.6 Enumeration Type Documentation

29.6.1 enum sysmpu_region_total_num_t

Enumerator

kSYSMPU_8Regions SYSMPU supports 8 regions.
kSYSMPU_12Regions SYSMPU supports 12 regions.
kSYSMPU_16Regions SYSMPU supports 16 regions.

29.6.2 enum sysmpu_slave_t

Enumerator

kSYSMPU_Slave0 SYSMPU slave port 0.
kSYSMPU_Slave1 SYSMPU slave port 1.
kSYSMPU_Slave2 SYSMPU slave port 2.
kSYSMPU_Slave3 SYSMPU slave port 3.
kSYSMPU_Slave4 SYSMPU slave port 4.

29.6.3 enum sysmpu_err_access_control_t

Enumerator

kSYSMPU_NoRegionHit No region hit error.
kSYSMPU_NoneOverlappRegion Access single region error.
kSYSMPU_OverlappRegion Access overlapping region error.

29.6.4 enum sysmpu_err_access_type_t

Enumerator

kSYSMPU_ErrTypeRead SYSMPU error access type — read.
kSYSMPU_ErrTypeWrite SYSMPU error access type — write.

29.6.5 enum sysmpu_err_attributes_t

Enumerator

kSYSMPU_InstructionAccessInUserMode Access instruction error in user mode.
kSYSMPU_DataAccessInUserMode Access data error in user mode.
kSYSMPU_InstructionAccessInSupervisorMode Access instruction error in supervisor mode.
kSYSMPU_DataAccessInSupervisorMode Access data error in supervisor mode.

29.6.6 enum sysmpu_supervisor_access_rights_t

Enumerator

kSYSMPU_SupervisorReadWriteExecute Read write and execute operations are allowed in supervisor mode.
kSYSMPU_SupervisorReadExecute Read and execute operations are allowed in supervisor mode.
kSYSMPU_SupervisorReadWrite Read write operations are allowed in supervisor mode.
kSYSMPU_SupervisorEqualToUsermode Access permission equal to user mode.

29.6.7 enum sysmpu_user_access_rights_t

Enumerator

kSYSMPU_UserNoAccessRights No access allowed in user mode.
kSYSMPU_UserExecute Execute operation is allowed in user mode.
kSYSMPU_UserWrite Write operation is allowed in user mode.
kSYSMPU_UserWriteExecute Write and execute operations are allowed in user mode.
kSYSMPU_UserRead Read is allowed in user mode.
kSYSMPU_UserReadExecute Read and execute operations are allowed in user mode.
kSYSMPU_UserReadWrite Read and write operations are allowed in user mode.
kSYSMPU_UserReadWriteExecute Read write and execute operations are allowed in user mode.

29.7 Function Documentation

29.7.1 void SYSMPU_Init (SYSMPU_Type * *base*, const sysmpu_config_t * *config*)

This function configures the SYSMPU module with the user-defined configuration.

Parameters

base	SYSMPU peripheral base address.
config	The pointer to the configuration structure.

29.7.2 void SYSMPU_Deinit (SYSMPU_Type * *base*)

Parameters

base	SYSMPU peripheral base address.

29.7.3 static void SYSMPU_Enable (SYSMPU_Type * *base,* bool *enable*) [inline], [static]

Call this API to enable or disable the SYSMPU module.

Parameters

base	SYSMPU peripheral base address.
enable	True enable SYSMPU, false disable SYSMPU.

29.7.4 static void SYSMPU_RegionEnable (SYSMPU_Type * *base,* uint32_t *number,* bool *enable*) [inline], [static]

When SYSMPU is enabled, call this API to disable an unused region of an enabled SYSMPU. Call this API to minimize the power dissipation.

Parameters

base	SYSMPU peripheral base address.
number	SYSMPU region number.
enable	True enable the special region SYSMPU, false disable the special region SYSMPU.

29.7.5 void SYSMPU_GetHardwareInfo (SYSMPU_Type * *base,* sysmpu_hardware_info_t * *hardwareInform*)

Parameters

base	SYSMPU peripheral base address.
hardware-Inform	The pointer to the SYSMPU hardware information structure. See "sysmpu_hardware-_info_t".

29.7.6 void SYSMPU_SetRegionConfig (SYSMPU_Type * *base*, const sysmpu_region_config_t * *regionConfig*)

Note: Due to the SYSMPU protection, the region number 0 does not allow writes from core to affect the start and end address nor the permissions associated with the debugger. It can only write the permission fields associated with the other masters.

Parameters

base	SYSMPU peripheral base address.
regionConfig	The pointer to the SYSMPU user configuration structure. See "sysmpu_region_-config_t".

29.7.7 void SYSMPU_SetRegionAddr (SYSMPU_Type * *base*, uint32_t *regionNum*, uint32_t *startAddr*, uint32_t *endAddr*)

Memory region start address. Note: bit0 ~ bit4 is always marked as 0 by SYSMPU. The actual start address by SYSMPU is 0-modulo-32 byte address. Memory region end address. Note: bit0 ~ bit4 always be marked as 1 by SYSMPU. The end address used by the SYSMPU is 31-modulo-32 byte address. Note: Due to the SYSMPU protection, the startAddr and endAddr can't be changed by the core when regionNum is 0.

Parameters

base	SYSMPU peripheral base address.
regionNum	SYSMPU region number. The range is from 0 to FSL_FEATURE_SYSMPU_DES-CRIPTOR_COUNT - 1.
startAddr	Region start address.
endAddr	Region end address.

29.7.8 void SYSMPU_SetRegionRwxMasterAccessRights (SYSMPU_Type * *base*, uint32_t *regionNum*, uint32_t *masterNum*, const sysmpu_rwxrights_master_access_control_t * *accessRights*)

The SYSMPU access rights depend on two board classifications of bus masters. The privilege rights masters and the normal rights masters. The privilege rights masters have the read, write, and execute access rights. Except the normal read and write rights, the execute rights are also allowed for these masters. The privilege rights masters normally range from bus masters 0 - 3. However, the maximum master number is device-specific. See the "SYSMPU_PRIVILEGED_RIGHTS_MASTER_MAX_INDEX". The normal rights masters access rights control see "SYSMPU_SetRegionRwMasterAccessRights()".

Parameters

base	SYSMPU peripheral base address.
regionNum	SYSMPU region number. Should range from 0 to FSL_FEATURE_SYSMPU_DESCRIPTOR_COUNT - 1.
masterNum	SYSMPU bus master number. Should range from 0 to SYSMPU_PRIVILEGED_RIGHTS_MASTER_MAX_INDEX.
accessRights	The pointer to the SYSMPU access rights configuration. See "sysmpu_rwxrights_master_access_control_t".

29.7.9 bool SYSMPU_GetSlavePortErrorStatus (SYSMPU_Type ∗ *base*, sysmpu_slave_t *slaveNum*)

Parameters

base	SYSMPU peripheral base address.
slaveNum	SYSMPU slave port number.

Returns

The slave ports error status. true - error happens in this slave port. false - error didn't happen in this slave port.

29.7.10 void SYSMPU_GetDetailErrorAccessInfo (SYSMPU_Type ∗ *base*, sysmpu_slave_t *slaveNum*, sysmpu_access_err_info_t ∗ *errInform*)

Parameters

base	SYSMPU peripheral base address.
slaveNum	SYSMPU slave port number.
errInform	The pointer to the SYSMPU access error information. See "sysmpu_access_err_info_t".

Chapter 30
UART: Universal Asynchronous Receiver/Transmitter Driver

30.1 Overview

Modules

- UART CMSIS Driver
- UART DMA Driver
- UART Driver
- UART FreeRTOS Driver

30.2 UART Driver

30.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) module of MCUXpresso SDK devices.

The UART driver includes functional APIs and transactional APIs.

Functional APIs are used for UART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the UART peripheral and how to organize functional APIs to meet the application requirements. All functional APIs use the peripheral base address as the first parameter. UART functional operation groups provide the functional API set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the uart_handle_t as the second parameter. Initialize the handle by calling the UART_TransferCreateHandle() API.

Transactional APIs support asynchronous transfer, which means that the functions UART_TransferSendNonBlocking() and UART_TransferReceiveNonBlocking() set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_UART_TxIdle and kStatus_UART_RxIdle.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the UART_TransferCreateHandle(). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The UART_TransferReceiveNonBlocking() function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the kStatus_UART_RxIdle.

If the receive ring buffer is full, the upper layer is informed through a callback with the kStatus_UART_RxRingBufferOverrun. In the callback function, the upper layer reads data out from the ring buffer. If not, existing data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/uart In this example, the buffer size is 32, but only 31 bytes are used for saving data.

30.2.2 Typical use case

30.2.2.1 UART Send/receive using a polling method

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

30.2.2.2 UART Send/receive using an interrupt method

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

30.2.2.3 UART Receive using the ringbuffer feature

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

30.2.2.4 UART Send/Receive using the DMA method

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

Data Structures

- struct uart_config_t
  UART configuration structure. More...
- struct uart_transfer_t
  UART transfer structure. More...
- struct uart_handle_t
  UART 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(∗ uart_transfer_callback_t)(UART_Type ∗base, uart_handle_t ∗handle, status_t status, void ∗userData)
  UART transfer callback function.

Enumerations

- enum {
kStatus_UART_TxBusy = MAKE_STATUS(kStatusGroup_UART, 0),
kStatus_UART_RxBusy = MAKE_STATUS(kStatusGroup_UART, 1),
kStatus_UART_TxIdle = MAKE_STATUS(kStatusGroup_UART, 2),
kStatus_UART_RxIdle = MAKE_STATUS(kStatusGroup_UART, 3),
kStatus_UART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_UART, 4),
kStatus_UART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_UART, 5),
kStatus_UART_FlagCannotClearManually,
kStatus_UART_Error = MAKE_STATUS(kStatusGroup_UART, 7),
kStatus_UART_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_UART, 8),
kStatus_UART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_UART, 9),
kStatus_UART_NoiseError = MAKE_STATUS(kStatusGroup_UART, 10),
kStatus_UART_FramingError = MAKE_STATUS(kStatusGroup_UART, 11),
kStatus_UART_ParityError = MAKE_STATUS(kStatusGroup_UART, 12),
kStatus_UART_BaudrateNotSupport,
kStatus_UART_IdleLineDetected = MAKE_STATUS(kStatusGroup_UART, 14),
kStatus_UART_Timeout = MAKE_STATUS(kStatusGroup_UART, 15) }
  Error codes for the UART driver.
- enum uart_parity_mode_t {
kUART_ParityDisabled = 0x0U,
kUART_ParityEven = 0x2U,
kUART_ParityOdd = 0x3U }
  UART parity mode.
- enum uart_stop_bit_count_t {
kUART_OneStopBit = 0U,
kUART_TwoStopBit = 1U }
  UART stop bit count.
- enum uart_idle_type_select_t {
kUART_IdleTypeStartBit = 0U,
kUART_IdleTypeStopBit = 1U }
  UART idle type select.
- enum _uart_interrupt_enable {
kUART_RxActiveEdgeInterruptEnable = (UART_BDH_RXEDGIE_MASK),
kUART_TxDataRegEmptyInterruptEnable = (UART_C2_TIE_MASK << 8),
kUART_TransmissionCompleteInterruptEnable = (UART_C2_TCIE_MASK << 8),
kUART_RxDataRegFullInterruptEnable = (UART_C2_RIE_MASK << 8),
kUART_IdleLineInterruptEnable = (UART_C2_ILIE_MASK << 8),
kUART_RxOverrunInterruptEnable = (UART_C3_ORIE_MASK << 16),
kUART_NoiseErrorInterruptEnable = (UART_C3_NEIE_MASK << 16),
kUART_FramingErrorInterruptEnable = (UART_C3_FEIE_MASK << 16),
kUART_ParityErrorInterruptEnable = (UART_C3_PEIE_MASK << 16),
kUART_RxFifoOverflowInterruptEnable = (UART_CFIFO_RXOFE_MASK << 24),
kUART_TxFifoOverflowInterruptEnable = (UART_CFIFO_TXOFE_MASK << 24),
kUART_RxFifoUnderflowInterruptEnable = (UART_CFIFO_RXUFE_MASK << 24) }

UART interrupt configuration structure, default settings all disabled.
- enum {
kUART_TxDataRegEmptyFlag = (UART_S1_TDRE_MASK),
kUART_TransmissionCompleteFlag = (UART_S1_TC_MASK),
kUART_RxDataRegFullFlag = (UART_S1_RDRF_MASK),
kUART_IdleLineFlag = (UART_S1_IDLE_MASK),
kUART_RxOverrunFlag = (UART_S1_OR_MASK),
kUART_NoiseErrorFlag = (UART_S1_NF_MASK),
kUART_FramingErrorFlag = (UART_S1_FE_MASK),
kUART_ParityErrorFlag = (UART_S1_PF_MASK),
kUART_RxActiveEdgeFlag,
kUART_RxActiveFlag,
kUART_NoiseErrorInRxDataRegFlag = (UART_ED_NOISY_MASK << 16),
kUART_ParityErrorInRxDataRegFlag = (UART_ED_PARITYE_MASK << 16),
kUART_TxFifoEmptyFlag = (int)(UART_SFIFO_TXEMPT_MASK << 24),
kUART_RxFifoEmptyFlag = (UART_SFIFO_RXEMPT_MASK << 24),
kUART_TxFifoOverflowFlag = (UART_SFIFO_TXOF_MASK << 24),
kUART_RxFifoOverflowFlag = (UART_SFIFO_RXOF_MASK << 24),
kUART_RxFifoUnderflowFlag = (UART_SFIFO_RXUF_MASK << 24) }
UART status flags.

Functions

- uint32_t UART_GetInstance (UART_Type ∗base)
Get the UART instance from peripheral base address.

Variables

- void ∗ s_uartHandle [ ]
Pointers to uart handles for each instance.
- uart_isr_t s_uartIsr
Pointer to uart IRQ handler for each instance.

Driver version

- #define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))
UART driver version.

Initialization and deinitialization

- status_t UART_Init (UART_Type ∗base, const uart_config_t ∗config, uint32_t srcClock_Hz)
Initializes a UART instance with a user configuration structure and peripheral clock.
- void UART_Deinit (UART_Type ∗base)

Deinitializes a UART instance.
- void UART_GetDefaultConfig (uart_config_t ∗config)
  Gets the default configuration structure.
- status_t UART_SetBaudRate (UART_Type ∗base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the UART instance baud rate.
- void UART_Enable9bitMode (UART_Type ∗base, bool enable)
  Enable 9-bit data mode for UART.
- static void UART_SetMatchAddress (UART_Type ∗base, uint8_t address1, uint8_t address2)
  Set the UART slave address.
- static void UART_EnableMatchAddress (UART_Type ∗base, bool match1, bool match2)
  Enable the UART match address feature.
- static void UART_Set9thTransmitBit (UART_Type ∗base)
  Set UART 9th transmit bit.
- static void UART_Clear9thTransmitBit (UART_Type ∗base)
  Clear UART 9th transmit bit.

Status

- uint32_t UART_GetStatusFlags (UART_Type ∗base)
  Gets UART status flags.
- status_t UART_ClearStatusFlags (UART_Type ∗base, uint32_t mask)
  Clears status flags with the provided mask.

Interrupts

- void UART_EnableInterrupts (UART_Type ∗base, uint32_t mask)
  Enables UART interrupts according to the provided mask.
- void UART_DisableInterrupts (UART_Type ∗base, uint32_t mask)
  Disables the UART interrupts according to the provided mask.
- uint32_t UART_GetEnabledInterrupts (UART_Type ∗base)
  Gets the enabled UART interrupts.

DMA Control

- static uint32_t UART_GetDataRegisterAddress (UART_Type ∗base)
  Gets the UART data register address.
- static void UART_EnableTxDMA (UART_Type ∗base, bool enable)
  Enables or disables the UART transmitter DMA request.
- static void UART_EnableRxDMA (UART_Type ∗base, bool enable)
  Enables or disables the UART receiver DMA.

Bus Operations

- static void UART_EnableTx (UART_Type ∗base, bool enable)
  Enables or disables the UART transmitter.
- static void UART_EnableRx (UART_Type ∗base, bool enable)

Enables or disables the UART receiver.
- static void UART_WriteByte (UART_Type *base, uint8_t data)
  Writes to the TX register.
- static uint8_t UART_ReadByte (UART_Type *base)
  Reads the RX register directly.
- static uint8_t UART_GetRxFifoCount (UART_Type *base)
  Gets the rx FIFO data count.
- static uint8_t UART_GetTxFifoCount (UART_Type *base)
  Gets the tx FIFO data count.
- void UART_SendAddress (UART_Type *base, uint8_t address)
  Transmit an address frame in 9-bit data mode.
- status_t UART_WriteBlocking (UART_Type *base, const uint8_t *data, size_t length)
  Writes to the TX register using a blocking method.
- status_t UART_ReadBlocking (UART_Type *base, uint8_t *data, size_t length)
  Read RX data register using a blocking method.

Transactional

- void UART_TransferCreateHandle (UART_Type *base, uart_handle_t *handle, uart_transfer_callback_t callback, void *userData)
  Initializes the UART handle.
- void UART_TransferStartRingBuffer (UART_Type *base, uart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void UART_TransferStopRingBuffer (UART_Type *base, uart_handle_t *handle)
  Aborts the background transfer and uninstalls the ring buffer.
- size_t UART_TransferGetRxRingBufferLength (uart_handle_t *handle)
  Get the length of received data in RX ring buffer.
- status_t UART_TransferSendNonBlocking (UART_Type *base, uart_handle_t *handle, uart_transfer_t *xfer)
  Transmits a buffer of data using the interrupt method.
- void UART_TransferAbortSend (UART_Type *base, uart_handle_t *handle)
  Aborts the interrupt-driven data transmit.
- status_t UART_TransferGetSendCount (UART_Type *base, uart_handle_t *handle, uint32_t *count)
  Gets the number of bytes sent out to bus.
- status_t UART_TransferReceiveNonBlocking (UART_Type *base, uart_handle_t *handle, uart_transfer_t *xfer, size_t *receivedBytes)
  Receives a buffer of data using an interrupt method.
- void UART_TransferAbortReceive (UART_Type *base, uart_handle_t *handle)
  Aborts the interrupt-driven data receiving.
- status_t UART_TransferGetReceiveCount (UART_Type *base, uart_handle_t *handle, uint32_t *count)
  Gets the number of bytes that have been received.
- status_t UART_EnableTxFIFO (UART_Type *base, bool enable)
  Enables or disables the UART Tx FIFO.
- status_t UART_EnableRxFIFO (UART_Type *base, bool enable)
  Enables or disables the UART Rx FIFO.
- static void UART_SetRxFifoWatermark (UART_Type *base, uint8_t water)

Sets the rx FIFO watermark.
- static void UART_SetTxFifoWatermark (UART_Type ∗base, uint8_t water)
  Sets the tx FIFO watermark.
- void UART_TransferHandleIRQ (UART_Type ∗base, void ∗irqHandle)
  UART IRQ handle function.
- void UART_TransferHandleErrorIRQ (UART_Type ∗base, void ∗irqHandle)
  UART Error IRQ handle function.

30.2.3 Data Structure Documentation

30.2.3.1 struct uart_config_t

Data Fields

- uint32_t baudRate_Bps
  UART baud rate.
- uart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- uint8_t txFifoWatermark
  TX FIFO watermark.
- uint8_t rxFifoWatermark
  RX FIFO watermark.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- uart_idle_type_select_t idleType
  IDLE type select.
- bool enableTx
  Enable TX.
- bool enableRx
  Enable RX.

Field Documentation

(1) uart_idle_type_select_t uart_config_t::idleType

30.2.3.2 struct uart_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* uart_transfer_t::data

(2) uint8_t* uart_transfer_t::rxData

(3) const uint8_t* uart_transfer_t::txData

(4) size_t uart_transfer_t::dataSize

30.2.3.3 struct _uart_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.
- uart_transfer_callback_t callback
  Callback function.
- void * userData
  UART callback function parameter.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

Field Documentation

(1) const uint8_t* volatile uart_handle_t::txData

(2) volatile size_t uart_handle_t::txDataSize

(3) size_t uart_handle_t::txDataSizeAll

(4) uint8_t∗ volatile uart_handle_t::rxData

(5) volatile size_t uart_handle_t::rxDataSize

(6) size_t uart_handle_t::rxDataSizeAll

(7) uint8_t∗ uart_handle_t::rxRingBuffer

(8) size_t uart_handle_t::rxRingBufferSize

(9) volatile uint16_t uart_handle_t::rxRingBufferHead

(10) volatile uint16_t uart_handle_t::rxRingBufferTail

(11) uart_transfer_callback_t uart_handle_t::callback

(12) void∗ uart_handle_t::userData

(13) volatile uint8_t uart_handle_t::txState

30.2.4 Macro Definition Documentation

30.2.4.1 #define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))

30.2.4.2 #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/

30.2.5 Typedef Documentation

30.2.5.1 typedef void(∗ uart_transfer_callback_t)(UART_Type ∗base, uart_handle_t ∗handle, status_t status, void ∗userData)

30.2.6 Enumeration Type Documentation

30.2.6.1 anonymous enum

Enumerator

kStatus_UART_TxBusy Transmitter is busy.
kStatus_UART_RxBusy Receiver is busy.
kStatus_UART_TxIdle UART transmitter is idle.
kStatus_UART_RxIdle UART receiver is idle.
kStatus_UART_TxWatermarkTooLarge TX FIFO watermark too large.
kStatus_UART_RxWatermarkTooLarge RX FIFO watermark too large.
kStatus_UART_FlagCannotClearManually UART flag can't be manually cleared.
kStatus_UART_Error Error happens on UART.
kStatus_UART_RxRingBufferOverrun UART RX software ring buffer overrun.

kStatus_UART_RxHardwareOverrun UART RX receiver overrun.
kStatus_UART_NoiseError UART noise error.
kStatus_UART_FramingError UART framing error.
kStatus_UART_ParityError UART parity error.
kStatus_UART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_UART_IdleLineDetected UART IDLE line detected.
kStatus_UART_Timeout UART times out.

30.2.6.2 enum uart_parity_mode_t

Enumerator

kUART_ParityDisabled Parity disabled.
kUART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kUART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

30.2.6.3 enum uart_stop_bit_count_t

Enumerator

kUART_OneStopBit One stop bit.
kUART_TwoStopBit Two stop bits.

30.2.6.4 enum uart_idle_type_select_t

Enumerator

kUART_IdleTypeStartBit Start counting after a valid start bit.
kUART_IdleTypeStopBit Start counting after a stop bit.

30.2.6.5 enum _uart_interrupt_enable

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

Enumerator

kUART_RxActiveEdgeInterruptEnable RX active edge interrupt.
kUART_TxDataRegEmptyInterruptEnable Transmit data register empty interrupt.
kUART_TransmissionCompleteInterruptEnable Transmission complete interrupt.
kUART_RxDataRegFullInterruptEnable Receiver data register full interrupt.
kUART_IdleLineInterruptEnable Idle line interrupt.
kUART_RxOverrunInterruptEnable Receiver overrun interrupt.

kUART_NoiseErrorInterruptEnable Noise error flag interrupt.
kUART_FramingErrorInterruptEnable Framing error flag interrupt.
kUART_ParityErrorInterruptEnable Parity error flag interrupt.
kUART_RxFifoOverflowInterruptEnable RX FIFO overflow interrupt.
kUART_TxFifoOverflowInterruptEnable TX FIFO overflow interrupt.
kUART_RxFifoUnderflowInterruptEnable RX FIFO underflow interrupt.

30.2.6.6 anonymous enum

This provides constants for the UART status flags for use in the UART functions.

Enumerator

kUART_TxDataRegEmptyFlag TX data register empty flag.
kUART_TransmissionCompleteFlag Transmission complete flag.
kUART_RxDataRegFullFlag RX data register full flag.
kUART_IdleLineFlag Idle line detect flag.
kUART_RxOverrunFlag RX overrun flag.
kUART_NoiseErrorFlag RX takes 3 samples of each received bit. If any of these samples differ, noise flag sets
kUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected.
kUART_ParityErrorFlag If parity enabled, sets upon parity error detection.
kUART_RxActiveEdgeFlag RX pin active edge interrupt flag,sets when active edge detected.
kUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start bit.
kUART_NoiseErrorInRxDataRegFlag Noisy bit, sets if noise detected.
kUART_ParityErrorInRxDataRegFlag Parity bit, sets if parity error detected.
kUART_TxFifoEmptyFlag TXEMPT bit, sets if TX buffer is empty.
kUART_RxFifoEmptyFlag RXEMPT bit, sets if RX buffer is empty.
kUART_TxFifoOverflowFlag TXOF bit, sets if TX buffer overflow occurred.
kUART_RxFifoOverflowFlag RXOF bit, sets if receive buffer overflow.
kUART_RxFifoUnderflowFlag RXUF bit, sets if receive buffer underflow.

30.2.7 Function Documentation

30.2.7.1 uint32_t UART_GetInstance (UART_Type ∗ *base*)

Parameters

base	UART peripheral base address.

Returns

UART instance.

30.2.7.2 status_t UART_Init (UART_Type ∗ *base*, const uart_config_t ∗ *config*, uint32_t *srcClock_Hz*)

This function configures the UART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the UART_GetDefaultConfig() function. The example below shows how to use this API to configure UART.

```
*  uart_config_t uartConfig;
*  uartConfig.baudRate_Bps = 115200U;
*  uartConfig.parityMode = kUART_ParityDisabled;
*  uartConfig.stopBitCount = kUART_OneStopBit;
*  uartConfig.txFifoWatermark = 0;
*  uartConfig.rxFifoWatermark = 1;
*  UART_Init(UART1, &uartConfig, 20000000U);
*
```

Parameters

base	UART peripheral base address.
config	Pointer to the user-defined configuration structure.
srcClock_Hz	UART clock source frequency in HZ.

Return values

kStatus_UART_Baudrate-NotSupport	Baudrate is not support in current clock source.
kStatus_Success	Status UART initialize succeed

30.2.7.3 void UART_Deinit (UART_Type ∗ *base*)

This function waits for TX complete, disables TX and RX, and disables the UART clock.

Parameters

base	UART peripheral base address.

30.2.7.4 void UART_GetDefaultConfig (uart_config_t * *config*)

This function initializes the UART configuration structure to a default value. The default values are as follows. uartConfig->baudRate_Bps = 115200U; uartConfig->bitCountPerChar = kUART_8BitsPerChar; uartConfig->parityMode = kUART_ParityDisabled; uartConfig->stopBitCount = kUART_OneStopBit; uartConfig->txFifoWatermark = 0; uartConfig->rxFifoWatermark = 1; uartConfig->idleType = kUART_IdleTypeStartBit; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

config	Pointer to configuration structure.

30.2.7.5 status_t UART_SetBaudRate (UART_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the UART_Init.

```
*  UART_SetBaudRate(UART1, 115200U, 20000000U);
*
```

Parameters

base	UART peripheral base address.
baudRate_Bps	UART baudrate to be set.
srcClock_Hz	UART clock source frequency in Hz.

Return values

kStatus_UART_Baudrate-NotSupport	Baudrate is not support in the current clock source.

kStatus_Success	Set baudrate succeeded.

30.2.7.6 void UART_Enable9bitMode (UART_Type ∗ *base,* bool *enable*)

This function set the 9-bit mode for UART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

base	UART peripheral base address.
enable	true to enable, flase to disable.

30.2.7.7 static void UART_SetMatchAddress (UART_Type ∗ *base,* uint8_t *address1,* uint8_t *address2*) [inline], [static]

This function configures the address for UART 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 UART 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	UART peripheral base address.
address1	UART slave address 1.
address2	UART slave address 2.

30.2.7.8 static void UART_EnableMatchAddress (UART_Type ∗ *base,* bool *match1,* bool *match2*) [inline], [static]

Parameters

base	UART peripheral base address.
match1	true to enable match address1, false to disable.
match2	true to enable match address2, false to disable.

30.2.7.9 static void UART_Set9thTransmitBit (UART_Type ∗ *base*) [inline], [static]

Parameters

base	UART peripheral base address.

30.2.7.10 static void UART_Clear9thTransmitBit (UART_Type ∗ *base*) [inline], [static]

Parameters

base	UART peripheral base address.

30.2.7.11 uint32_t UART_GetStatusFlags (UART_Type ∗ *base*)

This function gets all UART status flags. The flags are returned as the logical OR value of the enumerators _uart_flags. To check a specific status, compare the return value with enumerators in _uart_flags. For example, to check whether the TX is empty, do the following.

```
*     if (kUART_TxDataRegEmptyFlag & UART_GetStatusFlags(UART1))
*     {
*         ...
*     }
*
```

Parameters

base	UART peripheral base address.

Returns

UART status flags which are ORed by the enumerators in the _uart_flags.

30.2.7.12 status_t UART_ClearStatusFlags (UART_Type * *base*, uint32_t *mask*)

This function clears UART status flags with a provided mask. An automatically cleared flag can't be cleared by this function. These flags can only be cleared or set by hardware. kUART_TxDataRegEmptyFlag, kUART_TransmissionCompleteFlag, kUART_RxDataRegFullFlag, kUART_RxActiveFlag, kUART_NoiseErrorInRxDataRegFlag, kUART_ParityErrorInRxDataRegFlag, kUART_TxFifoEmptyFlag,kUART_RxFifoEmptyFlag

Note

that this API should be called when the Tx/Rx is idle. Otherwise it has no effect.

Parameters

base	UART peripheral base address.
mask	The status flags to be cleared; it is logical OR value of _uart_flags.

Return values

kStatus_UART_FlagCannotClearManually	The flag can't be cleared by this function but it is cleared automatically by hardware.
kStatus_Success	Status in the mask is cleared.

30.2.7.13 void UART_EnableInterrupts (UART_Type * *base*, uint32_t *mask*)

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _uart_interrupt_enable. For example, to enable TX empty interrupt and RX full interrupt, do the following.

```
*     UART_EnableInterrupts(UART1,
      kUART_TxDataRegEmptyInterruptEnable |
      kUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	UART peripheral base address.
mask	The interrupts to enable. Logical OR of _uart_interrupt_enable.

30.2.7.14 void UART_DisableInterrupts (UART_Type * *base*, uint32_t *mask*)

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _uart_interrupt_enable. For example, to disable TX empty interrupt and RX full interrupt do the following.

```
*     UART_DisableInterrupts(UART1,
      kUART_TxDataRegEmptyInterruptEnable |
      kUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	UART peripheral base address.
mask	The interrupts to disable. Logical OR of _uart_interrupt_enable.

30.2.7.15 uint32_t UART_GetEnabledInterrupts (UART_Type * *base*)

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _uart_interrupt_enable. To check a specific interrupts enable status, compare the return value with enumerators in _uart_interrupt_enable. For example, to check whether TX empty interrupt is enabled, do the following.

```
*     uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);
*
*     if (kUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
*
```

Parameters

base	UART peripheral base address.

Returns

UART interrupt flags which are logical OR of the enumerators in _uart_interrupt_enable.

30.2.7.16 static uint32_t UART_GetDataRegisterAddress (UART_Type * *base*) [inline], [static]

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

base	UART peripheral base address.

Returns

UART data register addresses which are used both by the transmitter and the receiver.

30.2.7.17 static void UART_EnableTxDMA (UART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the transmit data register empty flag, S1[TDRE], to generate the DMA requests.

Parameters

base	UART peripheral base address.
enable	True to enable, false to disable.

30.2.7.18 static void UART_EnableRxDMA (UART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the receiver data register full flag, S1[RDRF], to generate DMA requests.

Parameters

base	UART peripheral base address.
enable	True to enable, false to disable.

30.2.7.19 static void UART_EnableTx (UART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the UART transmitter.

Parameters

base	UART peripheral base address.
enable	True to enable, false to disable.

30.2.7.20 static void UART_EnableRx (UART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the UART receiver.

Parameters

base	UART peripheral base address.
enable	True to enable, false to disable.

30.2.7.21 static void UART_WriteByte (UART_Type * *base*, uint8_t *data*) [inline], [static]

This function writes data to the TX register directly. The upper layer must ensure that the TX register is empty or TX FIFO has empty room before calling this function.

Parameters

base	UART peripheral base address.
data	The byte to write.

30.2.7.22 static uint8_t UART_ReadByte (UART_Type * *base*) [inline], [static]

This function reads data from the RX register directly. The upper layer must ensure that the RX register is full or that the TX FIFO has data before calling this function.

Parameters

base	UART peripheral base address.

Returns

The byte read from UART data register.

30.2.7.23 static uint8_t UART_GetRxFifoCount (UART_Type * *base*) [inline], [static]

Parameters

base	UART peripheral base address.

Returns

rx FIFO data count.

30.2.7.24 static uint8_t UART_GetTxFifoCount (UART_Type * *base*) [inline], [static]

Parameters

base	UART peripheral base address.

Returns

tx FIFO data count.

30.2.7.25 void UART_SendAddress (UART_Type ∗ *base,* uint8_t *address*)

Parameters

base	UART peripheral base address.
address	UART slave address.

30.2.7.26 status_t UART_WriteBlocking (UART_Type ∗ *base,* const uint8_t ∗ *data,* size_t *length*)

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

base	UART peripheral base address.
data	Start address of the data to write.
length	Size of the data to write.

Return values

kStatus_UART_Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully wrote all data.

30.2.7.27 status_t UART_ReadBlocking (UART_Type ∗ *base,* uint8_t ∗ *data,* size_t *length*)

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the TX register.

Parameters

base	UART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_UART_Rx-HardwareOverrun	Receiver overrun occurred while receiving data.
kStatus_UART_Noise-Error	A noise error occurred while receiving data.
kStatus_UART_Framing-Error	A framing error occurred while receiving data.
kStatus_UART_Parity-Error	A parity error occurred while receiving data.
kStatus_UART_Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully received all data.

30.2.7.28 void UART_TransferCreateHandle (UART_Type * *base*, uart_handle_t * *handle*, uart_transfer_callback_t *callback*, void * *userData*)

This function initializes the UART handle which can be used for other UART transactional APIs. Usually, for a specified UART instance, call this API once to get the initialized handle.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

30.2.7.29 void UART_TransferStartRingBuffer (UART_Type * *base*, uart_handle_t * *handle*, uint8_t * *ringBuffer*, size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If data is already received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
ringBuffer	Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	Size of the ring buffer.

30.2.7.30 void UART_TransferStopRingBuffer (UART_Type * *base*, uart_handle_t * *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.

30.2.7.31 size_t UART_TransferGetRxRingBufferLength (uart_handle_t * *handle*)

Parameters

handle	UART handle pointer.

Returns

Length of received data in RX ring buffer.

30.2.7.32 status_t UART_TransferSendNonBlocking (UART_Type * *base*, uart_handle_t * *handle*, uart_transfer_t * *xfer*)

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the kStatus_UART_TxIdle as status parameter.

Note

The kStatus_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the kUART_-TransmissionCompleteFlag to ensure that the TX is finished.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
xfer	UART transfer structure. See uart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_UART_TxBusy	Previous transmission still not finished; data not all written to TX register yet.
kStatus_InvalidArgument	Invalid argument.

30.2.7.33 void UART_TransferAbortSend (UART_Type ∗ *base*, uart_handle_t ∗ *handle*)

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.

30.2.7.34 status_t UART_TransferGetSendCount (UART_Type ∗ *base*, uart_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes sent out to bus by using the interrupt method.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferIn-Progress	No send in progress.
kStatus_InvalidArgument	The parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

30.2.7.35 status_t UART_TransferReceiveNonBlocking (UART_Type * *base*, uart_handle_t * *handle*, uart_transfer_t * *xfer*, size_t * *receivedBytes*)

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the xfer->data and this function returns with the parameter `receivedBytes` set to 5. For the left 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the xfer->data. When all data is received, the upper layer is notified.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
xfer	UART transfer structure, see uart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into transmit queue.
kStatus_UART_RxBusy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

30.2.7.36 void UART_TransferAbortReceive (UART_Type * *base*, uart_handle_t * *handle*)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.

30.2.7.37 status_t UART_TransferGetReceiveCount (UART_Type * *base*, uart_handle_t * *handle*, uint32_t * *count*)

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

Parameters

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

Return values

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

30.2.7.38 status_t UART_EnableTxFIFO (UART_Type * *base*, bool *enable*)

This function enables or disables the UART Tx FIFO.

param base UART peripheral base address. param enable true to enable, false to disable. retval kStatus_-Success Successfully turn on or turn off Tx FIFO. retval kStatus_Fail Fail to turn on or turn off Tx FIFO.

30.2.7.39 status_t UART_EnableRxFIFO (UART_Type * *base*, bool *enable*)

This function enables or disables the UART Rx FIFO.

param base UART peripheral base address. param enable true to enable, false to disable. retval kStatus_-Success Successfully turn on or turn off Rx FIFO. retval kStatus_Fail Fail to turn on or turn off Rx FIFO.

30.2.7.40 static void UART_SetRxFifoWatermark (UART_Type * *base*, uint8_t *water*) [inline], [static]

Parameters

base	UART peripheral base address.
water	Rx FIFO watermark.

30.2.7.41 static void UART_SetTxFifoWatermark (UART_Type ∗ *base*, uint8_t *water*) [inline], [static]

Parameters

base	UART peripheral base address.
water	Tx FIFO watermark.

30.2.7.42 void UART_TransferHandleIRQ (UART_Type ∗ *base*, void ∗ *irqHandle*)

This function handles the UART transmit and receive IRQ request.

Parameters

base	UART peripheral base address.
irqHandle	UART handle pointer.

30.2.7.43 void UART_TransferHandleErrorIRQ (UART_Type ∗ *base*, void ∗ *irqHandle*)

This function handles the UART error IRQ request.

Parameters

base	UART peripheral base address.
irqHandle	UART handle pointer.

30.2.8 Variable Documentation

30.2.8.1 void∗ s_uartHandle[]

30.2.8.2 uart_isr_t s_uartIsr

30.3 UART DMA Driver

30.3.1 Overview

Data Structures

- struct uart_dma_handle_t
  UART DMA handle. More...

Typedefs

- typedef void(∗ uart_dma_transfer_callback_t)(UART_Type ∗base, uart_dma_handle_t ∗handle, status_t status, void ∗userData)
  UART transfer callback function.

Driver version

- #define FSL_UART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))
  UART DMA driver version.

eDMA transactional

- void UART_TransferCreateHandleDMA (UART_Type ∗base, uart_dma_handle_t ∗handle, uart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the UART handle which is used in transactional functions and sets the callback.
- status_t UART_TransferSendDMA (UART_Type ∗base, uart_dma_handle_t ∗handle, uart_transfer_t ∗xfer)
  Sends data using DMA.
- status_t UART_TransferReceiveDMA (UART_Type ∗base, uart_dma_handle_t ∗handle, uart_transfer_t ∗xfer)
  Receives data using DMA.
- void UART_TransferAbortSendDMA (UART_Type ∗base, uart_dma_handle_t ∗handle)
  Aborts the send data using DMA.
- void UART_TransferAbortReceiveDMA (UART_Type ∗base, uart_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t UART_TransferGetSendCountDMA (UART_Type ∗base, uart_dma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of bytes written to UART TX register.
- status_t UART_TransferGetReceiveCountDMA (UART_Type ∗base, uart_dma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of bytes that have been received.
- void UART_TransferDMAHandleIRQ (UART_Type ∗base, void ∗uartDmaHandle)
  UART DMA IRQ handle function.

30.3.2 Data Structure Documentation

30.3.2.1 struct _uart_dma_handle

Data Fields

- UART_Type * base
  UART peripheral base address.
- uart_dma_transfer_callback_t callback
  Callback function.
- void * userData
  UART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- dma_handle_t * txDmaHandle
  The DMA TX channel used.
- dma_handle_t * rxDmaHandle
  The DMA RX channel used.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

Field Documentation

(1) UART_Type* uart_dma_handle_t::base

(2) uart_dma_transfer_callback_t uart_dma_handle_t::callback

(3) void* uart_dma_handle_t::userData

(4) size_t uart_dma_handle_t::rxDataSizeAll

(5) size_t uart_dma_handle_t::txDataSizeAll

(6) dma_handle_t* uart_dma_handle_t::txDmaHandle

(7) dma_handle_t* uart_dma_handle_t::rxDmaHandle

(8) volatile uint8_t uart_dma_handle_t::txState

30.3.3 Macro Definition Documentation

30.3.3.1 #define FSL_UART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))

30.3.4 Typedef Documentation

30.3.4.1 typedef void(∗ uart_dma_transfer_callback_t)(UART_Type ∗base, uart_dma_handle_t ∗handle, status_t status, void ∗userData)

30.3.5 Function Documentation

30.3.5.1 void UART_TransferCreateHandleDMA (UART_Type ∗ *base,* uart_dma_handle_t ∗ *handle,* uart_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *txDmaHandle,* dma_handle_t ∗ *rxDmaHandle*)

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_dma_handle_t structure.
callback	UART callback, NULL means no callback.
userData	User callback function data.
rxDmaHandle	User requested DMA handle for the RX DMA transfer.
txDmaHandle	User requested DMA handle for the TX DMA transfer.

30.3.5.2 status_t UART_TransferSendDMA (UART_Type ∗ *base,* uart_dma_handle_t ∗ *handle,* uart_transfer_t ∗ *xfer*)

This function sends data using DMA. This is non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
xfer	UART DMA transfer structure. See uart_transfer_t.

Return values

kStatus_Success	if succeeded; otherwise failed.
kStatus_UART_TxBusy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

30.3.5.3 status_t UART_TransferReceiveDMA (UART_Type * *base*, uart_dma_handle_t * *handle*, uart_transfer_t * *xfer*)

This function receives data using DMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_dma_handle_t structure.
xfer	UART DMA transfer structure. See uart_transfer_t.

Return values

kStatus_Success	if succeeded; otherwise failed.
kStatus_UART_RxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

30.3.5.4 void UART_TransferAbortSendDMA (UART_Type ∗ *base*, uart_dma_handle_t ∗ *handle*)

This function aborts the sent data using DMA.

Parameters

base	UART peripheral base address.
handle	Pointer to uart_dma_handle_t structure.

30.3.5.5 void UART_TransferAbortReceiveDMA (UART_Type ∗ *base*, uart_dma_handle_t ∗ *handle*)

This function abort receive data which using DMA.

Parameters

base	UART peripheral base address.
handle	Pointer to uart_dma_handle_t structure.

30.3.5.6 status_t UART_TransferGetSendCountDMA (UART_Type ∗ *base*, uart_dma_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes written to UART TX register by DMA.

Parameters

base	UART peripheral base address.
handle	UART 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`;

30.3.5.7 status_t UART_TransferGetReceiveCountDMA (UART_Type * *base,* uart_dma_handle_t * *handle,* uint32_t * *count*)

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

Parameters

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

Return values

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

30.3.5.8 void UART_TransferDMAHandleIRQ (UART_Type * *base,* void * *uartDmaHandle*)

This function handles the UART transmit complete IRQ request and invoke user callback.

Parameters

base	UART peripheral base address.
uartDma-Handle	UART handle pointer.

30.4 UART FreeRTOS Driver

30.4.1 Overview

Data Structures

- struct uart_rtos_config_t
  UART configuration structure. More...

Driver version

- #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))
  UART FreeRTOS driver version.

UART RTOS Operation

- int UART_RTOS_Init (uart_rtos_handle_t ∗handle, uart_handle_t ∗t_handle, const uart_rtos_config_t ∗cfg)
  Initializes a UART instance for operation in RTOS.
- int UART_RTOS_Deinit (uart_rtos_handle_t ∗handle)
  Deinitializes a UART instance for operation.

UART transactional Operation

- int UART_RTOS_Send (uart_rtos_handle_t ∗handle, uint8_t ∗buffer, uint32_t length)
  Sends data in the background.
- int UART_RTOS_Receive (uart_rtos_handle_t ∗handle, uint8_t ∗buffer, uint32_t length, size_t ∗received)
  Receives data.

30.4.2 Data Structure Documentation

30.4.2.1 struct uart_rtos_config_t

Data Fields

- UART_Type ∗ base
  UART base address.
- uint32_t srcclk
  UART source clock in Hz.
- uint32_t baudrate
  Desired communication speed.
- uart_parity_mode_t parity
  Parity setting.

- uart_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.

30.4.3 Macro Definition Documentation

30.4.3.1 #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))

30.4.4 Function Documentation

30.4.4.1 int UART_RTOS_Init (uart_rtos_handle_t * *handle,* uart_handle_t * *t_handle,* const uart_rtos_config_t * *cfg*)

Parameters

handle	The RTOS UART handle, the pointer to an allocated space for RTOS context.
t_handle	The pointer to the allocated space to store the transactional layer internal state.
cfg	The pointer to the parameters required to configure the UART after initialization.

Returns

0 succeed; otherwise fail.

30.4.4.2 int UART_RTOS_Deinit (uart_rtos_handle_t * *handle*)

This function deinitializes the UART module, sets all register values to reset value, and frees the resources.

Parameters

handle	The RTOS UART handle.

30.4.4.3 int UART_RTOS_Send (uart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length*)

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

handle	The RTOS UART handle.
buffer	The pointer to the buffer to send.
length	The number of bytes to send.

30.4.4.4 int UART_RTOS_Receive (uart_rtos_handle_t ∗ *handle,* uint8_t ∗ *buffer,* uint32_t *length,* size_t ∗ *received*)

This function receives data from UART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

Parameters

handle	The RTOS UART handle.
buffer	The pointer to the buffer 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.

30.5 UART CMSIS Driver

This section describes the programming interface of the UART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage methord see http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The UART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

30.5.1 UART CMSIS Driver

30.5.1.1 UART Send/receive using an interrupt method

```
/* UART  callback */
void UART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txBufferFull = false;
        txOnGoing = false;
    }

    if (event == ARM_USART_EVENT_RECEIVE_COMPLETE)
    {
        rxBufferEmpty = false;
        rxOnGoing = false;
    }
}
Driver_USART0.Initialize(UART_Callback);
Driver_USART0.PowerControl(ARM_POWER_FULL);
/* Send g_tipString out. */
txOnGoing = true;
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

30.5.1.2 UART Send/Receive using the DMA method

```
/* UART  callback */
void UART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txBufferFull = false;
        txOnGoing = false;
    }

    if (event == ARM_USART_EVENT_RECEIVE_COMPLETE)
```

```
    {
        rxBufferEmpty = false;
        rxOnGoing = false;
    }
}

Driver_USART0.Initialize(UART_Callback);
DMAMGR_Init();
Driver_USART0.PowerControl(ARM_POWER_FULL);

/* Send g_tipString out. */
txOnGoing = true;

Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

Chapter 31
VREF: Voltage Reference Driver

31.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar Voltage Reference (VREF) block of MCUXpresso SDK devices.

The Voltage Reference(VREF) supplies an accurate 1.2 V voltage output that can be trimmed in 0.5 mV steps. VREF can be used in applications to provide a reference voltage to external devices and to internal analog peripherals, such as the ADC, DAC, or CMP. The voltage reference has operating modes that provide different levels of supply rejection and power consumption.

31.2 VREF functional Operation

To configure the VREF driver, configure vref_config_t structure in one of two ways.

1. Use the VREF_GetDefaultConfig() function.
2. Set the parameter in the vref_config_t structure.

To initialize the VREF driver, call the VREF_Init() function and pass a pointer to the vref_config_t structure.

To de-initialize the VREF driver, call the VREF_Deinit() function.

31.3 Typical use case and example

This example shows how to generate a reference voltage by using the VREF module.

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

Data Structures

- struct vref_config_t
  The description structure for the VREF module. More...

Enumerations

- enum vref_buffer_mode_t {
  kVREF_ModeBandgapOnly = 0U,
  kVREF_ModeHighPowerBuffer = 1U,
  kVREF_ModeLowPowerBuffer = 2U }
  VREF modes.

Driver version

- #define FSL_VREF_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  Version 2.1.2.

VREF functional operation

- void VREF_Init (VREF_Type *base, const vref_config_t *config)
  Enables the clock gate and configures the VREF module according to the configuration structure.
- void VREF_Deinit (VREF_Type *base)
  Stops and disables the clock for the VREF module.
- void VREF_GetDefaultConfig (vref_config_t *config)
  Initializes the VREF configuration structure.
- void VREF_SetTrimVal (VREF_Type *base, uint8_t trimValue)
  Sets a TRIM value for the reference voltage.
- static uint8_t VREF_GetTrimVal (VREF_Type *base)
  Reads the value of the TRIM meaning output voltage.
- void VREF_SetLowReferenceTrimVal (VREF_Type *base, uint8_t trimValue)
  Sets the TRIM value for the low voltage reference.
- static uint8_t VREF_GetLowReferenceTrimVal (VREF_Type *base)
  Reads the value of the TRIM meaning output voltage.

31.4 Data Structure Documentation

31.4.1 struct vref_config_t

Data Fields

- vref_buffer_mode_t bufferMode
  Buffer mode selection.
- bool enableLowRef
  Set VREFL (0.4 V) reference buffer enable or disable.
- bool enableExternalVoltRef
  Select external voltage reference or not (internal)

31.5 Macro Definition Documentation

31.5.1 #define FSL_VREF_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

31.6 Enumeration Type Documentation

31.6.1 enum vref_buffer_mode_t

Enumerator

kVREF_ModeBandgapOnly Bandgap on only, for stabilization and startup.
kVREF_ModeHighPowerBuffer High-power buffer mode enabled.
kVREF_ModeLowPowerBuffer Low-power buffer mode enabled.

31.7 Function Documentation

31.7.1 void VREF_Init (VREF_Type * *base*, const vref_config_t * *config*)

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up vref_config_-

t parameters and how to call the VREF_Init function by passing in these parameters. This is an example.

```
*   vref_config_t vrefConfig;
*   vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
*   vrefConfig.enableExternalVoltRef = false;
*   vrefConfig.enableLowRef = false;
*   VREF_Init(VREF, &vrefConfig);
*
```

Parameters

base	VREF peripheral address.
config	Pointer to the configuration structure.

31.7.2 void VREF_Deinit (VREF_Type * *base*)

This function should be called to shut down the module. This is an example.

```
*   vref_config_t vrefUserConfig;
*   VREF_Init(VREF);
*   VREF_GetDefaultConfig(&vrefUserConfig);
*   ...
*   VREF_Deinit(VREF);
*
```

Parameters

base	VREF peripheral address.

31.7.3 void VREF_GetDefaultConfig (vref_config_t * *config*)

This function initializes the VREF configuration structure to default values. This is an example.

```
*   vrefConfig->bufferMode = kVREF_ModeHighPowerBuffer;
*   vrefConfig->enableExternalVoltRef = false;
*   vrefConfig->enableLowRef = false;
*
```

Parameters

config	Pointer to the initialization structure.

31.7.4 void VREF_SetTrimVal (VREF_Type * *base*, uint8_t *trimValue*)

This function sets a TRIM value for the reference voltage. Note that the TRIM value maximum is 0x3F.

Parameters

base	VREF peripheral address.
trimValue	Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

31.7.5 static uint8_t VREF_GetTrimVal (VREF_Type * *base*) [inline], [static]

This function gets the TRIM value from the TRM register.

Parameters

base	VREF peripheral address.

Returns

Six-bit value of trim setting.

31.7.6 void VREF_SetLowReferenceTrimVal (VREF_Type * *base*, uint8_t *trimValue*)

This function sets the TRIM value for low reference voltage. Note the following.

- The TRIM value maximum is 0x05U
- The values 111b and 110b are not valid/allowed.

Parameters

base	VREF peripheral address.

trimValue	Value of the trim register to set output low reference voltage (maximum 0x05U (3-bit)).

31.7.7 static uint8_t VREF_GetLowReferenceTrimVal (VREF_Type * *base*) [inline], [static]

This function gets the TRIM value from the VREFL_TRM register.

Parameters

base	VREF peripheral address.

Returns

Three-bit value of the trim setting.

Chapter 32
WDOG: Watchdog Timer Driver

32.1 Overview

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

32.2 Typical use case

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

Data Structures

- struct wdog_work_mode_t
  Defines WDOG work mode. More...
- struct wdog_config_t
  Describes WDOG configuration structure. More...
- struct wdog_test_config_t
  Describes WDOG test mode configuration structure. More...

Enumerations

- enum wdog_clock_source_t {
  kWDOG_LpoClockSource = 0U,
  kWDOG_AlternateClockSource = 1U }
  Describes WDOG clock source.
- enum wdog_clock_prescaler_t {
  kWDOG_ClockPrescalerDivide1 = 0x0U,
  kWDOG_ClockPrescalerDivide2 = 0x1U,
  kWDOG_ClockPrescalerDivide3 = 0x2U,
  kWDOG_ClockPrescalerDivide4 = 0x3U,
  kWDOG_ClockPrescalerDivide5 = 0x4U,
  kWDOG_ClockPrescalerDivide6 = 0x5U,
  kWDOG_ClockPrescalerDivide7 = 0x6U,
  kWDOG_ClockPrescalerDivide8 = 0x7U }
  Describes the selection of the clock prescaler.
- enum wdog_test_mode_t {
  kWDOG_QuickTest = 0U,
  kWDOG_ByteTest = 1U }
  Describes WDOG test mode.
- enum wdog_tested_byte_t {
  kWDOG_TestByte0 = 0U,
  kWDOG_TestByte1 = 1U,
  kWDOG_TestByte2 = 2U,

kWDOG_TestByte3 = 3U }
Describes WDOG tested byte selection in byte test mode.
- enum _wdog_interrupt_enable_t { kWDOG_InterruptEnable = WDOG_STCTRLH_IRQRSTEN_-MASK }
  WDOG interrupt configuration structure, default settings all disabled.
- enum _wdog_status_flags_t {
  kWDOG_RunningFlag = WDOG_STCTRLH_WDOGEN_MASK,
  kWDOG_TimeoutFlag = WDOG_STCTRLL_INTFLG_MASK }
  WDOG status flags.

Driver version

- #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  Defines WDOG driver version 2.0.1.

Unlock sequence

- #define WDOG_FIRST_WORD_OF_UNLOCK (0xC520U)
  First word of unlock sequence.
- #define WDOG_SECOND_WORD_OF_UNLOCK (0xD928U)
  Second word of unlock sequence.

Refresh sequence

- #define WDOG_FIRST_WORD_OF_REFRESH (0xA602U)
  First word of refresh sequence.
- #define WDOG_SECOND_WORD_OF_REFRESH (0xB480U)
  Second word of refresh sequence.

WDOG Initialization and De-initialization

- void WDOG_GetDefaultConfig (wdog_config_t ∗config)
  Initializes the WDOG configuration structure.
- void WDOG_Init (WDOG_Type ∗base, const wdog_config_t ∗config)
  Initializes the WDOG.
- void WDOG_Deinit (WDOG_Type ∗base)
  Shuts down the WDOG.
- void WDOG_SetTestModeConfig (WDOG_Type ∗base, wdog_test_config_t ∗config)
  Configures the WDOG functional test.

WDOG Functional Operation

- static void WDOG_Enable (WDOG_Type ∗base)
  Enables the WDOG module.
- static void WDOG_Disable (WDOG_Type ∗base)
  Disables the WDOG module.
- static void WDOG_EnableInterrupts (WDOG_Type ∗base, uint32_t mask)
  Enables the WDOG interrupt.
- static void WDOG_DisableInterrupts (WDOG_Type ∗base, uint32_t mask)
  Disables the WDOG interrupt.

- uint32_t WDOG_GetStatusFlags (WDOG_Type ∗base)
  Gets the WDOG all status flags.
- void WDOG_ClearStatusFlags (WDOG_Type ∗base, uint32_t mask)
  Clears the WDOG flag.
- static void WDOG_SetTimeoutValue (WDOG_Type ∗base, uint32_t timeoutCount)
  Sets the WDOG timeout value.
- static void WDOG_SetWindowValue (WDOG_Type ∗base, uint32_t windowValue)
  Sets the WDOG window value.
- static void WDOG_Unlock (WDOG_Type ∗base)
  Unlocks the WDOG register written.
- void WDOG_Refresh (WDOG_Type ∗base)
  Refreshes the WDOG timer.
- static uint16_t WDOG_GetResetCount (WDOG_Type ∗base)
  Gets the WDOG reset count.
- static void WDOG_ClearResetCount (WDOG_Type ∗base)
  Clears the WDOG reset count.

32.3 Data Structure Documentation

32.3.1 struct wdog_work_mode_t

Data Fields

- bool enableStop
  Enables or disables WDOG in stop mode.
- bool enableDebug
  Enables or disables WDOG in debug mode.

32.3.2 struct wdog_config_t

Data Fields

- bool enableWdog
  Enables or disables WDOG.
- wdog_clock_source_t clockSource
  Clock source select.
- wdog_clock_prescaler_t prescaler
  Clock prescaler value.
- wdog_work_mode_t workMode
  Configures WDOG work mode in debug stop and wait mode.
- bool enableUpdate
  Update write-once register enable.
- bool enableInterrupt
  Enables or disables WDOG interrupt.
- bool enableWindowMode
  Enables or disables WDOG window mode.
- uint32_t windowValue
  Window value.

- uint32_t timeoutValue
  Timeout value.

32.3.3 struct wdog_test_config_t

Data Fields

- wdog_test_mode_t testMode
  Selects test mode.
- wdog_tested_byte_t testedByte
  Selects tested byte in byte test mode.
- uint32_t timeoutValue
  Timeout value.

32.4 Macro Definition Documentation

32.4.1 #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

32.5 Enumeration Type Documentation

32.5.1 enum wdog_clock_source_t

Enumerator

kWDOG_LpoClockSource WDOG clock sourced from LPO.
kWDOG_AlternateClockSource WDOG clock sourced from alternate clock source.

32.5.2 enum wdog_clock_prescaler_t

Enumerator

kWDOG_ClockPrescalerDivide1 Divided by 1.
kWDOG_ClockPrescalerDivide2 Divided by 2.
kWDOG_ClockPrescalerDivide3 Divided by 3.
kWDOG_ClockPrescalerDivide4 Divided by 4.
kWDOG_ClockPrescalerDivide5 Divided by 5.
kWDOG_ClockPrescalerDivide6 Divided by 6.
kWDOG_ClockPrescalerDivide7 Divided by 7.
kWDOG_ClockPrescalerDivide8 Divided by 8.

32.5.3 enum wdog_test_mode_t

Enumerator

kWDOG_QuickTest Selects quick test.
kWDOG_ByteTest Selects byte test.

32.5.4 enum wdog_tested_byte_t

Enumerator

kWDOG_TestByte0 Byte 0 selected in byte test mode.
kWDOG_TestByte1 Byte 1 selected in byte test mode.
kWDOG_TestByte2 Byte 2 selected in byte test mode.
kWDOG_TestByte3 Byte 3 selected in byte test mode.

32.5.5 enum _wdog_interrupt_enable_t

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

Enumerator

kWDOG_InterruptEnable WDOG timeout generates an interrupt before reset.

32.5.6 enum _wdog_status_flags_t

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

kWDOG_RunningFlag Running flag, set when WDOG is enabled.
kWDOG_TimeoutFlag Interrupt flag, set when an exception occurs.

32.6 Function Documentation

32.6.1 void WDOG_GetDefaultConfig (wdog_config_t * *config*)

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

```
*   wdogConfig->enableWdog = true;
*   wdogConfig->clockSource = kWDOG_LpoClockSource;
*   wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
*   wdogConfig->workMode.enableWait = true;
*   wdogConfig->workMode.enableStop = false;
*   wdogConfig->workMode.enableDebug = false;
*   wdogConfig->enableUpdate = true;
*   wdogConfig->enableInterrupt = false;
*   wdogConfig->enableWindowMode = false;
*   wdogConfig->windowValue = 0;
*   wdogConfig->timeoutValue = 0xFFFFU;
*
```

Parameters

config	Pointer to the WDOG configuration structure.

See Also

wdog_config_t

32.6.2 void WDOG_Init (WDOG_Type ∗ *base*, const wdog_config_t ∗ *config*)

This function initializes the WDOG. When called, the WDOG runs according to the configuration. To reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

This is an example.

```
*   wdog_config_t config;
*   WDOG_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   config.enableUpdate = true;
*   WDOG_Init(wdog_base,&config);
*
```

Parameters

base	WDOG peripheral base address
config	The configuration of WDOG

32.6.3 void WDOG_Deinit (WDOG_Type ∗ *base*)

This function shuts down the WDOG. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which indicates that the register update is enabled.

32.6.4 void WDOG_SetTestModeConfig (WDOG_Type * *base*, wdog_test_config_t * *config*)

This function is used to configure the WDOG functional test. When called, the WDOG goes into test mode and runs according to the configuration. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

This is an example.

```
*   wdog_test_config_t test_config;
*   test_config.testMode = kWDOG_QuickTest;
*   test_config.timeoutValue = 0xfffffu;
*   WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

base	WDOG peripheral base address
config	The functional test configuration of WDOG

32.6.5 static void WDOG_Enable (WDOG_Type * *base*) [inline], [static]

This function write value into WDOG_STCTRLH register to enable the WDOG, it is a write-once register, make sure that the WCT window is still open and this register has not been written in this WCT while this function is called.

Parameters

base	WDOG peripheral base address

32.6.6 static void WDOG_Disable (WDOG_Type * *base*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG. It is a write-once register. Ensure that the WCT window is still open and that register has not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address

32.6.7 static void WDOG_EnableInterrupts (WDOG_Type * *base*, uint32_t *mask*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to enable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address
mask	The interrupts to enable The parameter can be combination of the following source if defined. • kWDOG_InterruptEnable

32.6.8 static void WDOG_DisableInterrupts (WDOG_Type * *base*, uint32_t *mask*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address
mask	The interrupts to disable The parameter can be combination of the following source if defined. • kWDOG_InterruptEnable

32.6.9 uint32_t WDOG_GetStatusFlags (WDOG_Type * *base*)

This function gets all status flags.

This is an example for getting the Running Flag.

```
*   uint32_t status;
*   status = WDOG_GetStatusFlags (wdog_base) &
      kWDOG_RunningFlag;
```

```
*
```

Parameters

base	WDOG peripheral base address

Returns

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

See Also

_wdog_status_flags_t
- true: a related status flag has been set.
- false: a related status flag is not set.

32.6.10 void WDOG_ClearStatusFlags (WDOG_Type ∗ *base*, uint32_t *mask*)

This function clears the WDOG status flag.

This is an example for clearing the timeout (interrupt) flag.

```
*   WDOG_ClearStatusFlags(wdog_base,kWDOG_TimeoutFlag);
*
```

Parameters

base	WDOG peripheral base address
mask	The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

32.6.11 static void WDOG_SetTimeoutValue (WDOG_Type ∗ *base*, uint32_t *timeoutCount*) [inline], [static]

This function sets the timeout value. It should be ensured that the time-out value for the WDOG is always greater than 2xWCT time + 20 bus clock cycles. This function writes a value into WDOG_TOVALH and WDOG_TOVALL registers which are wirte-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address
timeoutCount	WDOG timeout value; count of WDOG clock tick.

32.6.12 static void WDOG_SetWindowValue (WDOG_Type ∗ *base,* uint32_t *windowValue*) [inline], [static]

This function sets the WDOG window value. This function writes a value into WDOG_WINH and W-DOG_WINL registers which are wirte-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address
windowValue	WDOG window value.

32.6.13 static void WDOG_Unlock (WDOG_Type ∗ *base*) [inline], [static]

This function unlocks the WDOG register written. Before starting the unlock sequence and following configuration, disable the global interrupts. Otherwise, an interrupt may invalidate the unlocking sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

base	WDOG peripheral base address

32.6.14 void WDOG_Refresh (WDOG_Type ∗ *base*)

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

base	WDOG peripheral base address

32.6.15 static uint16_t WDOG_GetResetCount (WDOG_Type * *base*) [inline], [static]

This function gets the WDOG reset count value.

Parameters

base	WDOG peripheral base address

Returns

WDOG reset count value.

32.6.16 static void WDOG_ClearResetCount (WDOG_Type * *base*) [inline], [static]

This function clears the WDOG reset count value.

Parameters

base	WDOG peripheral base address

Chapter 33
XBAR: Inter-Peripheral Crossbar Switch

33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBAR) block of MCUXpresso SDK devices.

The XBAR peripheral driver configures the XBAR (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBAR module.

XBAR driver has two parts:

- Signal connection - This part interconnects input and output signals.
- Active edge feature - Some of the outputs provides active edge detection. If an active edge occurs, an interrupt or a DMA request can be called. APIs handle user callbacks for the interrupts. The driver also includes API for clearing and reading status bit.

33.2 Function groups

33.2.1 XBAR Initialization

To initialize the XBAR driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBAR module is initialized by calling the XBAR_Init() function.

33.2.2 Call diagram

1. Call the "XBAR_Init()" function to initialize the XBAR module.
2. Optionally, call the "XBAR_SetSignalsConnection()" function to Set connection between the selected XBAR_IN[∗] input and the XBAR_OUT[∗] output signal. It connects the XBAR input to the selected XBAR output. A configuration structure of the "xbar_input_signal_t" type and "xbar_-output_signal_t" type is required.
3. Call the "XBAR_SetOutputSignalConfig" function to set the active edge features, such interrupts or DMA requests. A configuration structure of the "xbar_control_config_t" type is required to point to structure that keeps configuration of control register.
4. Finally, the XBAR works properly.

33.3 Typical use case

Data Structures

- struct xbar_control_config_t
  Defines the configuration structure of the XBAR control register. More...

Enumerations

- enum xbar_active_edge_t {
  kXBAR_EdgeNone = 0U,
  kXBAR_EdgeRising = 1U,
  kXBAR_EdgeFalling = 2U,
  kXBAR_EdgeRisingAndFalling = 3U }
  XBAR active edge for detection.
- enum xbar_request_t {
  kXBAR_RequestDisable = 0U,
  kXBAR_RequestDMAEnable = 1U,
  kXBAR_RequestInterruptEnalbe = 2U }
  Defines the XBAR DMA and interrupt configurations.
- enum xbar_status_flag_t { kXBAR_EdgeDetectionOut0 }
  XBAR status flags.

XBAR functional Operation

- void XBAR_Init (XBAR_Type ∗base)
  Initializes the XBAR modules.
- void XBAR_Deinit (XBAR_Type ∗base)
  Shutdown the XBAR modules.
- void XBAR_SetSignalsConnection (XBAR_Type ∗base, xbar_input_signal_t input, xbar_output_signal_t output)
  Set connection between the selected XBAR_IN[∗] input and the XBAR_OUT[∗] output signal.
- void XBAR_ClearStatusFlags (XBAR_Type ∗base, uint32_t mask)
  Clears the edge detection status flags of relative mask.
- uint32_t XBAR_GetStatusFlags (XBAR_Type ∗base)
  Gets the active edge detection status.
- void XBAR_SetOutputSignalConfig (XBAR_Type ∗base, xbar_output_signal_t output, const xbar_control_config_t ∗controlConfig)
  Configures the XBAR control register.

33.4 Data Structure Documentation

33.4.1 struct xbar_control_config_t

This structure keeps the configuration of XBAR control register for one output. Control registers are available only for a few outputs. Not every XBAR module has control registers.

Data Fields

- xbar_active_edge_t activeEdge
  Active edge to be detected.
- xbar_request_t requestType
  Selects DMA/Interrupt request.

Field Documentation

(1) xbar_active_edge_t xbar_control_config_t::activeEdge

(2) xbar_request_t xbar_control_config_t::requestType

33.5 Enumeration Type Documentation

33.5.1 enum xbar_active_edge_t

Enumerator

kXBAR_EdgeNone Edge detection status bit never asserts.
kXBAR_EdgeRising Edge detection status bit asserts on rising edges.
kXBAR_EdgeFalling Edge detection status bit asserts on falling edges.
kXBAR_EdgeRisingAndFalling Edge detection status bit asserts on rising and falling edges.

33.5.2 enum xbar_request_t

Enumerator

kXBAR_RequestDisable Interrupt and DMA are disabled.
kXBAR_RequestDMAEnable DMA enabled, interrupt disabled.
kXBAR_RequestInterruptEnalbe Interrupt enabled, DMA disabled.

33.5.3 enum xbar_status_flag_t

This provides constants for the XBAR status flags for use in the XBAR functions.

Enumerator

kXBAR_EdgeDetectionOut0 XBAR_OUT0 active edge interrupt flag, sets when active edge detected.

33.6 Function Documentation

33.6.1 void XBAR_Init (XBAR_Type * *base*)

This function un-gates the XBAR clock.

Parameters

base	XBAR peripheral address.

33.6.2 void XBAR_Deinit (XBAR_Type * *base*)

This function disables XBAR clock.

Parameters

base	XBAR peripheral address.

33.6.3 void XBAR_SetSignalsConnection (XBAR_Type * *base,* xbar_input_signal_t *input,* xbar_output_signal_t *output*)

This function connects the XBAR input to the selected XBAR output. If more than one XBAR module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBAR_SetSignalsConnection(XBAR, kXBAR_InputTMR_CH0_Output, kXBAR_OutputXB_DMA_INT2
      );
```

Parameters

base	XBAR peripheral address
input	XBAR input signal.
output	XBAR output signal.

33.6.4 void XBAR_ClearStatusFlags (XBAR_Type * *base,* uint32_t *mask*)

Parameters

base	XBAR peripheral address

mask	the status flags to clear.

33.6.5 uint32_t XBAR_GetStatusFlags (XBAR_Type ∗ *base*)

This function gets the active edge detect status of all XBAR_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBAR_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Example:

```
uint32_t status;

status = XBAR_GetStatusFlags(XBAR);
```

Parameters

base	XBAR peripheral address.

Returns

the mask of these status flag bits.

33.6.6 void XBAR_SetOutputSignalConfig (XBAR_Type ∗ *base*, xbar_output_signal_t *output*, const xbar_control_config_t ∗ *controlConfig*)

This function configures an XBAR control register. The active edge detection and the DMA/IRQ function on the corresponding XBAR output can be set.

Example:

```
xbar_control_config_t userConfig;
userConfig.activeEdge = kXBAR_EdgeRising;
userConfig.requestType = kXBAR_RequestInterruptEnalbe;
XBAR_SetOutputSignalConfig(XBAR, kXBAR_OutputXB_DMA_INT0, &userConfig);
```

Parameters

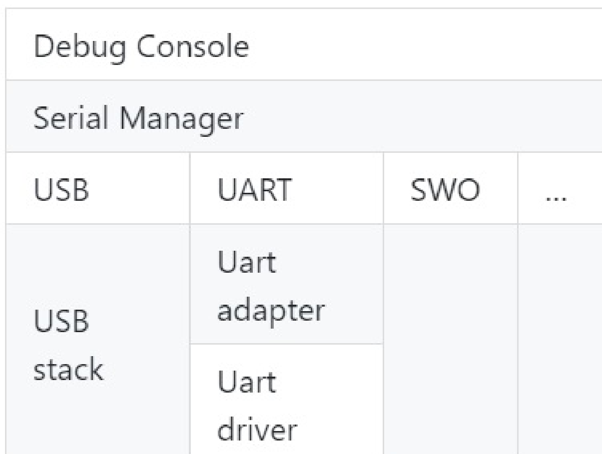
base	XBAR peripheral address
output	XBAR output number.
controlConfig	Pointer to structure that keeps configuration of control register.

Chapter 34
Debug Console

34.1 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

34.2 Function groups

34.2.1 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);
```

34.2.2 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 */
```

34.2.3 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.

34.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

Print out failure messages using MCUXpresso SDK __assert_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
       , line, func);
    for (;;)
    {}
}
```

Note:

To use 'printf' and 'scanf' for GNUC Base, add file **'fsl_sbrk.c'** in path: **..\{package}\devices\{subset}\utilities\fsl-_sbrk.c** to your project.

Modules

- Semihosting

Macros

- #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U
  Definition select redirect toolchain printf, scanf to uart or not.
- #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U
  Select SDK version printf, scanf.
- #define DEBUGCONSOLE_DISABLE 2U
  Disable debugconsole function.
- #define SDK_DEBUGCONSOLE 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.

34.4 Macro Definition Documentation

34.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

34.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

34.4.3 #define DEBUGCONSOLE_DISABLE 2U

34.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

34.4.5 #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.

34.5 Function Documentation

34.5.1 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

34.5.2 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.

34.5.3 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.

34.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

34.5.5 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.

34.5.6 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.

34.5.7 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.

34.5.8 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.

34.5.9 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.

34.5.10 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.

34.5.11 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.

34.5.12 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.

34.5.13 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

34.5.14 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 "".

34.6 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as printf() and scanf(), to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

34.6.1 Guide Semihosting for IAR

NOTE: After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the SDK_DEBUGCONSOLE is DEBUGCONSOLE_REDIRECT_-TO_TOOLCHAIN.

Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

Step 3: Starting semihosting

1. Choose "Semihosting_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting. Please Make sure the SDK_DEBUGCONSOLE_UART is not defined in project settings.
4. Start the project by choosing Project>Download and Debug.
5. Choose View>Terminal I/O to display the output from the I/O operations.

34.6.2 Guide Semihosting for Keil µVision

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

34.6.3 Guide Semihosting for MCUXpresso IDE

Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK_DEBUGCONSOLE=0, if set SDK_DEBUGCONSOLE=1, the log will be redirect to the UART.

Step 2: Building the project

1. Compile and link the project.

Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

34.6.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
   - "Host Name (or IP address)" : localhost
   - "Port" :2333
   - "Connection type" : Telet.
   - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

Add to "CMakeLists.txt"

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__heap_size__=0x2000")

Step 2: Building the project

1. Change "CMakeLists.txt":
   Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=nano.specs")"
   to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=rdimon.specs")"
   Replace paragraph
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
   To
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
   Remove
   target_link_libraries(semihosting_ARMGCC.elf debug nosys)
2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

2. After the setting, press "enter". The PuTTY window now shows the printf() output.

Chapter 35
Notification Framework

35.1 Overview

This section describes the programming interface of the Notifier driver.

35.2 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.

35.3 Data Structure Documentation

35.3.1 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

35.3.2 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

35.3.3 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

35.4 Typedef Documentation

35.4.1 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.

35.4.2 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.

35.4.3 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.

35.5 Enumeration Type Documentation

35.5.1 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.

35.5.2 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.

35.5.3 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.

35.5.4 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.

35.6 Function Documentation

35.6.1 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.

35.6.2 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.

35.6.3 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 36
Shell

36.1 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.

36.2 Function groups

36.2.1 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>");
```

36.2.2 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.

36.2.3 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.

36.3 Data Structure Documentation

36.3.1 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

36.4 Macro Definition Documentation

36.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

36.4.2 #define SHELL_AUTO_COMPLETE (1U)

36.4.3 #define SHELL_BUFFER_SIZE (64U)

36.4.4 #define SHELL_MAX_ARGS (8U)

36.4.5 #define SHELL_HISTORY_COUNT (3U)

36.4.6 #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

36.4.7 #define SHELL_USE_COMMON_TASK (0U)

36.4.8 #define SHELL_TASK_PRIORITY (2U)

36.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

36.4.10 #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.

36.4.11 #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 SH-ELL_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.

36.4.12 #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".

36.5 Typedef Documentation

36.5.1 typedef shell_status_t(∗ cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char ∗∗argv)

36.6 Enumeration Type Documentation

36.6.1 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.

36.7 Function Documentation

36.7.1 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.

36.7.2 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.

36.7.3 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.

36.7.4 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.

36.7.5 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.

36.7.6 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.

36.7.7 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.

36.7.8 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.

36.7.9 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

shellHandle	The shell module handle pointer.

Returns

NULL.

36.7.10 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.

36.7.11 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 37
Serial Manager

37.1 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 Uart

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 ∗pfCallback)
  This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.

37.2 Data Structure Documentation

37.2.1 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.

37.2.2 struct serial_manager_callback_message_t

Data Fields

- uint8_t ∗ buffer
  Transferred buffer.
- uint32_t length
  Transferred data length.

37.3 Macro Definition Documentation

37.3.1 #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)

37.3.2 #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)

37.3.3 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

Macro to determine whether use common task.

37.3.4 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_-SIZE_TEMP + 12U)

37.3.5 #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.

37.3.6 #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.

37.3.7 #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.

37.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

37.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

37.4 Enumeration Type Documentation

37.4.1 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.

37.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.
kSerialManager_Blocking Blocking handle.

37.4.3 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.

37.5 Function Documentation

37.5.1 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.

37.5.2 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.

37.5.3 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);
*
```

37.5.4 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.

37.5.5 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));
*
```

37.5.6 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.

37.5.7 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.

37.5.8 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.

37.5.9 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.

37.5.10 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.

37.5.11 void SerialManager_SetLowpowerCriticalCb (const serial_manager_-lowpower_critical_CBs_t ∗ *pfCallback*)

Parameters

pfCallback	Pointer to the function structure used to allow/disable lowpower.

37.6 Serial Port Uart

37.6.1 Overview

Macros

- #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_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

37.6.2 Enumeration Type Documentation

37.6.2.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

37.6.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support